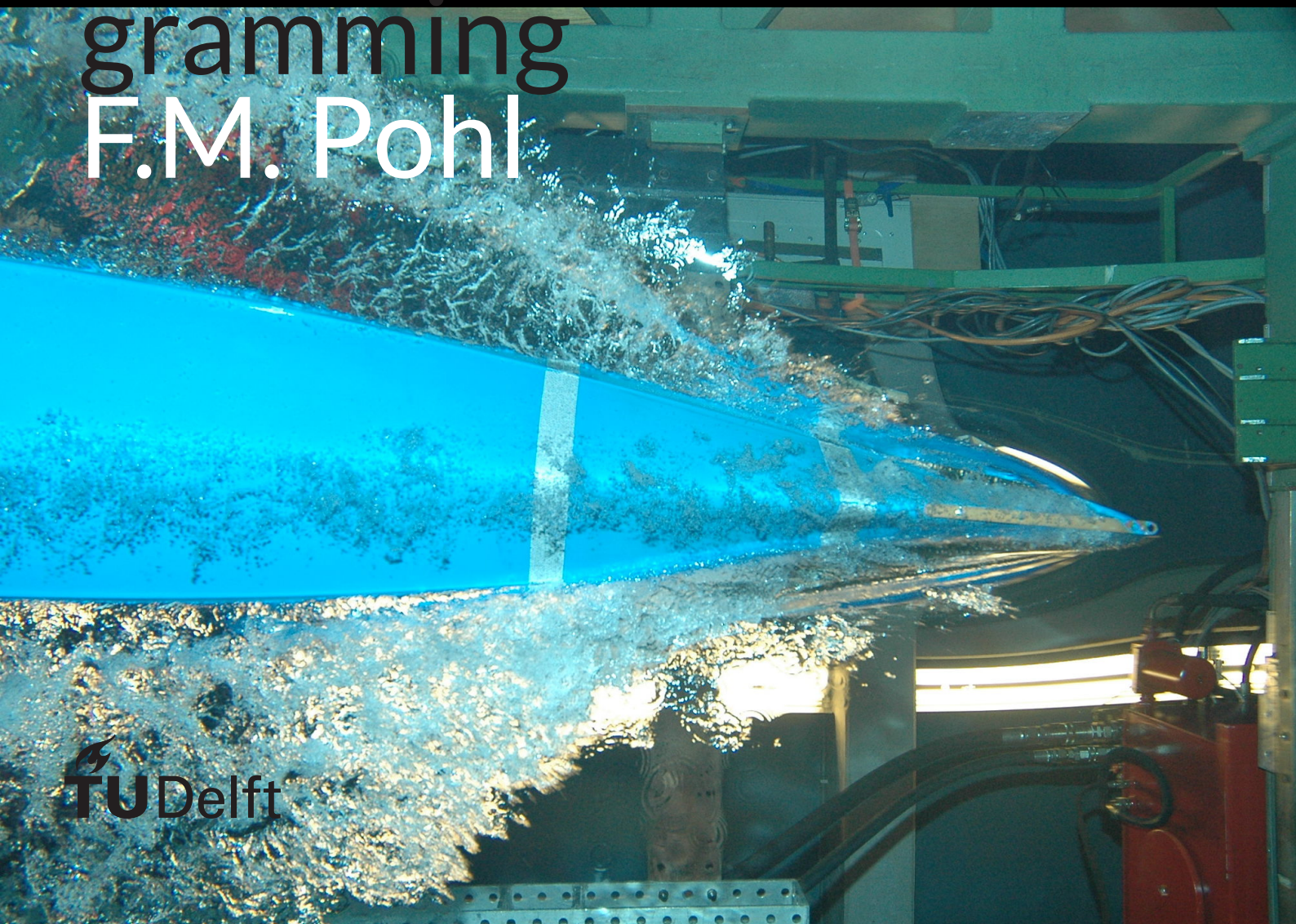
Adaptive-Critic Designs For Aircraft Control

A comparative study between Heuristic Dynamic Programming and Dual Heuristic Programming

F.M. Pohl



TU Delft

MSC THESIS

TU Delft

Faculty of Aerospace Engineering

Adaptive-Critic Designs for Aircraft Control

F.M. Pohl

Adaptive-Critic Designs For Aircraft Control

A comparative study between Heuristic Dynamic Programming and Dual Heuristic Programming

by

F.M. Pohl

to obtain the degree of Master of Science in Control and Simulations
at Delft University of Technology,
defended publicly on Thursday August 31, 2017 at 14:00.

Supervisor:	Dr. ir. E. van Kampen,	TU Delft
Thesis committee:	Dr. Q.P. Chu,	TU Delft
	Dr. ir. D. Dirkx,	TU Delft

TU Delft

Preface

The past year has been very challenging in a lot of ways and I have learned a lot in those months. In the beginning of this MSc thesis I was striving towards the end of my study, but now that the thesis is finished I got even more curious about the topic and would love to research more. However, it has not always been easy and there were many moments where my ambition vanished and gave way to frustration and demotivation. I am happy that I got through that moments thanks to the help of the people around me.

I would like to thank Dr. Erik-Jan van Kampen for his help and supervision throughout the project. It was a long way, but he has always been patient and a great support, not only intellectually but also personally.

A warm thanks also goes to my mentor at NASA Ames, Dr. Nhan Nguyen. I am very grateful to have been part of this strong research team and for all the experience and knowledge I could gain during my time in the Silicon Valley. Càm o'n ban!

Lastly, I would like to thank my family and friends who have pulled me through this emotional rollercoaster. You guys have taken a major role to the success of this thesis and I would not be at the place where I am right now without you!

F.M. Pohl
Delft, August 2017

Contents

List of Figures

List of Tables

1 Introduction

In 2010, the Intelligent Systems Division of NASA Ames together with Boeing Research & Technology jointly started a research on *Performance Adaptive Aeroelastic Wing systems* (PAAW). The goal was to actively control elastic wing shapes and optimize their performance (Nguyen [43]). Under the study of the Elastically Shaped Future Air Vehicle Concept, the *Various Camber Continuous Trailing Edge Flap* concept (VCCTEF) was born (Lebofsky et al. [30], Nguyen and Urnes Sr. [39]). As the name implies, this concept is a span-wise trailing edge flap system which consists of multiple, semi-independent flap segments (see figure 1.1). In order to prevent flow discontinuity those segments are connected via an elastomer material. The morphing-wing design enables active flutter suppression, prevents unintentional aeroelastic effects such as wing-bending or wing-torsion and gives the opportunity to optimize the aerodynamic performance in any flight envelope. It promises significant fuel savings and improved stability.

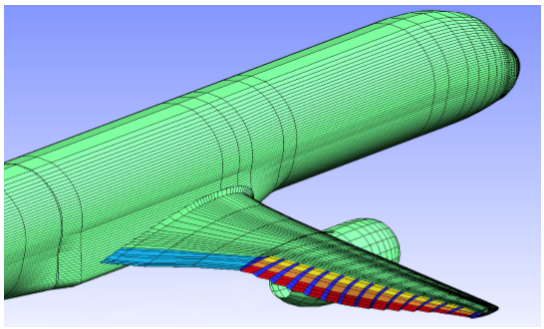


Figure 1.1: The Variable Camber Continuous Trailing Edge Flap design

In combination with the hardware, NASA developed the theory of a *Multi-Objective Flight Controller*[46]. However there are several challenges that the controller is facing. First, the additional amount of control surfaces that can be actuated increased to a number of 48 surfaces per wing, which is significantly more when compared to a conventional wing. This increases the probability of malfunctions and as there is a dependency between the segments, a potential failure of a single flap segment immediately effects all other segments and the controller would need to adapt. Second, as the control system is designed and tuned in simulation studies with the use of a complex model, it may fail to generalize to inaccuracies in the real-life application as the accuracy of such model never matches 100% with the true case. Furthermore, a current step in the control design is to linearize the model, as designing controllers for non-linear systems remains a challenging task. Evidently, this introduces additional control inaccuracies.

One potential approach to circumvent these problems is to use Adaptive-Critic control (AC). AC is a promising branch of reinforcement learning algorithms where the controller learns itself the optimal control law (online or offline). It is an adaptive, non-linear control approach that does not require a complex model of the system a priori.

1.1. Adaptive-Critic Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning and provides a framework that combines mathematical optimization methods with the theory of classical conditioning in psychology. An agent (decision-maker) learns itself an optimal behavior by interacting with the environment (Sutton and Barto [60]). Every interaction is directly evaluated by a numerical reward (stimulus) and depending on how good or bad the actions are, the reward turns out to be more positive or negative. In other words, each action is either *reinforced* or *punished*. By some means, the learning process needs to be memorized by the agent. This is done by the *value function*, which stores the relationship between the behavior and the rewards, while the behavior is determined by the *policy*. Each time-step, the agent decides on an action based on the policy and the state it is located in. Performing that action, the environment transitions to a new state and the learning agent receives a reward as an evaluative feedback. Based on that feedback, the agent updates its value function accordingly in order to make a better decision the next time it is visiting the same state. Eventually, the goal is to learn a behavior that ensures the maximum sum of rewards in the long run.

In the 1950s, Richard Bellman formulated the Bellman equation that defines the recursive relationship of an optimal value function with respect to the states (Bellman [4]). An early approach to solve this equation was to use *Dynamic Programming* (DP), which is a mathematical optimization method that quantizes the state-space and recursively updates the value for each state in time. DP is one of the first branches in RL. However, the problem is that the computational complexity increases exponentially with size of the state-space and this methods is therefore only feasible in small-scale control tasks. In literature this is referred to as the curse of dimensionality (Bellman [4, 5]).

In the past decades, efforts have been made to overcome this problem by approximating the DP solutions. *Approximate Dynamic Programming* (ADP) or *Adaptive-Critic Designs* (ACD) estimate the solution to the Bellman equation and policy by means of function approximators (Barto et al. [3], Konda and Tsitsiklis [28], Witten [71]). The heuristic-process is split into two components. One component, the so called *critic*, evaluates the current state and estimates the value function. The other component, i.e. the *actor*, maps the state to an (optimal) control law and tries to minimize the value function that is approximated by the critic. By incrementally updating the parameters of the actor- and critic-functions at each time-step, the controller will eventually converge to an optimal policy. The use of function approximators additionally enables to cover the full (continuous) state-action space. Depending on the layout, there are different forms of ACD's (Prokhorov and Wunsch [51]), all of which are based on two basic forms, namely *Heuristic Dynamic Programming* (HDP) and *Dual Heuristic dynamic Programming* (DHP). These two forms will also be the focus of this thesis. HDP approximates the value function and acts according to the minimum cost, while in DHP, the critic approximates the derivative of the cost function (i.e. the gradient) and determines the optimal control- law based on that. A more technical description is given in chapter 2.3.

The adaptive and self-learning properties make ACD's in particular popular for the field of research in control theory. Both methods have been introduced successfully to multiple (highly non-linear) domains already. Ferrari and Stengel [17] applied DHP for flight control on a small business aircraft, making use of existing knowledge by pre-training the AC with a basic controller off-line and then improving the performance by online training. Liu et al. [35] implemented HDP to automotive engine torque control, achieving excellent transient response. One of the first comparisons of HDP and DHP has been conducted by Venayagamoorthy et al. [66], applying both algorithms for controlling a turbogenerator offline. It was concluded that DHP showed better performance with faster rise time and better damping then HDP.

1.2. Problem Statement

The contribution of this research is the practical comparison of HDP and DHP for aircraft control, while keeping in mind their future potential for the VCCTEF design. The main *research objective* is:

***Compare** the performance of **Heuristic Dynamic Programming** and **Dual Heuristic Programming** for **aircraft control**.*

This poses the following main *research question*:

*What are the **theoretical and practical differences** of **Heuristic Dynamic Programming and Dual Heuristic Programming** when applied for **aircraft control**?*

The main research question is further divided into sub-questions to distribute the work into smaller work packages:

1. **How does the VCCTEF concept work?**
   (a) What is the VCCTEF system?
   (b) What is the current control design?
   (c) What are the main challenges controlling the VCCTEF?
   (d) What are the requirements that VCCTEF imposes on the control design?
2. **What are Actor-Critic Methods?**
   (a) What is Reinforcement Learning and the concept of Dynamic Programming?
   (b) What is Approximate Dynamic Programming?
   (c) How are Artificial Neural Networks applied for function approximation?
   (d) What is the algorithmic structure of HDP and DHP?
3. **Which controller (HDP or DHP) works better when be applied for aircraft control?**
   (a) How are the controllers applied for (non-)linear control tasks?
   (b) How to incorporate them into an aircraft control loop?
   (c) How to compare the two ACD designs with respect to performance practically?
4. **As an indication, which ACD design (if at all) is more favorable for aircraft control in combination with the Variable Camber Continuous Trailing Edge Flap System?**

1.3. Research approach

In order to answer the research questions as listed in the previous section, the following steps are taken:

Literature review: First, a detailed literature study is carried out in two domains. The first domain reviews the concept of the VCCTEF architecture to give the reader an overview of the control structure as well as recent developments in control design. It concludes to the challenges that the controls are facing and a performance metric to classify and rate future control designs.
The second part of the literature study covers the theory of reinforcement learning and leads to a detailed view of adaptive critic designs, heuristic dynamic programming and dual heuristic programming. It reveals the theoretical differences of HDP and DHP and summarizes the results of preceding research.

Preliminary analysis: In the preliminary analysis, HDP and DHP are applied on a simple inverted pendulum task. This experiment is often used as a benchmark for advanced control system design, because it is a highly non-linear system and can easily be modified in complexity. The results clearly prove a learning behavior of the controllers and reveal a first trend of the practical differences between the algorithms. Focus of the comparison will be in particular the performance metric that has been established from the literature study.

Implementation for aircraft control: After the algorithms are verified in the preliminary analysis, they are applied on a 2D pitch-tracking control task of a F-16 aircraft. The choice of the aircraft type is rather arbitrary and the extreme flight modes of the aircraft are not explicitly used in the simulations. The experimental study is split into an offline and online learning process. Both RL methods will first learn the baseline reference-following pitch control of the plant offline in multiple episodes and then need to adapt online to changes in plant dynamics. The simulation environment will be a high-fidelity model as provided by Russell [55]. The hyperparameter settings are found by an extensive grid-search. Eventually, the results shall show the potential benefits of one controller with respect to the other and give a tendency on the future potential of ACD's for the VCCTEF design.

The different phases of this thesis work can be seen in table 1.1.

Table 1.1: Outline of thesis work

Project Phase	**Content**	**Research Questions**
Literature Review	The VCCTEF design and it's control requirements	1
	Reinforcement learning and adaptive-critic designs	2
	Theoretical differences between HDP and DHP	2d
Preliminary analysis	HDP and DHP for the inverted pendulum task	3a
Final thesis work	HDP and DHP for F-16 aircraft control	3,4 & Main Research

1.4. Report Outline

The remaining chapters of this thesis are organized as follows. Part I summarizes the most important results for the implementation of HDP and DHP on the F-16 aircraft model in a scientific paper. This paper is standalone and should be read independently with respect to the rest of the report. Readers with few knowledge in the VCCTEF design or reinforcement learning are advised to read Part II of this report before engaging with the paper. Part II reviews the preliminary research, with the literature study and preliminary analysis. It covers the VCCTEF design in chapter 2, as well as the fundamentals in reinforcement learning (chapter 3 and 4). Chapter 4 also includes a more detailed technical description of adaptive critic designs with focus in heuristic dynamic programming and dual heuristic programming. The preliminary analysis explains the inverted pendulum experimental setup and the first results in chapter 5. Part III lists a collection of additional results that have been collected after the preliminary research. They provide additional information to the findings that have been presented in the scientific paper. Eventually, the whole thesis work is concluded in part IV with additional recommendations for future research. Appendix A provides additional information about artificial neural network function approximators, its working principle as well convergence improvements.

I

Scientific Paper

Comparison of Heuristic Dynamic Programming and Dual Heuristic Programming for Aircraft Control

F.M.Pohl *

The Variable Camber Continuous Trailing Edge Flap (VCCTEF) is a novel aircraft control system that intents to prevent undesired aeroelastic deflections by precise lift tailoring along the wing span. However, the unknown dynamics and increased complexity of the new hardware imposes difficulties to establish an optimal controller. One approach is to use Adaptive Critic Designs (ACD). Being part of reinforcement learning techniques, their intelligent and adaptive characteristics provide a fault tolerant solution to non-linear control problems. As a starting point, this paper compares the two fundamental forms of ACD's for aircraft control and evaluates their future potential for the VCCTEF design. The two forms are Heuristic Dynamic programming (HDP) and Dual Heuristic Programming (DHP). In an experimental study, both algorithms are integrated in a F-16 aircraft model. First, the agents are trained offline to learn to control the baseline F-16 dynamics. Then, the aircraft dynamics are changed online and the controllers need to adapt to the new plant dynamics. The results show that DHP has a higher success ratio in the offline learning phase and particularly converges faster to an optimal solution. During online simulations, both algorithms can deal with some changes in the F-16 aircraft dynamics even without adaptation, although HDP reveals more robustness in this case. DHP on the other hand, adapts better to changes in the plant dynamics when online learning is applied.

Nomenclature

α	Angle of attack
$\bar{c}$	Mean aerodynamic chord
η	Learning rate
μ	Momentum factor
δ_{el}	Elevator deflection
γ	Discount factor
π	Policy
θ	Pitch angle
θ_{ref}	Reference pitch angle
C_m	Total aerodynamic moment coefficient about the pitch axis in the body frame
C_{m_q}	Aerodynamic moment coefficient derivative w.r.t. q around the pitch axis
C_Z	Total aerodynamic force coefficient in z direction in the body frame
E_a	Actor error measure
e_a	Actor error
E_c	Critic error measure
e_c	Critic error
M	Mach number
e_c	Critic error
V	Value function
V_T	True airspeed
J	Approximated Value function
J^*	Optimal Value function
λ	Derivative of Value function w.r.t. states

*Graduate Student, Control and Simulation Division, Faculty of Aerospace Engineering

ρ Immediate reward
ϵ Exploration rate
ϵ_0 Initial exploration rate
u Control signal
q Pitch rate
W Artificial neural network weights
b Artificial neural network biases
x System state vector
$x_{c.g.}$ Longitudinal c.g. location

Subscripts
c Critic
a Actor
m Model
t Current time step

I. Introduction

In 2010, the Intelligent Systems Division of NASA Ames together with Boeing Research & Technology jointly started a research on performance adaptive aeroelastic wing systems. The goal was to actively control elastic wing shapes and optimize their aerodynamic performance.[1] As a potential concept they proposed the Variable Camber Continuous Trailing Edge Flap (VCCTEF).[2,3] The design is a span-wise trailing edge flap system which consists of 16 semi-independent flap sections. Each flap section contains additionally three chordwise segments for actively changing the camber of the airfoil and locally shaping the pressure distribution. This enables active flutter suppression, prevents unintentional wing-twist and bending and gives the opportunity to optimize the aerodynamic performance in any flight condition.
In combination with the hardware, NASA developed the theory of a *Multi-Objective Flight Controller*.[4] However, there are several challenges that the controller is facing. First, the additional amount of control surfaces that can be actuated increased to a number of 48 surfaces per wing, which is significantly more than a conventional wing. In that perspective, a failure of a single flap segment can have an immediate effect on all other segments due to the semi-dependency between the elements. This requires the controller to be adaptive. Second, the current control design is tuned in simulation studies with the use of a complex model. Accuracy of such model never coincides 100% with the true plant and the controller may fail to assimilate to new situations in the real-life application. Furthermore, using linear control theory to design the controller, requires the model to be linearized around a number of equilibrium points (trim conditions). Evidently this process introduces additional control inaccuracies. On the other hand, all current nonlinear control solutions such as dynamic inversion or model reference adaptive control require a detailed model of the system a priori. Such model can be computational expensive, especially for systems with fast dynamics and model inaccuracies can quickly lead to instability.[5]

One approach to circumvent these problems is the use of Reinforcement Learning(RL) methods for control. RL is a framework that combines mathematical optimization with the theory of classical conditioning. An agent (decision-maker) learns itself an optimal behavior (policy) by interacting with the environment, while each interaction is either *reinforced* or *punished* by means of a numerical reward (stimulus).[6] This provides the opportunity to learn non-linear control tasks even when the plant dynamics are unknown beforehand. Additionally, the self-learning property makes it possible for the system to adapt online to changing operating conditions, resulting in robust and fault-tolerant control characteristics. However, most approaches in the RL framework are only able to find a solution by (re-)visiting every element in a discrete state-action space of the *Markov Decision Process* (MDP). With that, they soon face the curse of dimensionality and become infeasible for most real-life control tasks.
Actor Critic Designs (ACD), is a branch of RL that approximates the solution to the Bellman equation and policy by means of function approximators.[7–9] The heuristic process is split into two components, namely the *actor* and the *critic*. The actor intents to approximate the optimal control law, while the critic is evaluating the current policy by estimating the sum of rewards that can be perceived in the long run. This approach

not only reduces the complexity of the learning process by splitting the task into multiple components, but also enables 1) a continuous approach to the state-action space of an MDP and 2) a global approximation of the solution.
The two fundamental forms of ACD's are *Heuristic Dynamic Programming* (HDP) and *Dual Heuristic Programming* (DHP). HDP is considered to be the most simple form of ACD, where the critic estimates the value function based on the states and immediate rewards, while the actor maps the state to an optimal control law and tries to minimize the value function that is approximated by the critic.[10] DHP on the other hand, is a more advanced form, estimating the derivative of the value function with respect to the states and approximates the optimal control-law based on that.

Both methods have been introduced successfully to multiple (highly non-linear) domains already. Ferrari et al.[11] applied DHP for flight control on a small business aircraft, making use of existing knowledge by pre-training the ACD with a basic controller offline and then improving the performance by online training. Liu et al.[12] implemented HDP to automotive engine torque control, achieving excellent transient response. One of the first comparisons of HDP and DHP has been conducted by Venayagamoorthy et al.,[13] applying both algorithms for controlling a turbogenerator offline. It was concluded that DHP showed better performance with faster rise time and better damping than HDP.

The contribution of this paper is the practical comparison of HDP and DHP for aircraft control, while keeping in mind their future potential for the VCCTEF design. Both controllers are applied on a 2D pitch-tracking control task of a F-16 aircraft. The choice of the aircraft type is rather arbitrary and the extreme flight modes of the aircraft are not explicitly used in the simulations. Any aircraft type could have been used, but the F-16 contains especially fast acting dynamics and gives a first indicator on how the algorithms perform for aircraft control. The experimental study is split into two phases, namely an offline and online learning phase. In the first phase, Both controllers need to learn to control the baseline model of the F-16 in several trials. The learning process starts from scratch, meaning that the controllers are trained without any prior knowledge in memory. Then, the trained agents are applied in an online simulation where they need to adapt to changes in plant dynamics in real-time.
Section II gives a short overview of adaptive critic reinforcement learning techniques with focus on HDP and DHP. The implementation of both algorithms on the F-16 model is explained in Section III, followed by a detailed description of the training procedure in Section IV. All results of the experimental study are shown in Section V. Finally, Section VI gives a conclusion and recommendations for future research.

II. Adaptive Critic Reinforcement Learning

In the RL framework, the system is learning an optimal behavior by interacting with the environment and receiving a numerical reward for every action it is executing. By some means, the learning process needs to be memorized. This is done by the so called value function V, which stores the relationship between the behavior (usually denoted as π) and the total amount of rewards that can be collected in the long run:

$$V^{\pi}(\mathbf{x}(t)) = \sum_{i=0}^{\infty} \gamma^k \rho(\mathbf{x}(t_k), \mathbf{u}(t_k)) \tag{1}$$

Where $\mathbf{x}$ and $\mathbf{u}$ is the state and action at time-step t_k respectively, $\gamma \in [0\ 1)$ is the discount factor for infinite learning tasks and $\rho(\mathbf{x}, \mathbf{u})$ is the reward function. In the 1950s, Richard Bellman formulated the Bellman equation that defines the recursive relationship of an optimal value function with respect to the states.[14] An early approach to solve this equation was to use *Dynamic Programming* (DP), which is a mathematical optimization method that quantizes the state-space and recursively updates the value for each state in time. DP is one of the first branches in RL. However, the downside is that the state-action space is discrete and the computational complexity increases exponentially with size of the state-space. In literature this is referred to as the curse of dimensionality and the concept is therefore only feasible in small-scale control tasks.[14,15]

Approximate Dynamic Programming methods (ADP), try to overcome the curse of dimensionality by approximating the value function and the optimal policy with the use of parametric functions.[6,16] The key feature is to cover the infinite continuous domain of the state-action space with a finite number of parameters. In that perspective, the reinforcement learning problem is split into two entities, namely the *actor* and

the *critic*.[7,9] The actor is a parametric function that approximates the optimal control-law (i.e. policy π) such that it maps the current state to a control signal u based on its parameters W_a:

$$u(t) = \pi(\mathbf{x}(t), W_a) \tag{2}$$

The critic on the other hand, evaluates the current policy by estimating the corresponding value function:

$$J(\mathbf{x}(t)) = f_c(\mathbf{x}(t); W_c) \approx V^{\pi}(\mathbf{x}) \tag{3}$$

The estimation J is the output of the critic function f_c and depends on the current state-input as well as the critic parameters W_c. The superscript π denotes the dependency of the value function estimation with respect to the current policy. Evidently, the advantage of this approach is that the complexity of the learning problem is distributed and handled internally by two separate components, i.e. the controlling and learning component. Based on this set-up ADP methods are also referred to as Adaptive Critic Designs (ACD).[8]

A. ACD types

The first ACD design family was presented by P. Werbos in the early 1990s[17–19] and has been widely employed to different applications ever since. His work includes the basic forms: HDP and DHP as well as extensions such as Global Dual Heuristic Programming (GDHP) and their action dependent versions (AD-). Prokhorov et. al describes those ACD formulations in detail in his paper[10] and suggests two new modifications to the original GDHP design. All ACD forms have a similar structure but they can be distinguished by the input-output characteristics of the critic and the requirements of plant model derivatives:[20]

Critic Input: The critic receives a selected set of state information by the plant or plant-model. In action-dependent cases, such as ADHDP or ADDHP, the actor-output (i.e. the control signal) is also linked to the critic and serves as an additional input to the function approximator next to the state.

Critic Outputs: For HDP the critic approximates the value function which is usually denoted as J, while for DHP the critic-outputs are the derivatives of the value function with respect to each states $\mathbf{x}$, i.e. $\frac{\partial J}{\partial \mathbf{x}}$, also denoted as λ_x. In the GDHP case the output contains both, J and λ_x.

Plant Model Derivatives: The backpropagation pathway for updating the function parameters, differs with each ACD. Most designs require certain derivatives, such as $\frac{\partial x_{t+1}}{\partial x_t}$ or $\frac{\partial x_{t+1}}{\partial u_t}$ that can only be obtained via an on-board model of the plant. Here, x_t and x_{t+1} are the states in the current and subsequent time step respectively and u_t is the current control signal. The plant-model itself can be trained simultaneously with the actor and critic or be established prior the training

All forms can be considered as a class of non-linear control methods. This means that the plant that needs to be controlled does not need to be linearized around a certain operating point. Making use of non-linear function approximators enables also to control systems in continuous state-action space and therefore makes ACD the preferred reinforcement learning approach to solve real-life control tasks. Other benefits are 1) the algorithms do not require a very complex plant model a priori, 2) it is an intelligent type of controller that is adaptive to changing operating conditions and 3) it is robust in a noisy environment. The function approximators in those algorithms can be any parametric function, but the most common method is to use artificial neural networks (ANN). Their characteristics of universal function approximation and non-linear mapping have made them a popular tool in the field of reinforcement learning and artificial intelligence. They contain a large number of free parameters that can describe very complex systems and high state-space dimension.[21]

B. Heuristic Dynamic Programming

In HDP there are four entities that interact with one another: the critic, the plant, the plant model and the actor (see Figure 1). While the critic maps the current states to the associated value function and evaluates the environment, there must be some kind of exchange of information that delivers the knowledge of the critic to the actor. There is no direct link between the two entities and the information flow is provided only indirectly during the training process. The actor maps the current state to a control signal which changes the state of the plant that is being controlled. As the new state information is also provided to the critic

network, the actor influences indirectly the current estimation of the value function. On the other hand, the actor needs to approximate a control law that maximizes the value function. Starting in state $x(t)$, the critic estimates the associated value and the actor determines a control action $u(t)$ that is applied to the plant and the plant model. As a result of the transition to the new state $x(t + \Delta t)$ the agent receives an immediate reward r, which is determined by the reward function $r = \rho(x(t + \Delta t), u(t))$. Here $t + \Delta t$ indicates the next time-step. It is the only source of information for the error measure of the critic. In other words, the critic maps the new state to the value function and the approximation error is determined by the temporal difference error, which is the difference of the estimated values at two consecutive time steps and the immediate reward:

$$e_c(t) = J(x(t)) - (\rho(x(t + \Delta t), u(t)) + \gamma J(x(t + \Delta t)) \tag{4}$$

The critic is therefore trained backwards in time as a future estimation affects the current error and the task of the critic network is to minimize its individual cost function E_c by reducing e_c to zero, where E_c is defined as:

$$E_c = \sum_t \frac{1}{2} {e_c}^2(t) \tag{5}$$

The actor network on the other hand, needs to approximate a policy that maps the states to the optimal actions, thus controlling the plant to the regions of maximum value. The difference between the predefined optimal value J^* and the current value is defining the actor approximation error e_a, which in turn defines the cost function E_a that needs to be minimized:

$$e_a(t) = J(x(t)) - J^* \tag{6}$$

$$E_a = \sum_t \frac{1}{2} {e_a}^2(t) \tag{7}$$

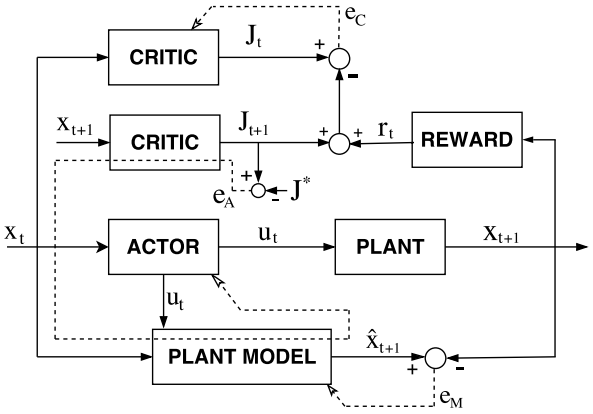


Figure 1. HDP algorithm at two consecutive moments in time. The backpropagation paths are indicated by the dotted line. e_a, e_c and e_m are the relative training errors for the actor, critic and model functions. r_t and $\hat{x}_{t+1}$ are the reward and predicted one-step ahead state respectively.

J^* is dependent on the chosen reward function $\rho(x, u)$ and needs to be set by the control engineer. One approach is to define the reward function in such a way that it is negative definite, i.e. $\forall x, u : \rho(x, u) \leq 0$. This leads to a maximum sum of rewards of zero and therefore $J^* = 0$. Backpropagating the current value function through the critic and plant model network to the actor results in the required information exchange from the critic to the actual controller.

As HDP is a model-based RL method, a plant model needs to be trained as well. This is done by finding the difference between the approximation and true states (e_m) and the cost E_m that needs to be minimized:

$$e_m(t) = \hat{x}(t) - x(t) \tag{8}$$

$$E_m = \sum_t \frac{1}{2} {e_m}^2(t) \tag{9}$$

Where $\hat{x}(t)$ is the plant model output and $x(t)$ is the true state vector. Knowing the associated errors and costs E_c, E_a and E_m, the effect of the network parameters to the related cost can be determined by using backpropagation:

$$\frac{\partial E_c(t)}{\partial W_c(t)} = \frac{\partial E_c(t)}{\partial e_c(t)} \frac{\partial e_c(t)}{\partial J(x(t))} \frac{\partial J(x(t))}{\partial W_c(t)} \tag{10}$$

$$\frac{\partial E_a(t + \Delta t)}{\partial W_a(t)} = \frac{\partial E_a(t + \Delta t)}{\partial e_a(t + \Delta t)} \frac{\partial e_a(t + \Delta t)}{\partial J(t + \Delta t)} \frac{\partial J(t + \Delta t)}{\partial x(t + \Delta t)} \frac{\partial x(t + \Delta t)}{\partial u(t)} \frac{\partial u(t)}{\partial W_a(t)} \tag{11}$$

$$\frac{\partial E_m(t)}{\partial W_m(t)} = \frac{\partial E_m(t)}{\partial e_m(t)} \frac{\partial e_m(t)}{\partial \hat{x}(t)} \frac{\partial \hat{x}(t)}{\partial W_m(t)} \tag{12}$$

Here, the network parameters are denoted by W. Their update is then done by gradient descent:

$$W(t+\Delta t) \leftarrow W(t) - \eta \frac{\partial E(t)}{\partial W(t)} \tag{13}$$

Where η is the learning rate. A graphical illustration of the backpropagation paths are indicated as dotted lines in Figure 1.

C. Dual Heuristic Programming

The major difference between HDP and DHP is that the critic network in Figure 2 approximates the derivative of the value function $J(x(t))$ with respect to the states $x(t)$, which are usually denoted as $\lambda_x(t)$. This means that having a state vector with n variables as input to the critic, the critic outputs contains n elements as well, describing $\frac{\partial J(x(t)}{\partial x_1(t)}, \frac{\partial J(x(t)}{\partial x_2(t)}$ ... $, \frac{\partial J(x(t)}{\partial x_n(t)}$. The function approximator trains to minimize the following error measure over time:

$$E_c = \sum_t \frac{1}{2} {e_c}^T(t) e_c(t) \tag{14}$$

Where

$$e_c(t) = \frac{\partial J(x(t))}{\partial x(t)} - \gamma \frac{\partial J(x(t+\Delta t))}{\partial x(t)} - \frac{\partial \rho(t)}{\partial x(t)} \tag{15}$$

Figure 2 shows all relevant backpropagation elements by dotted lines. It is clear that the training process is more complicated than in HDP due to the long backpropagation pathways. Applying the chain rule for the derivatives, we get for state variable k:

$$\frac{\partial J(x(t+\Delta t))}{\partial x_k(t)} = \sum_{i=1}^{n} \lambda_i(t+\Delta t) \frac{\partial x_i(t+\Delta t)}{\partial x_k(t)} + \sum_{s=1}^{m} \sum_{i=1}^{n} \lambda_i(t+\Delta t) \frac{\partial x_i(t+1)}{\partial u_s(t)} \frac{\partial u_s(t)}{\partial x_k(t)} \tag{16}$$

with

$$\lambda(x_k(t)) = \frac{\partial J(x(t))}{\partial x_k(t)} \tag{17}$$

Where n is the total number of inputs to the critic and m the number of outputs of the actor network. With that, $\frac{\partial E_c}{\partial W_c}$ can be determined and the network weights are updated as in equation 13.

The actor function has the same objective as in HDP, namely mapping the states to optimal actions. Its parameters W_a are updated by backpropagating $\lambda(x(t+\Delta t))$ through the critic and model network. The error may be expressed as

$$e_a = \frac{\partial \rho(t)}{\partial u(t)} + \gamma \frac{\partial J(x(t+\Delta t))}{\partial u(t)} = ...$$

$$\frac{\partial \rho(t)}{\partial u(t)} + \gamma \lambda_x(t+\Delta t) \frac{\partial x(t+\Delta t)}{\partial u(t)} \tag{18}$$

Then the update of the actor parameters becomes

$$W_a(t+\Delta t) \leftarrow W_a(t) - \eta_a e_a(t) \frac{\partial u(t)}{\partial W_a(t)} \tag{19}$$

In a simple form of ACD, such as HDP, the actor receives the value function derivatives only indirectly via backpropagation of the value function $J(x)$ through the critic and model network. This may result in a too coarse actor behavior with a slight off-set to the desired point. One approach to improve the results is suggested by Borghese et al.,[22] where the actor explores additional trajectories around the nominal one. However, for mathematical optimization it is more valuable to obtain

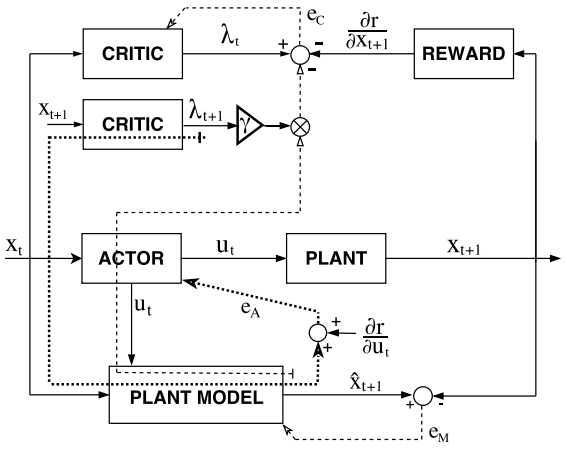


Figure 2. DHP algorithm at two consecutive moments in time. The backpropagation paths are indicated by the dotted line. e_a, e_c and e_m are the relative training errors for the actor, critic and model functions. The symbol $\otimes$ represents the factor within the critic estimation and the obtained derivatives from the actor and plant model. The plant model is the same as for HDP and trained the same way. λ_t relates to the partial derivative of the value function with respect to each state $\frac{\partial J(x_t)}{\partial x_t}$.

the derivatives of an optimization criterion than the optimization criterion itself. As part of advanced ACD's, the DHP method directly estimates the derivatives of J with respect to the states and is explicitly trained by the reward function derivatives $\frac{\partial \rho(x,u)}{\partial u(t)}$ and $\frac{\partial \rho(x,u)}{\partial x(t)}$. This has the advantage that the derivatives are directly provided to the actor rather than the value function itself. According to Prokhorov et. al,[10] the quality of such direct approximation is noted to be better than obtaining the indirect approximations. Furthermore, the model network is used for both, the critic and the actor, in the training process which leads to a better knowledge exchange between the different entities. The drawback is that the update process itself is more complex, involving more derivatives and update rules which makes it more tedious to set-up the algorithm for a given problem.

III. Implementation of HDP and DHP for pitch control of the F-16 Model

For this study, HDP and DHP are tested on a 2D pitch-reference tracking problem on a F-16 aircraft model and are compared in terms of performance. This section describes the implementation of both algorithms.

A. F-16 Model

The F-16 aircraft model is provided by Russell et al.[23] Although the choice of the aircraft type is rather arbitrary it gives a first indicator on how the two RL controllers perform for aircraft control. Before the experiment, the aircraft is trimmed to a steady-state flight with the settings as listed in Table 1 and then linearized around that equilibrium point. The aircraft-trim itself is found via a numerical procedure that finds the trim settings for which the state derivatives are near zero. Linearizing the model simplifies the controls and reduces the dimensionality of the state space. Furthermore, it enables to extract the pure longitudinal state-space model from the full-state, nonlinear plant. Consider the extracted system to be of the form:

Table 1. F-16 trimm settings

Variable	Symbol	Value	Units
Altitude	h	5000	ft
Airspeed	V_T	600	ft/s
Mach	M	0.547	-
AoA	α	0.0273	rad
pitch angle	θ	0.0273	rad
Elevator defl.	δ_{el}	-1.468	deg
Throttle	δ_T	0.2102	–

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} = f(\mathbf{x}, \mathbf{u}) \tag{20}$$

Where the state vector $\mathbf{x}$ consists of five state variables, i.e. altitude h, airspeed V_T, angle of attack α, pitch angle θ as well as pitch rate q. The control input is denoted by the control vector $\mathbf{u}$ and is limited to 2 signals, namely the elevator deflection δ_{el} and the throttle δ_T. It is assumed that the state information is available to the controller at all times (full state feedback), containing zero measurement noise. Furthermore, the controls are applied instantaneously with zero delay.

B. Approximated Plant Dynamics

Both, HDP and DHP are model-based RL algorithms, which means that they require an on-board model of the plant for their backpropagation pathways. The plant model can be trained online simultaneously with the actor and the critic, or it can be pre-trained before implementing it to the RL controller. Evidently, training the plant model a priori gives the benefit that the actor and critic networks are provided with correct derivatives in their updates from the start of their training loops. This reduces the complexity of the training cycle and convergence is improved. Therefore, in this study the plant is trained beforehand.

Figure 3 shows the chosen structure of the plant model. It is a feed forward neural network with 10 hidden layer neurons. The inputs are $\theta(t)$, $q(t)$ and the elevator deflection δ_{el}. Its outputs are the pitch angle and pitch rate at the subsequent time step Δt. Table 2 gives an overview of the used hyperparameters for the training procedure.

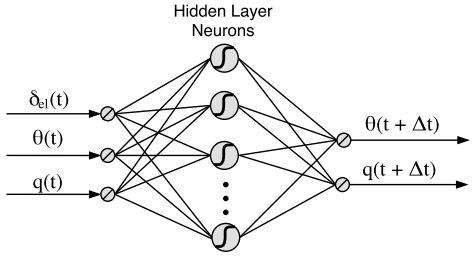


Figure 3. Model neural network with three inputs, 10 sigmoidal hidden layer neurons and two linear output neurons

Table 2. Hyperparameter settings for the model network

Variable	Symbol	value
Learning rate	η_m	0.4
Momentum factor	μ_m	0.6
Regularization	L	0.01
Max epochs	n_{max}	1000

The offline training is conducted with 120 seconds of simulation data which equals to 12000 data samples. 50% of the data-set is assigned for training and the remaining 50% for validation. The data is conducted around the same operating region as the controllers will operate and the input-output samples are normalized to an interval of -1 to 1 to enable faster learning. Figure 4 shows the results of the trained model with respect to a) training data and b) validation data. It can be concluded that the plant-model successfully captures the dynamics around the operating region with maximum difference of ± 0.2[deg]. Also the pitch rate is approximated with less than ± 0.5[deg/s] deviation.

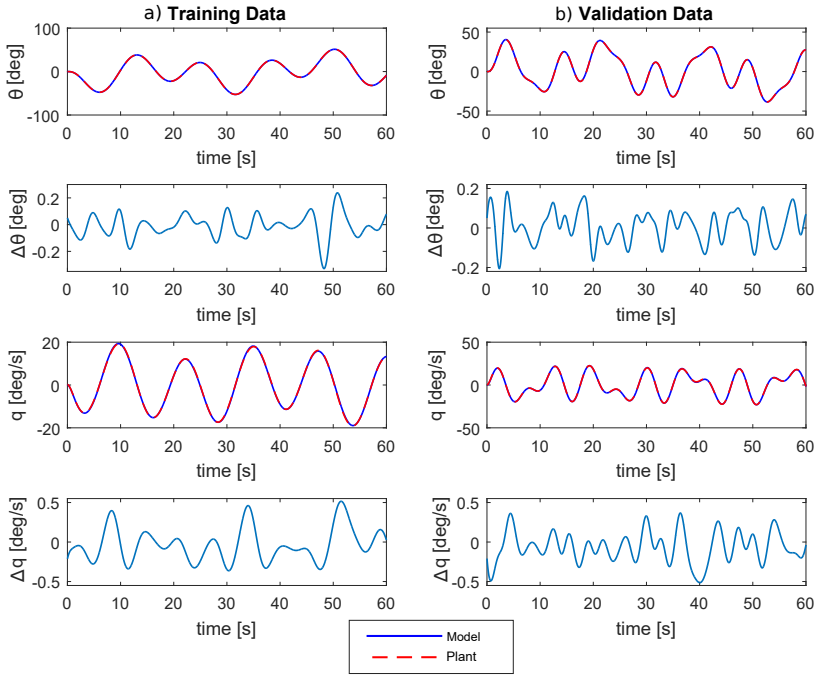


Figure 4. Approximated plant dynamics network behavior after training. Results are shown with respect to the a) training data and b) validation data.

C. Reward Function

A crucial design step is the definition of the reward signal as it has to define unambiguously the goal of the control task in a single function. In this case the goal for the controllers is to regulate the aircraft to follow a pre-defined pitch-reference signal θ_{ref}. Therefore, any deviations of the true pitch angle θ with respect to the reference is penalized. The instantaneous reward function is defined as negative quadratic error, normalized by the maximum angle pitch angle θ_{max}:

$$\rho(\theta_t) = -\left(\frac{\theta_t - \theta_{ref_t}}{\theta_{max}}\right)^2 \tag{21}$$

Where the maximum allowed pitch is in this case set to 60°, i.e. 1.0472 rad. Hence, the function itself in Equation 21 is negative definite, meaning that $\forall x : r(\mathbf{x}) \leq 0$. This implies that the optimal value that can be achieved is $J^* = 0$. Note that the controls are not penalized as with the current problem definition it is striven for the best tracking performance irrespectively of the control effort.

D. Actor-Critic Setup

The RL controller is integrated as shown in Figure 5. It is provided with the state and state-reference $\mathbf{x}$ & $\mathbf{x}_{ref}$ and maps those to an optimal elevator deflection δ_{el}. Before delivering the action to the plant it is added to the trim condition of the controls $\delta_{el_{trim}}$. The RL controller contains the actor-critic and plant model interaction scheme as shown in Figure 1 or 2 (depending on the algorithm). Both, the actor and the critic are feed forward neural networks with tangent hyperbolic activation functions in the hidden layer. A detailed overview of the network structures is given in the following paragraphs. Note that the throttle δ_T is not adjusted and kept at its trim at all times.

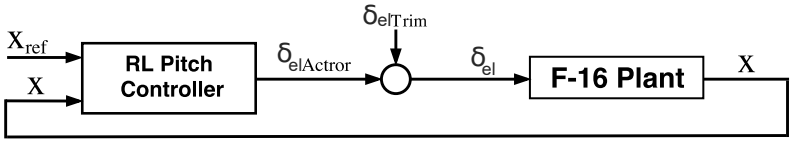


Figure 5. Reinforcement learning control setup.

1. *Actor Neural Network*

The actor neural network maps the current difference between the pitch angle and desired pitch angle $\Delta\theta(t)$, as well as the current body rate $q(t)$ to the elevator deflection δ_{el}. Its structure can be seen in Figure 6 and has the same layout for both RL algorithms. The hidden layer consists of 10 neurons with tangent hyperbolic activation functions. The same activation function is used in the output layer neuron. This enables to represent the saturation limits of the controls, which is in this case $\delta_{el_{max}} = \pm 25°$. A Tangent hyperbolic function in the output neuron has minimum and maximum values of [-1 1] which therefore represents the normalized values of the elevator deflection and need to be denormalized before its send to the plant.

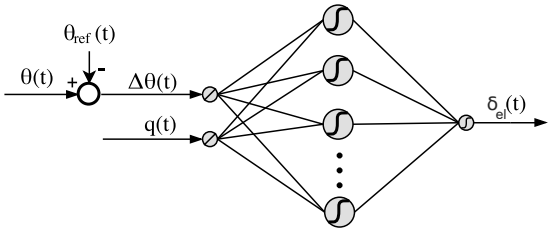
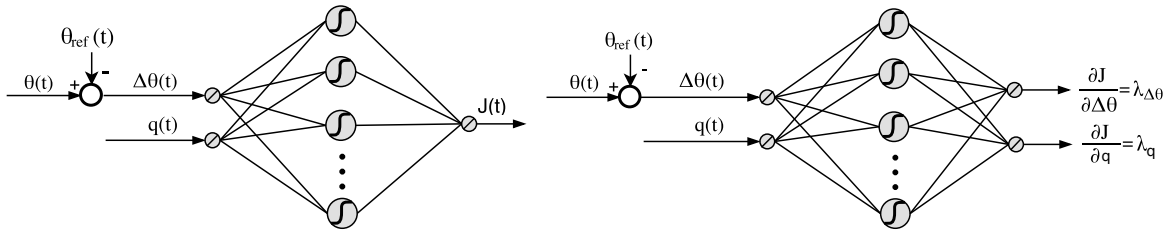


Figure 6. Actor ANN estimates the policy with $\Delta\theta$ and q as inputs and δ_{el} as output. It consists of 10 sigmoidal hidden layer neurons and 1 sigmoidal output neuron. The network structure is the same for HDP and DHP.

2. *Critic Neural Network*

Figures 7(a) and 7(b) show the critic structure for HDP and DHP respectively. Clearly, they differ in terms of number of outputs. In case of HDP, the critic network estimates the value function $J(t)$ with respects to the current $\Delta\theta(t)$ and $q(t)$ while in DHP the critic estimates the value function derivatives with respect to the inputs, i.e. $\frac{\partial J}{\partial \Delta\theta} = \lambda_{\Delta\theta}$ and $\frac{\partial J}{\partial g} = \lambda_g$. Both networks contain 8 hidden layer neurons with tangent-hyperbolic activation functions. The in- and output neurons are linear.



(a) Critic ANN of the HDP algorithm with 8 hidden layer neurons, estimating the value function $J(t)$.

(b) Critic ANN of the DHP algorithm with 8 hidden layer neurons, estimating the value function derivatives $\frac{\partial J}{\partial \Delta \theta} = \lambda_{\Delta\theta}$ and $\frac{\partial J}{\partial g} = \lambda_g$.

Figure 7. Critic network layout for the different algorithms with $\Delta\theta(t)$ and $q(t)$ as inputs.

E. Convergence improvements

Artificial neural networks are considered as a powerful tool for approximating non-linear functions. However, the problem of convergence and proper tuning of the hyperparameters still remains, especially when convergence of multiple networks depend on each other. To improve performance and convergence rates of the neural networks, the following improvement are integrated in the algorithms:

1. *Weight initalization:* Whenever the neuron receives high or low values, the output saturates to the interval bounds [-1 1], which means that the derivative of the sigmoid becomes nearly zero and learning stagnates. This effect is curbed by dividing all pseudo-random initial weight values by the square root of the number of input-nodes, while the weights itself are initialized with normalized Gaussians, i.e. $W \sim \frac{1}{\sqrt{n_{in}}} \cdot \mathcal{N}(0,\, \frac{3}{2})$.[24]

2. *Adaptive learning rate:* One way to improve weight-updates is to adapt the learning rate η during the learning process.[25,26] A high learning rate is desirable to make bigger steps down the slope of a long, continual decreasing cost-function and reducing the rate leads to a good asymptotic behavior and prevents oscillations at the later stage. For this purpose, the algorithms starts with a small initial η, where the learning rate increases when the gradient of successive trials points into the same direction, but is reduced as soon as the direction of the gradient fluctuates strongly.

3. *Momentum factor:* A momentum co-efficient $\mu \in [0\ 1)$ modifies the current gradient descent update by adding a scaled gradient term from the previous step:[24]

$$w \to w_{t+1} = \mu w_t - \eta \frac{\partial E}{\partial w_t} \tag{22}$$

$$W \to W_{t+1} = W_t + w_{t+1} \tag{23}$$

   Where the temporary term w is initialized to be zero. Momentum-based gradient descent algorithms have the advantage to include the knowledge of the past gradient and shall accelerate convergence. It is only applied to the network weights and not to the biases.

4. *L2 regularization:* Regularization, also known as weight decay, helps to prevent the network from overfitting by penalizing large weights.[24] It adds a regularization term to the error function E:

$$E = \frac{1}{2n} \sum_x \mathbf{e}(x)^T \mathbf{e}(x) + \frac{L}{2n} \sum_W W^2 \tag{24}$$

   where L is known as the *regularization parameter*, n is the total amount of samples in the data-set and W are the network weights excluding the biases. With this addend, the weight update gets:

$$W \to W_{t+1} = (1 - \frac{\eta L}{n}) W_t + w_{t+1} \tag{25}$$

   and the term $(1 - \frac{\eta L}{n})$ is considered to be the weight decay.

IV. Training Procedure

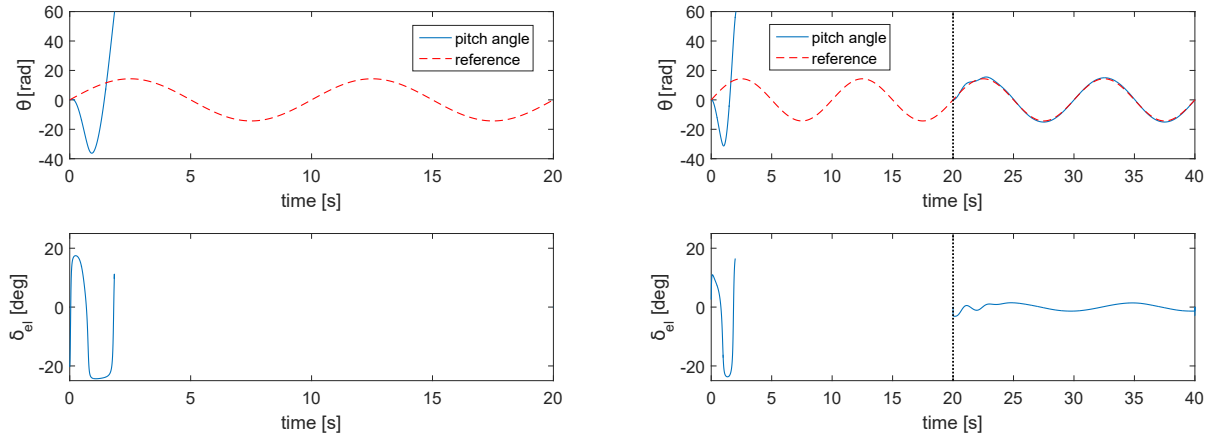
After the implementation into the F-16 aircraft model, the two controllers are trained in two separate training sessions, namely an offline and online learning phase. A description of the training process and the different update cycles is given in this section.

A. Offline Training

For complex systems, RL methods are in general trained offline as the learning process involves exploratory actions that often brings the system into undesired states. Figure 8 shows an example of trajectories where the controllers are directly learned online and offline. Clearly, in a pure online learning procedure, the controller soon starts to deviate strongly, unable to grasp any valuable information. In an offline learning set-up the system is reset after a certain period of time to the initial condition, independent of the previous control path which gives it multiple trials to learn. Furthermore, a guarantee of convergence for the critic and action neural networks during offline training has been given by Prokhorov et al.[27,28] That is why the two agents are first trained offline for learning to control the baseline F-16 model. The networks will be initialized with random weights at the start of each experimental run, meaning that the agents will have no prior knowledge in memory. One experimental run consists of 100 training episodes that last each a maximum of $T = 20$ seconds or are terminated once the aircraft exceeds a pitch angle limit of $\theta_{lim} = \pm 60°$. The aircraft is (re-)set to the trimmed condition (see Table 1) at every t_0, i.e. at the start of an episode. Then, the controllers must track a sinusoidal reference signal with a period of 10 seconds and an amplitude of 14.32°. The time-steps in the simulation are 0.01 seconds and during every step, the actor and critic network parameters are updated simultaneously, while the pre-trained plant model parameters stay fixed. The throttle setting is kept at trim throughout the whole simulation. An experimental run is considered a success once the actor error measure drops below a certain threshold. As the scale of the error measure is different for the two algorithms, the threshold is not the same. For HDP, the experiment is a success if $E_a \leq 1 \cdot 10^{-2}$ and for DHP if $E_a \leq 2 \cdot 10^{-5}$. The actor error is chosen as a success indicator as this entity is the actual controller of the system. Eventually the controllers are compared in terms tracking performance (off-set errors), convergence success rate, convergence time and number of trials that were necessary to drop below the error threshold. In order to speed up the learning process, all signals that are provided to the networks are normalized to [-1 1]. The pseudocode for the offline learning process can be viewed in the appendix of this paper.

Table 3. Simulation Settings

Variable	Symbol	Value
Number of trials	n_{trial}	100
Number of experiments	N	100
Max simulation time	T	20[s]
Time-step	Δt	0.01[s]
Pitch limit	θ	60[deg]



(a) Typical example of failed online training with no prior knowledge

(b) Successful offline training within 2 trials, by resetting the simulation every 20 seconds

Figure 8. Online vs offline training

1. hyperparameter selection

As the success of both training algorithms heavily depends on the set of chosen hyperparameters, it is important to select them carefully. In general, all hyperparameters are determined by brute force and extensive grid search. The search itself was the first phase of the offline training study. The search space contained in total almost 10000 different parameter constellations per algorithm and all have been evaluated by averaging the results of each setting over 5 experimental runs. Eventually, only the best 2 settings of each algorithm have then been chosen for the final offline simulation. Table 4 shows the two selected sets of hyperparameters.

Table 4. Hyperparameter settings

Variable	Symbol	**HDP**	**DHP**
Settings 1			
Learning rate critic	η_c	0.012	0.3
Learning rate actor	η_a	0.009	0.11
Momentum factor critic	μ_c	0.11	0.1
Momentum factor actor	μ_c	0.11	0.1
Settings 2			
Learning rate critic	η_c	0.05	0.2
Learning rate actor	η_a	0.01	0.08
Momentum factor critic	μ_c	0.1	0.11
Momentum factor actor	μ_c	0.11	0.1
Discount Factor	γ	0.97	

2. Exploration

Although exploration is not necessary in a policy gradient method for training the actor, the system may settle with a sub-optimal control law. This is due to the fact that the value function has not been able to explore other, more optimal states and exploits its current knowledge. In order to prevent this, the control actions are occasionally perturbed such that the system deviates from its intended states and ends up in 'new' states. The exploration rate ϵ determines the frequency of perturbation and adapts with the number of episodes. At the beginning of an experimental run the probability of taking an exploring action is 10% ($\epsilon_0 = 0.1$), meaning that almost every 10th step is perturbed. The probability soon decreases to near 0 after about 60 trails. An exploratory action is chosen from a Gaussian distribution with mean $\mu = 0$ and $\sigma = \frac{1}{3}$ such that 99% of the actions are within [-1 1] bounds. The exploration function is described as:

$$\epsilon(n_{trial}) = \epsilon_0 \cdot e^{(\xi \cdot n_{trial})} \tag{26}$$

Where ξ is the decay of the exploration and set to -0.1 and n_{trial} is the number of trials.

B. Online adaptation

After the agents have been trained successfully in the offline phase, they will be applied in an online learning setup. The goal is to determine 1) their robustness with respect to changes in the plant dynamics and 2) their ability to adapt online to those changes.

One of the most important parameters for the longitudinal stability of the F-16 aircraft is the moment coefficient C_m, which relates to the total moment about body axis:[29,30]

$$C_m = C_{m_0}(\alpha, \delta_{el}) + (\frac{q\bar{c}}{2V_T})C_{m_q}(\alpha) + (x_{c.g.ref} - x_{c.g.})C_Z \tag{27}$$

For the online simulation study, two different configurations are tested, namely a reduction of the pitch damping term C_{m_q} and a shift in center of gravity position $x_{c.g.}$. In the first experiment, the pitch damping is reduced by a factor of two. Then, in a separate experiment, the center of gravity is shifted such that the moment arm changes in sign:

$$\Delta x_{c.g.} = (x_{c.g.ref} - x_{c.g.new}) = 0.35\bar{c} - 0.40\bar{c} = -0.05\bar{c} < 0 \tag{28}$$

The individual changes will be made instantaneously at $t = 30s$ during a 60 seconds simulation run. Eventually the controllers are compared with respect to their control accuracy once with adaptation and once without. If adaptation is applied, all three entities, i.e. the actor, critic and plant-model will be updated simultaneously every time-step. Cleary, there are no exploratory actions in this case, i.e. $\epsilon = 0$. The measure of performance will be the root mean square error between the pitch angle vs. the reference as well as the time needed for adapting to the new plant dynamics.

V. Results

This section presents the results of the simulation studies. First the offline training performance is presented, followed by the online learning application.

A. Offline training

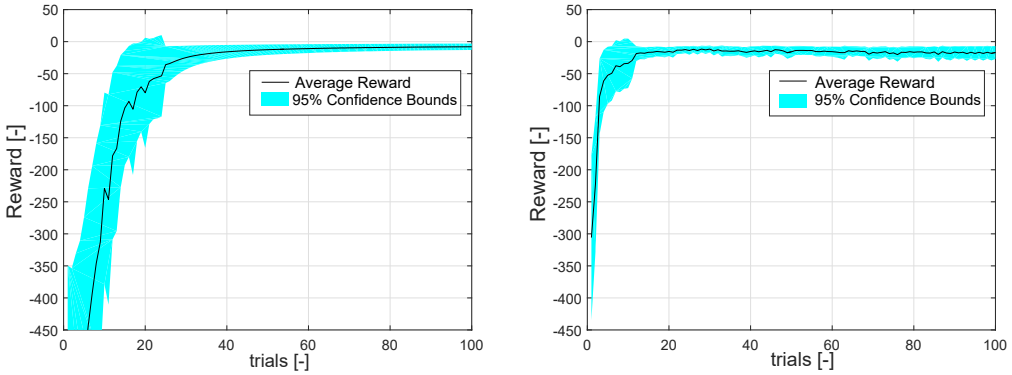
In order to make a fair comparison, the learning parameters need to be selected carefully. Extensive grid search has been applied together with personal experience to find 2 sets of hyperparameters that provided the best results in terms of control performance and learning time. However, despite the efforts, using brute force for infinite combinations of learning parameters is just not feasable and therefore, there is no guarantee that the selected parameters provide the best overall results. Nevertheless, they provide a tendency from which a reliable conclusion can be drawn from.

Four performance indicators will be looked at in this comparison. The success rate describes the percentage of experimental runs that successfully converged to an optimal control law. For all successful runs $\Delta\theta_{RMS}$ is computed, which is the root mean square error (RMS) between the pitch angle and the reference signal. Also the required trials and time needed to converge to that optimal state is listed.

Table 5 shows all the results averaged over 100 experimental runs for 2 selected hyperparameter settings. The chosen learning parameters can be viewed in Table 4. For both settings, it can be seen that DHP training results in a higher frequency of successful learning sessions with up to 63% success rate, while for HDP the best performed runs have given only 39%. Table 5 also shows clearly that using the same algorithm with just slightly different initial learning parameters results in a drastic change in performance. DHP, for example succeeds with the 2nd settings only 30 times out of 100 to converge to an optimal control law, similar to HDP. In both studies, DHP results on average in a lower RMS error, indicating a better overall control performance. Also the convergence time and number of trials required to converge to the optimal solution is almost 20-40% less than for HDP. Next to that, the typical learning behavior with the first hyperparameter set can be seen in Figure 9. It shows that DHP collects overall less penalty over the trials, while being also more stable in the process by having a more narrow confidence bound than HDP. Clearly it approaches a stable policy within the 13th trial on average. In the best case scenario it took even 1 trial to converge. HDP, on the other hand, reaches a stable policy between the 30 and 40 trials, or in the best case within 15 trials.

Table 5. Offline learning results with 2 different sets of hyperparameters

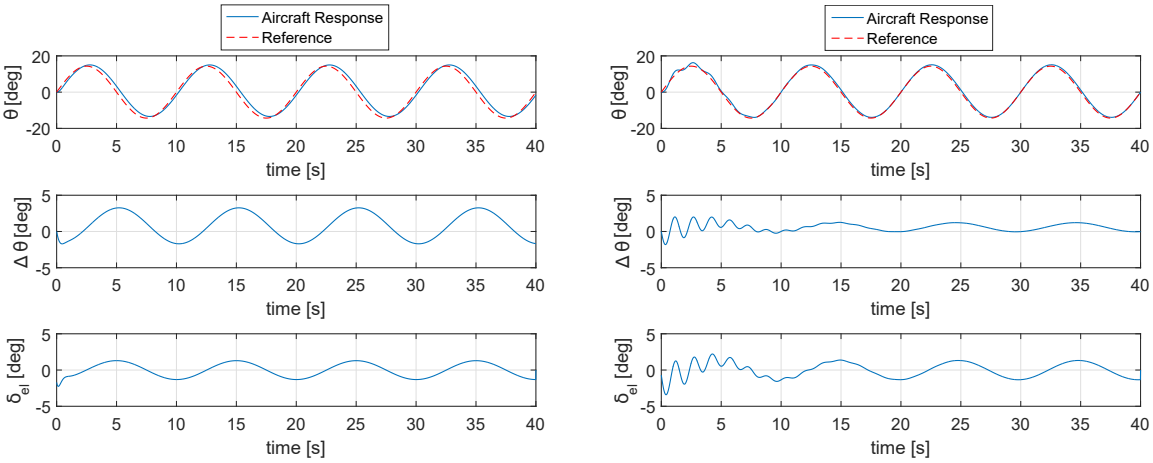
	HDP	**DHP**
Settings 1		
Success Rate	39%	63%
$\Delta\theta_{RMS}$	0.0389	0.0102
Required trials	22	13
Convergence time	55.5s	32.5s
Settings 2		
Success Rate	29%	30%
$\Delta\theta_{RMS}$	0.0335	0.0143
Required trials	25	21
Convergence time	62.5s	52.5s



(a) HDP (b) DHP

Figure 9. Collected Rewards over 100 experimental runs, each lasting 100 trials.

Looking at Figure 10 both algorithms are applied in a 40 second simulation after they have been trained. It shows as well a better tracking performance using the DHP algorithm with $\pm 1.3°$ off-set, seemingly representing a controller with higher gain. HDP always showed a slight bigger offset of about $\pm 4°$ with respect to the reference.



(a) Response of HDP controller after offline training.

(b) Response of DHP controller after offline training.

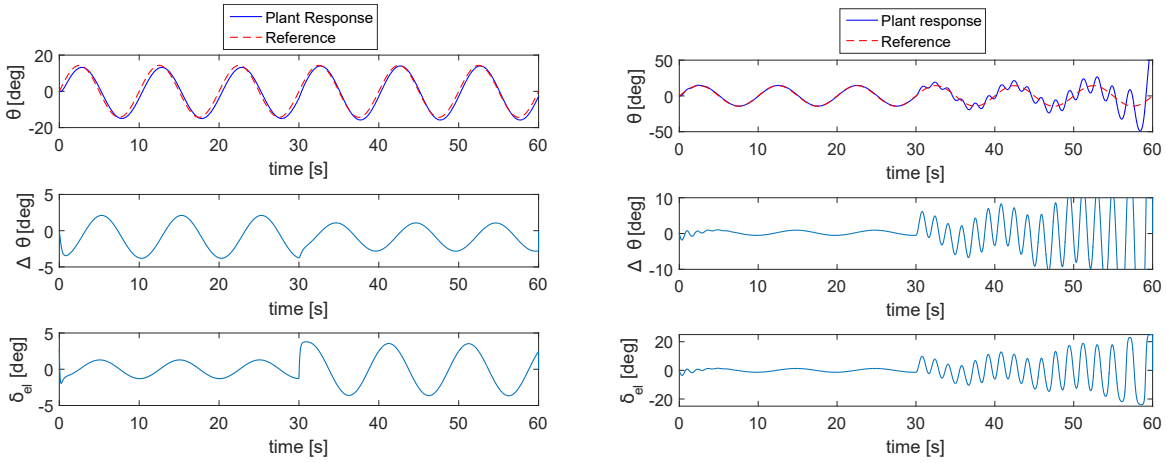
Figure 10. HDP vs. DHP control performance

B. Online adaptation

The best trained controllers from the previous section are tested in an online study where the plant dynamics change instantaneously. First, it is relevant to know if the controllers are robust enough to cope with the changes even without adapting the network parameters and then see as well how they perform when adaptation is enabled. The RMS error is compared as well as the convergence time for the simulations with adaptation. In this case, convergence time relates to interval between the start of the change in plant ($t = 30s$) up to the point when the weight update gradients are almost zero.

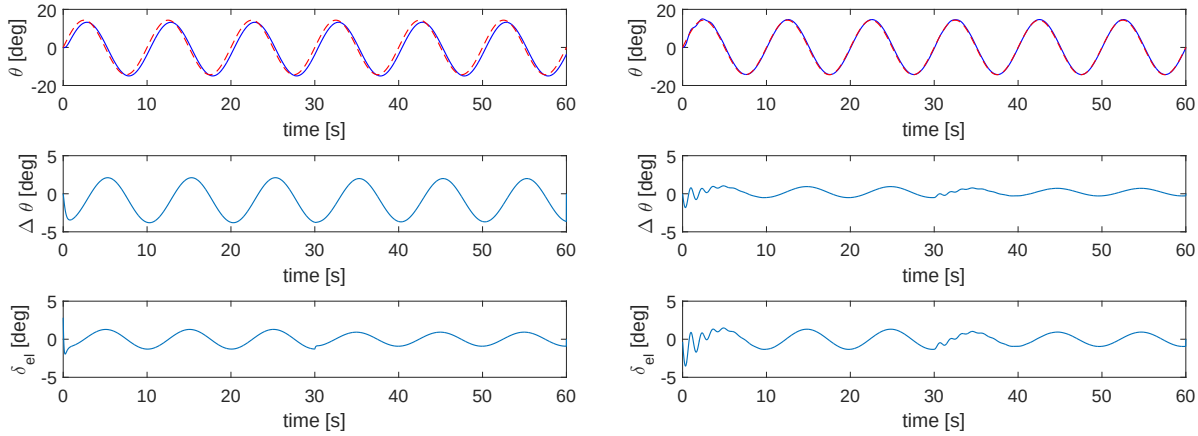
1. Response to changes in plant dynamics without adaptation

Figure 11(a) and 11(b) show the control performance for the first online simulation, where the center of gravity position is changed at $t = 30s$. HDP is able to adjust to the new plant dynamics quite well and tracks the reference with an off-set of less than 3.5 degrees. The RMS error even reduces from originally 0.0386 to 0.0288. Surprisingly, DHP is unable to cope with the new system dynamics and evolves into an unstable control law.
Similarly, Figure 12(a) to 12(b) shows the responses, where the moment coefficient C_m is reduced by a factor of two at $t = 30s$. This time, both controllers are able to cope with the changes in the plant dynamics and even improve in their performance. HDP and DHP reduce the RMS error from 0.0386 to 0.0382 and from 0.0104 to 0.0072 respectively which relates to a 1.2% and 31.% improvement.



(a) HDP (b) DHP

Figure 11. Online study without adaptation and changed center of gravity positiong ($\Delta x_{c.g.} = -0.05\bar{c}$). The plant dynamics are changed instantaneously at $t = 30s$.



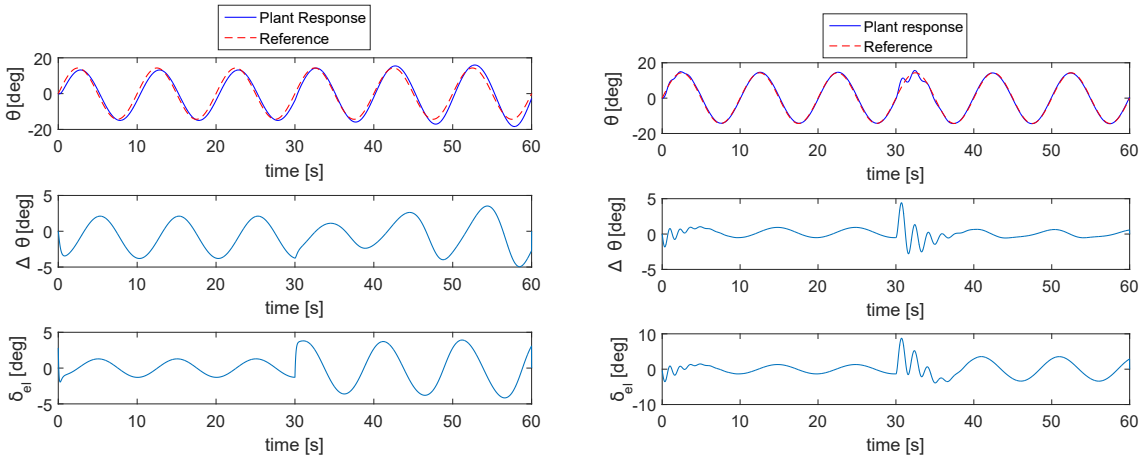
(a) HDP (b) DHP

Figure 12. Online study without adaptation and reduced pitch damping ($C_{m_q}/2$). The plant dynamics are changed instantaneously at $t = 30s$.

2. Response to changes in plant dynamics with adaptation

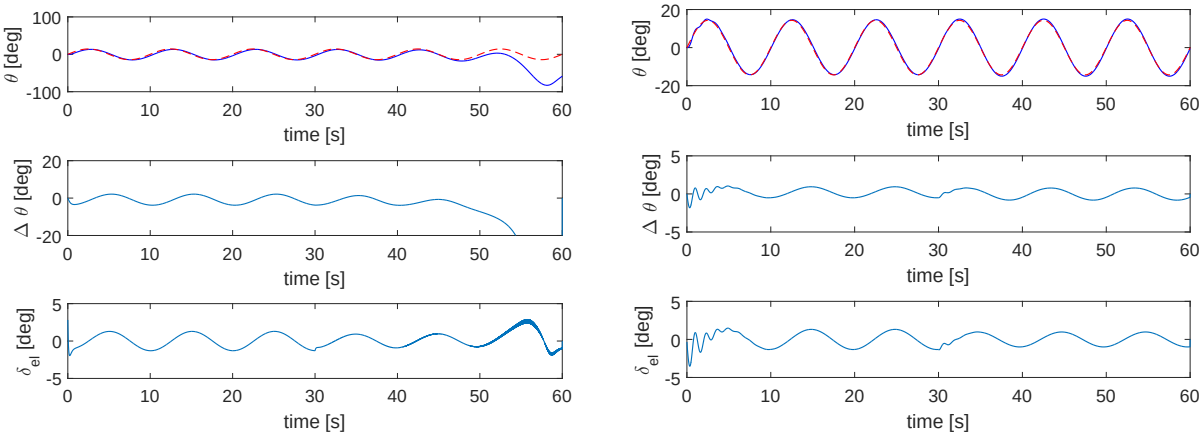
When enabling online learning, the controllers are able to update their network parameters during the simulation. In this case, the actor, critic and plant-model networks are updated with every time-step. Figure 13(a) and 13(b) show the results of the online study with the change in center of gravity. It can be seen that HDP adapts fast, but also seems to learn a wrong behavior in time as the pitch angle diverges more and more from the reference signal, seemingly getting more unstable. DHP, on the other hand, initially adapts very strongly to the changes in plant dynamics and overshoots slightly the reference by $\pm 5°$, but quickly converges to an optimal policy with a stable control and maximum error of $\pm 1.5°$ in less than 6 seconds.

The problem of stability for HDP becomes more apparent when updating the parameters of the neural networks during the second online experiment, where the pitch damping is reduced (see Figure 14(a)). HDP clearly gets unstable and is unable to adapt, although it was robust enough without adaption beforehand. DHP did not show any problems and converged to an optimum in less than 2 seconds with a final RMS error of 0.0099 (Figure 14(b)).



(a) HDP (b) DHP

Figure 13. Online study with adaptation and changed center of gravity positiong ($\Delta x_{c.g.} = -0.05\bar{c}$). The plant dynamics are changed instantaneously at $t = 30s$.



(a) HDP (b) DHP

Figure 14. Online study with adaptation and reduced pitch damping ($C_{m_q}/2$). The plant dynamics are changed instantaneously at $t = 30s$.

VI. Conclusions & Recommendations

In this paper, HDP and DHP are compared and evaluated on a 2D pitch-tracking task in a F-16 aircraft model. The experimental study involves an offline and online learning phase. The results show that during the initial offline phase, both controllers are able to learn the correct behavior and control the baseline model of the F-16 aircraft. DHP shows higher success ratios and converges to an optimal control-law in almost half the time as HDP. Also the tracking performance is more accurate, with an overall lower RMS error.
In the online learning phase, both controllers can handle most changes in the longitudinal aircraft dynamics even without adapting their parameters. This reveals their robustness as the baseline controller can cope with some changes in the dynamics, sometimes with better performance than without those changes. Especially HDP improves in the tracking performance for *all* the different modes. DHP seems less robust. It is able to follow the pitch reference signal very well for changes in the C_{m_q}, but is unable to handle the shift in the center of gravity position, resulting in an unstable behavior.
However, for simulations with parameter adaptation DHP shows a clear performance improvement, as it can adapt quickly to all the changes in plant dynamics, while HDP seemingly learns an unstable behavior.
It can be concluded that DHP is a better method for aircraft control than HDP. It showes higher success rates, faster learning, decent robust behavior as well as quick and stable adaptation to changes in the plant dynamics. The adaptability and self learning behavior makes it a promising solution for the VCCTEF design. However, a major drawback of these RL methods is that the learning process is not transparent and it is unknown what exact changes are made to the network parameters with every update cycle. This especially becomes a problem when the controller is learning a wrong policy, getting the system unstable as it is the case with HDP. Furthermore, the high sensitivity to the initial learning parameters makes it unclear whenever the best parameter combinations are chosen. Just a slight change in the settings leads to a drastic change in performance in this study. It is also noted that the RL methods are well able to control complex systems with two states and one control variable, but as soon as the amount of states and actions increases, the individual networks may have more trouble to converge. As overall convergence depends on the convergence of three separate entities (actor, plant-model and critic network) the probability of convergence of all entities to an optimum gets very low. Looking at the VCCTEF and its complexity with more than 48 control surfaces that can be actuated, a direct control of the effector using RL methods may not be feasible at this time.
For future research it is therefore recommended to apply HDP and DHP for aircraft control in a bigger state-action space. This shall reveal how the algorithms can perform with an increased complexity of the tasks. Furthermore, brute force methods and grid search for defining the hyperparameters of the neural networks is a tedious task and does not guarantee the best settings. A way to improve this (and also to make the comparison between the algorithms more fair) is to use machine learning methods such as Bayesian optimization to search in the hyperparameter space and solve for the best settings. Another recommendation is to check the sensitivity and performance of the controllers with respect to sensor noise and delays. Another idea would be to test other candidates of the ACD design family like GDHP, which supposedly combines the benefits of HDP and DHP and may be even more beneficial for aircraft control.

References

[1]Nguyen, N. T., "Elastically Shaped Future Air Vehicle Concept," *NASA Innovation Fund 2010 Project*, 2010.

[2]Lebofsky, S., Technologies, S. G., and Field, M., "Multidisciplinary Drag Optimization of Reduced Stiffness Flexible Wing Aircraft With Variable Camber Continuous Trailing Edge Flap," 2015.

[3]Nguyen, N. and Urnes Sr., J., "Aeroelastic modeling of elastically shaped aircraft concept via wing shaping control for drag reduction," *AIAA Atmospheric Flight Mechanics Conference*, , No. 650, 2012.

[4]Nguyen, N. T. and Tal, E. A., "A Multi-Objective Flight Control Approach for Performance Adaptive Aeroelastic Wing," *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, , No. 650, 2015.

[5]Chowdhary, V. G., Frazzoli, E., How, P. J., and Liu, H., *Nonlinear Flight Control Techniques for Unmanned Aerial Vehicles*, Springer, 2014, pp. 577–612.

[6]Sutton, R. S. and Barto, A. G., *Reinforcement learning : an introduction*, MIT, 1998.

[7]Barto, A., Sutton, R. S., and Anderson, C. W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *Transactions on Systems, Man, and Cybernetics*, No. 5, Bäuerle, 1983, pp. 834–846.

[8]Konda, V. R. and Tsitsiklis, J. N., "On Actor-Critic Algorithms," *Control Optim*, Vol. 42, No. 4, 2003, pp. 1143–1166.

[9]Witten, I. H., "An Adaptive Optimal Controller for Discrete-Time Markov Environments," *Information and Control 34*, 1977, pp. 286–295.

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

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

[12]Liu, D., Javaherian, H., Kovalenko, O., and Huang, T., "Adaptive critic learning techniques for engine torque and air-fuel ratio control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 38, No. 4, 2008, pp. 988–993.

[13]Venayagamoorthy, G. K., Harley, R. G., and Wunsch, D. C., "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, pp. 764–773.

[14]Bellman, R. E., "The Theory of Dynamic Programming," Princeton University Press, 1954.

[15]Bellman, R. E., *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

[16]Bertsekas, D. P. and Tsitsiklis, J. N., "Neuro-Dynamic Programming: An Overview," *Proceedings of the 34th Conference on Decision & Control*, , No. December, 1996, pp. 1–34.

[17]Werbos, P. J., "Approximate Dynamic Programming for Real-Time Control And Neural Modeling," *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, edited by D. A. White and D. A. Sofge, Van Nostrand, New York, 1992, p. ch 13.

[18]Werbos, P., "Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence," *General Systems*, , No. XXII, 1977, pp. 25–38.

[19]Werbos, P., Sutton, R. S., and Miller, W. T., "A menu of designs for reinforcement learning over time," *Neural Networks for Control*, Vol. MIT Press, 1990, pp. 67–95.

[20]Lendaris, G. G., "A retrospective on Adaptive Dynamic Programming for control," *Proceedings of the International Joint Conference on Neural Networks*, 2009, pp. 1750–1757.

[21]Barron, A. R., "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inf. Theory*, Vol. 39, No. 3, 1993, pp. 930–945.

[22]Borghese, N. and Arbib, M., "Generation of temporal sequences using local dynamic programming," *Generation of temporal sequences using local dynamic programming*, Vol. 1, 1995, pp. 39–54.

[23]Russell, R. S., "Non-Linear F-16 Aircraft Model," *University of Minnesota*, 2003.

[24]Nielsen, M. A., *Neural Networks and Deep Learning*, Determination Press, 2015.

[25]Schiffmann, W., Joost, M., and Werner, R., "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons," *Physics*, , No. September, 1994, pp. 1–36.

[26]Darken, C. and Moody, J., "Note on Learning Rate Schedules for Stochastic Optimization," *Neural Information Processing Systems*, 1991.

[27]Prokhorov, D. V. and Wunsch, D. C., "Convergence of Critic-based training," *Proc. IEEE int. Conf Syst. Man. Cybern*, 1997.

[28]Prokhorov, D. V. and Feldkamp, L. a., "Analyzing for Lyapunov Stability with Adaptive Critics," *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, , No. 2, 1998, pp. 1658–1661.

[29]Saeed, N., "Gain-Scheduled Lateral-Directional Control of F-16 Aircraft," Tech. rep., Delft University of Technology, Faculty of Aerospace Engineering, 2005.

[30]Kampen, E., Chu, Q. P., and Mulder, J. A., "Continuous Adaptive Critic Flight Control aided with Approximated Plant Dynamics Reinforcement learning," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2006.

Appendix: Pseudo code

Algorithm 1 Offline learning algorithm for HDP and DHP

Get the pre-trained model network
Initialize the parameters of the actor and critic function approximators, W_a, W_c
Definition: $\mathbf{x} = [\Delta\theta,\, q]^T$ and $u = \delta_{el}$
for n_{trials} **do**
    Initialize $t = 0$ and $x(t_0)$
    Set the exploration factor $\epsilon(n_{trial})$
    while $t < T$ and $\theta < \theta_{limit}$ **do**
        Compute the output of the critic at time t,
        $J(t)$ or $\lambda_x(t) \leftarrow f_c(\mathbf{x}(t); W_c(t))$
        Make an exploratory action or compute the actor output at time t,
        if $\epsilon > \mathcal{N}(0,\, 1)$ **then**
            $u(t) \sim \mathcal{N}(0,\, \frac{1}{3})$
        else
            $u(t) \leftarrow \pi(\mathbf{x}(t); W_a(t))$
        Estimate the one-step ahead state via the plant model,
        $\hat{\mathbf{x}}(t + \Delta t) \leftarrow f_m(\mathbf{x}(t), u(t); W_m(t))$
        Compute expected reward | reward derivatives,
        $r \leftarrow \rho(\mathbf{x}, u)$ or $\frac{\partial \rho}{\partial \mathbf{x}}$ and $\frac{\partial \rho}{\partial u}$
        Compute the output of the critic at $t + \Delta t$,
        $J(t + \Delta t)$ or $\lambda(t + \Delta t) \leftarrow f_c(\mathbf{x}(t + \Delta t); W_c(t))$
        Compute the critic error e_c
        Backpropagate $J(t + \Delta t)$ or $\lambda(t + \Delta t)$ and update the actor parameters W_a
        Update the critic parameters W_c
        Plant transitions to the new state,
        $\mathbf{x}(t + \Delta t) \leftarrow f(\mathbf{x}(t), u(t))$
        $t \leftarrow t + \Delta t$

II

Literature Review and Preliminary Analysis

2

Variable Camber Continuous Trailing Edge Flap

This chapter gives an overview of the past and most recent developments of the VCCTEF design. 2.1 introduces the background and motivation of such system, while the technical details are described in section 2.2 Finally, 2.3 gives an overview of the most recent multi-objective control design structure.

2.1. Introduction

Commercial aircraft are usually designed for a single operating point, compromising between cruise speed and landing requirements (Taylor [62]). Especially low performance at low speeds requires flaps and slats. Airplanes with a higher aspect ratio are considered to be more aerodynamically efficient, but the large wing span relative to the short root chord length may introduce stiffness problems. The use of lightweight materials amplifies this effect by causing the wing to be even more flexible. Advanced lightweight materials have been applied on airframe structures to a larger extent. One example is the Boeing 787 aircraft, which makes great use of advanced composite materials for a big part of its airframe structure. The Dreamliner was released in 2011 and contains almost 50% of mainly Carbon Fiber Reinforcement Plastic (CFRP). It is claimed that the weight savings add up to 20 percent on average when using CFRP compared to conventional aluminum designs (Hale [20]). In 2015, Airbus followed with the release of the A350XWB, an aircraft which contains nearly 53% of advanced composite materials and promises 25% reduced emissions as stated by Airbus [1].

Indeed, the weight savings are significant. CFRP has a high structural-strength to mass ratio, but at the same time may cause reduction in structural rigidity. Looking at modern commercial aircraft wings with wide wingspan and short root chord length, the emerged flexibility can result in large elastic deflections when exposed to aerodynamic forces (Nguyen et al. [47]). Especially at off design conditions, the elastic deflections of the wing may be undesired, causing suboptimal lift distribution and/or increased drag counts. In other words, the promised fuel savings may be counteracted by the decreased aerodynamic performance.

In recent years, there has been a study to actively adapt the shape of an aeroelastic wing to address those problems. The idea of the so called Performance Adaptive Aeroelastic Wing (PAAW) is to reshape the wing in any flight condition such that that optimal aerodynamic performance can be achieved and simultaneously control effectiveness can be ensured. Initiated as a NASA Innovation Fund project, a PAAW control design was proposed by Nguyen et al. [47] and his team in 2010. The Variable Camber Continuous Trailing Edge Flap (VCCTEF) concept was born. Today, the project is jointly being developed by NASA and Boeing Research & Technology under the NASA Advanced Air Transport Technology project. VCCTEF is a control surface that provides spanwise load tailoring with continuous trailing edge flaps (see figure 2.1). It is a control design that tackles multiple objectives, namely:

1. Neutralize negative aeroelastic effects

2. Improve handling qualities and

3. Improve aerodynamic performance

Up to this point, the state of research is on the level of concept evaluation, which is modeling the aerodynamics, optimizing the design and analyzing the performance. In simulation studies the VCCTEF concept already promises drag reduction for cruise flight conditions while improving stability margins and complying to operational constraints.

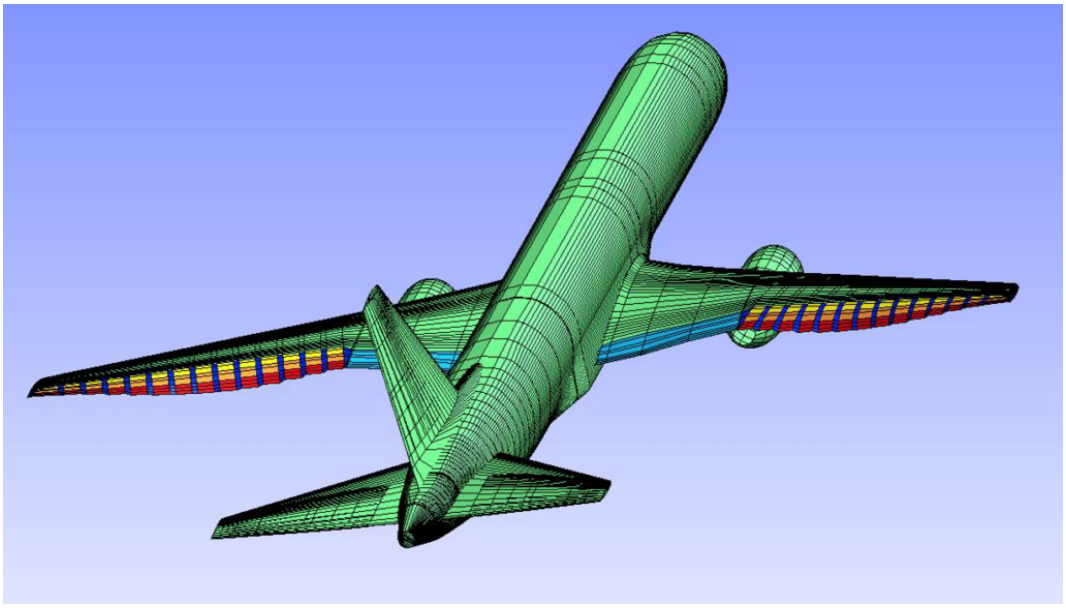


Figure 2.1: Variable Camber Continuous Trailing Edge Flap (VCCTEF) concept on a generic aircraft model (Nguyen and Tal [46])

2.2. The VCCTEF System

VCCTEF is a control surface that provides spanwise load tailoring with continuous trailing edge flaps, whereby adjacent flap sections are joined together by an elastomer material. Starting with the project kick-off in 2010, several studies such as Nguyen and Hornby [44] and [42] focused on improving the design with respect to optimal camber shape, number of flap sections and flap section widths. The most recent VCCTEF design layout can be seen in figure 2.1 and 2.2.
There are in total 16 flap sections spread along the complete wing trailing edge, with 15 sections being attached to the outer wing and 1 section being attached to the inner wing. Each flap consists of three chordwise segments that can actively shape the camber. A detailed view of a generic cross-section is given in figure 2.3.
The outspread, spanwise control surface enables to control wing-tip bending and the wing-twist shape along the whole wingspan. This, in turn, gives the benefit to settle for the best lift-to-drag ratio in any flight mode and at any aircraft grossweight. get an optimal, elliptical lift distribution (Nguyen and Tal [46]). The elastomer material between each flap prevents structural discontinuities, resulting in a more laminar airflow over the wing and thus reducing the viscous drag counts (Boeing [8]). However, due to this flexible link the individual flap segments are not entirely independent from each other. Even though the material is currently developed by Boeing Research & Technology, the material properties are still classified and are unknown. It is therefore an additional variable in the design instead of a constant, but the imposed angle constraints between two neighboring flap sections with index i and j are assumed to be:

$$|\delta_{flap_i} - \delta_{flap_j}| \leq 2\,\text{deg} \tag{2.1}$$

The VCCTEF only replaces the conventional flaps and ailerons on an aircraft while the rudder and elevator are still present. The thrust and control deflection limits are listed in table 2.1.
The current actuator design consists of a combination of Shaped Memory Alloys rods (SMA) in combination with electric motor drives for each flap segment (Nguyen and Tal [45]). SMA's have good weight vs power level characteristics and are applicable for adaptive control, but the actuation frequency is rather slow. They drive the two inner cordwise segments to gradually change the camber of the VCCTEF for the purpose of drag minimization. The out-most segment is driven by the electro-mechanical motors which enable high frequency

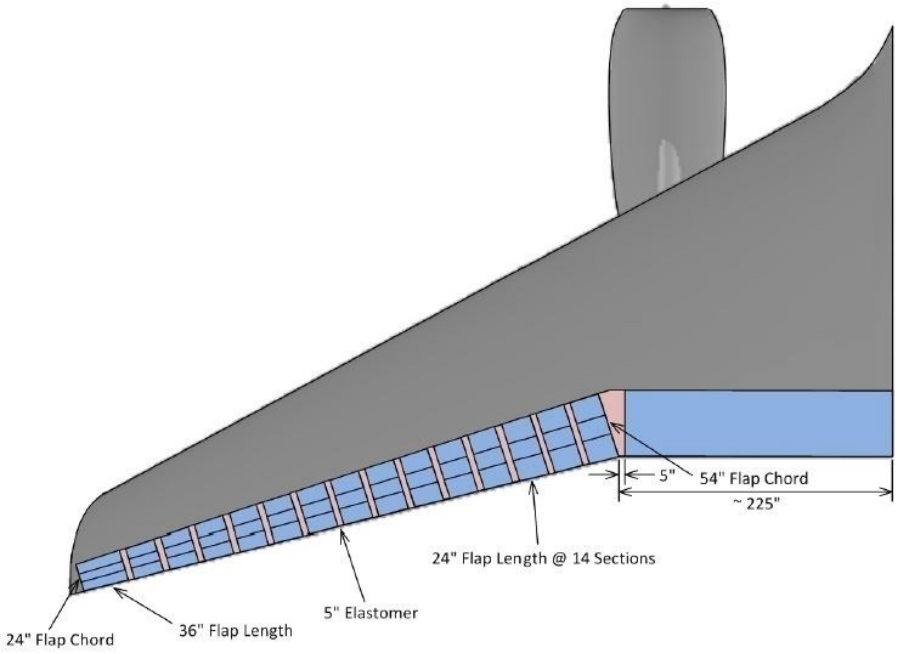


Figure 2.2: The Variable Camber Continuous Trailing Edge Flap consists of 15 flap sections on the outer wing and 1 flap section on the inner wing. All sections are connected via an elastomer material (Lebofsky et al. [29], Nguyen and Tal [46])

Table 2.1: Saturation limits of control surface deflections and thrust

Control	Minimum	Maximum
δ_{VCCTEF}	$-20\,\text{deg}$	$20\,\text{deg}$
δ_{el}	$-25\,\text{deg}$	$25\,\text{deg}$
δ_r	$-30\,\text{deg}$	$30\,\text{deg}$
T	$1000 lbs$	$40000 lbs$

actuation for fast flight maneuvers, aeroelastic mode suppression as well as gust load and maneuver load alleviation. A more detailed actuator design, however, still has to be developed in the near future.

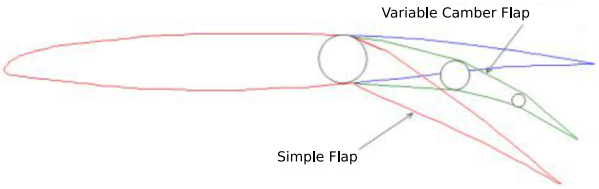


Figure 2.3: All individual segments contain 3 flap sections to actively change the camber shape (Lebofsky et al. [29], Nguyen and Tal [46])

2.3. Multi Objective Flight Control

There have been multiple approaches to develop a control system in combination with the VCCTEF. All of which have their main focus in drag reduction. It started with drag minimization for cruise flight at off-design conditions (Lebofsky et al. [29]) followed by subsonic cruise for rigid wing aircraft (Ippolito et al. [23]). S.Lebofksy, E.Ting and N. Nguyen continued with an approach of multidisciplinary drag optimization control for flexible wing aircraft in [30]. Additional studies such as Nguyen and Tal [45] and Lebofsky et al. [31] focused on flutter suppression and maneuver load alleviation.
As the VCCTEF is a multi-functional type of aerodynamic control surfaces, it provides multi-axes control options. In that perspective, single-axis control tasks such as a pitch command has to be re-examined. In order to ensure control effectiveness and optimal aerodynamic efficiency with the VCCTEF, a multidisciplinary con-

trol approach is necessary. That is, next to guaranteeing stability and controllability, the control system has to attain optimal aerodynamic performance. In 2015, Nguyen and Tal [46] proposed a multi-objective flight control approach using VCCTEF, where the flight controller addresses the following objectives:

1. Control actuation
2. Aeroelastic mode suppression
3. Gust and Maneuver Load alleviation
4. Aircraft Stability Augmentation
5. Drag Cognizant Control & Real-Time Drag Optimization

The multi-objective control architecture can be seen in figure 2.4. The design provides the ability to find the optimal flight control command as a compromise between the different, competing requirements. E.Tal and N.Nguyen focused in their study on aeroelastic mode suppression and drag minimization control. The simulations even included a gust-model to simulate noisy operating conditions.
Most recently, Ferrier et al. [18] published her first results for real-time drag minimization control using adaptive least squares and now continues with gust load and maneuver load alleviation control within the multi-objective flight control architecture. Y.Ferrier as well as E.Tal apply in their papers the concept of virtual control to account for imposed structural constraints which is explained in subsection 2.3.2.
For simulations and testing, the latest research makes use of a coupled linear symmetric aeroservoelastic model of the NASA Generic Transport Model. A small description of the simulation environment and its development is given in the next subsection.

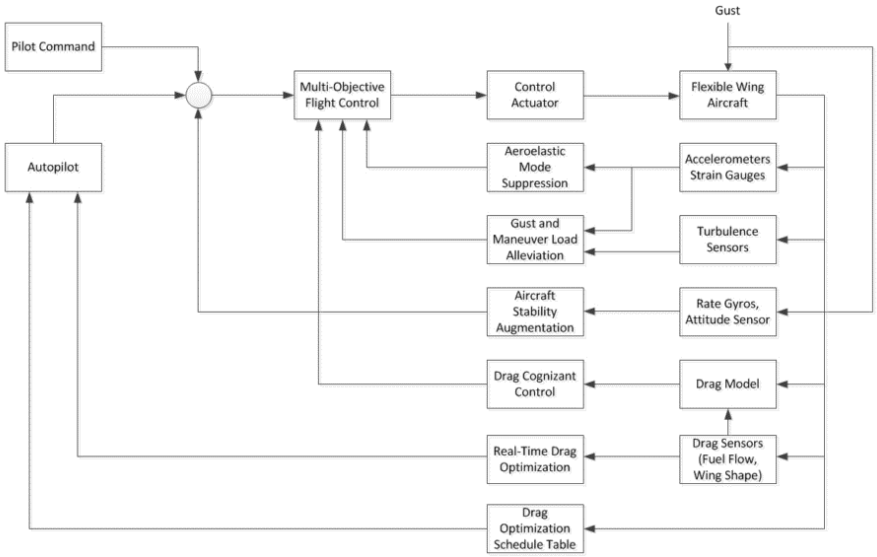


Figure 2.4: The multi-objective flight control unit and its related control loops Nguyen and Tal [46]

2.3.1. Simulation Environment

In 2004, NASA Langley developed a testbed for flight research experiments (Jordan et al. [24]). The so called Generic Transport Model (GTM) is a unique simulation environment that provides a platform for research in modeling dynamics and control systems. The GTM comes along with a dynamically scaled and turbine powered transport aircraft that is based on a Boeing 757, for in-flight validation of high risk flight tests. Nguyen and Urnes Sr. [39] extended the rigid GTM with an Elastically Shaped Aircraft Concept (ESAC). It not only includes light weight and highly flexible wing structures, but also the VCCTEF control effector. The baseline wing stiffness is arbitrarily reduced by 50%. Fuselage and tail surface deflections are considered to be negligible and for symmetric flight, the motion of the right wing is a mirror image of that of the left wing. In order to solve coupled wing-bending and torsion aeroelastic equations Galerkins method has been used (Nguyen et al. [41]). The conceptual design and preliminary modeling concluded phase I of the VCCTEF research and

was followed by phase II, which involved aeroelastic analysis and flutter suppression control. It also included 2 wind tunnel tests, one for cruise configuration in 2013 (Urnes and Nguyen [65]) and one for high lift configuration in 2014 (Boeing [9], Precup et al. [50]). The results formed a good database and helped to improve the simulation model. 2D and 3D CFD simulations gave a first indication about the performance of the VCCTEF (Kaul and Nguyen [26]). The circular-arc-camber and parabolic-arc-camber configuration achieved a higher lift-to-drag ratio compared to the clean wing setting. The current aeroelastic model is developed using NASA's vortex-latex code in combination with a Finite Element Model (FEM) (Ting et al. [64]) and a 2D transonic small disturbance code (Stahara [59]). In order to partially compensate for viscous drag, skin friction drag is included into the model as well (Nguyen et al. [40]).

2.3.2. Simplifications and Virtual Control

As the control effector contains 16 sections and each section, again, comprises of 3 flap segments, the total number of control surfaces adds up to $16x3 = 48$ per wing. However, any control algorithm is more feasible and more computationally reliable when fewer variables are used (Lebofsky et al. [29]). Simplifications can be made by approximating the shape of the wing and flap segments. First, it is assumed that the chordwise flap segments form a circular arc-camber shape, meaning that the deflection angles of the first two segments can be described in mathematical relation to the third and last flap deflection angle (δ_3) as follows

$$\delta_2 = \frac{2}{3}\delta_3 \tag{2.2}$$

$$\delta_1 = \frac{1}{3}\delta_3 \tag{2.3}$$

Equation 2.2 and 2.3 describe the coupling of the segments and figure 2.5 shows the definition of the deflection angles along the chordwise segments. For the development of the control design in Nguyen and Tal [46] and Ferrier et al. [18], only the third and last deflection angle δ_3 is considered, leaving out δ_1 and δ_2. It was found in a previous study that the circular-arc camber shape provides the best aerodynamic performance upon different camber shapes (Kaul and Nguyen [27]) and solely using δ_3 for controls reduces the size of the control vector by a factor of 3.

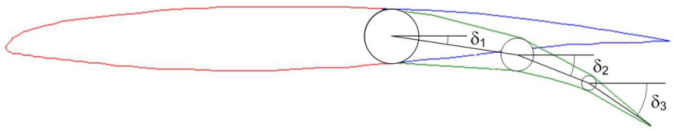


Figure 2.5: The three segments of the Variable Camber Continuous Trailing Edge Flap (Lebofsky et al. [29])

Furthermore, the control surface deflections can be defined by a shape function which is referred to as virtual control and was introduced by Nguyen and Urnes in 2012. The shape function can be of any shape with smooth and moderate slope (Tal [61]). That could be Fourier sine series or nth order Chebyshev polynomial (cheby). For example, using 5th order Chebyshev polynomials gives:

$$\partial_{i,3} = \sum_{n=0}^{N} a_n T_n(x) \tag{2.4}$$

Where $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2$, $T_3(x) = 4x^3 - 3x$, $T_4(x) = 8x^4 - 8x^2$, $T_5(x) = 6x^5 - 8x^4 - 16x^3 - 8x^2 - 2x - 1$ and

$$x = \frac{y_i}{L} \tag{2.5}$$

N is the order of the Chebyshev polynomial, a denotes the Chebyshev coefficients (a_0 to a_5), y_i is the midpoint of the location of the i-th flap section numbered from root to tip and L as the length of half a wing span. Using only the Chebyshev coefficients within the control vector is also called virtual control, where a_0 to a_5 are then the virtual control variables (Nguyen et al. [42]). The advantage of using the shape function is that it automatically imposes the relative position limits, i.e. structural constraints and saturation limits that are imposed by the elastomer material. Not using virtual commands for control design, the controls would exceed the VCCTEF constraints, i.e. saturate or the control design is unfeasible (Tal [61]).

2.3.3. Control Design

In 2015, E.Tal and Nguyen [46] focused in their study on three control objectives namely traditional pilot-following command (symmetric flight modes), drag minimization and aeroelastic mode suppression. In this research, three different controllers have been designed: a Linear Quadratic Regulator (LQR), a Linear Quadratic Gausian controller (LQG) and a LQG with active drag minimization control. For the purpose of the control approach, the flight dynamic model was linearized around a steady symmetric level condition. Also the model-order was reduced. A Kalman filter is added to estimate the aeroelastic states such as wing bending and wing twist. Multiple cost functions address different control objectives. For aerodynamic performance optimization, it is assumed that the drag coefficients can be estimated accurately via a drag-polar model. However, it is mentioned that drag minimization has still many technical challenges such as sensing and control. The study concludes that the controller was able to damp out aeroelastic modes effectively, while following the pitch control command. When engaging LQG with active drag minimization control, it finds a trim solution that corresponds a lower drag setting or higher lift-to-drag ratio. A higher lift-to-drag ratio of about 4% can be achieved during cruise steady-state climb and level flight, compared to control without an active drag minimization algorithm.

Ferrier et al. [18] followed up on the drag minimization task. In her approach she uses recursive least squares to estimate the aerodynamic coefficient parameters and then calculate the optimal VCCTEF and elevator deflections using Newton-Raphson method. Similar to E.Tal, the optimal flight control for altitude-hold mode is accomplished via a simple LQR. The speed and Mach number are kept constant with an additional speed-hold mode. As required by the control design, the model is lineraized about a trimmed operating condition. Although the algorithm was able to reduce the drag count by 2.9, it is not able to accurately estimate the aerodynamic coefficient parameters for the wingtip bending. Furthermore a constant Mach number is assumed and sensor noise is not considered. Also, measuring in drag counts does not give a good indication about total fuel burn or lift-to-drag behavior, i.e. the improved efficiency.
The most recent development is dealing with the 4th and last objective of the multi-objective flight control approach, namely gust and maneuver load alleviation. Three controllers are designed and tested. All of which include a LQG as a baseline and an additional component such as model reference adaptive control or observer-based robust control with loop transfer recovery. First results are expected to published in mid 2017.

2.4. Conclusions

This chapter has given a short overview of the VCCTEF design and current developments in control. Section 2.1 to 2.2 and 2.3 answer the research sub-questions 1.(a) and 1.(b) respectively.
The novel control effector promises significant fuel savings by actively changing the wing shape in flight as desired. Recent control designs mainly focused on flutter and aeroelastic mode suppression, maneuver load alleviation as well as drag minimization. However, there are several challenges that the controllers are facing:

1. The amount of control surfaces that can be controlled increases significantly. The size of the control vector is $16x3x2 = 96$ for the VCCTEF only while for a conventional aircraft it is 4 (one flap section and one aileron per wing).

2. The controllers are designed in combination with a (reduced-order) model of the VCCTEF aircraft. Accuracy of such model never matches 100% with the real-life case and even controllers with well tuned control parameters and perfect results in simulation may not be sufficient for final implementation.

3. When using linear controllers it is required to linearize the full flight dynamic and aeroservoelastic model about a steady condition, which introduces additional model inaccuracies.

4. With the high amount of control surfaces the probability of malfunctions increases as well. As there is a dependency of all sections and segments, a potential failure of a single flap segment immediately effects all other segments. This requires the control system to be adaptive to certain failure modes.

This enumeration answers the research sub-question 1.c and directly forms the baseline for the performance metric that needs to be established for the upcoming research. It is clear, given the enumeration above, that the controller needs to be robust, adaptive and accurate for larger state space systems.

Table 2.2: Performance Metric

EXPERIMENTS	LEARNING PERFORMANCE			CONTROL PERFORMANCE	
Complexity in #states	**Convergence rate** in epochs	∅ **Training time** in sec	**Success rate** in %	**Accuracy** offset error	**Rise time** in sec
2,3,4 ...	number of epochs to reach a certain threshold	inverse of ∅ training rate	$\frac{\#successes}{\#runs}$	$\lvert(y_{ref} - y_{ss})\rvert$	$t_r = t_T - t_0$

As the VCCTEF project is still situated in the experimental phase and the model accuracy may not represent the real life case completely, model based algorithms are not desired and adaptative characteristics become a beneficial requirement as well. Using a Reinforcement Learning controller, as proposed in this thesis, those requirements are accounted for. Additionally the control designs do not require linearizations around any operating point. The rate of adaptability and the success rate of the controller to converge to an optimal control-law are to be examined. As the experimental runs typically require a large number of trials with random initialized control parameters (i.e. weights in case of artificial neural networks, see section A), a single run can be misleading as the choice of initial parameters might have a bigger influence on the performance between the proposed algorithms. The results are therefore reported for over a minimum of 20 successful runs, excluding failed trials as they would worsen the variance measure and average-learning-time. Failed runs will be represented within the column of '% of learning success' (view table 2.2 for the proposed performance metric). The problem with this is that it may not be a clear choice to go for an algorithm if one has a higher success rate, but lower overall performance or vice versa. So an additional performance parameter is included as adopted by Tesauro and Janssens [63]. The average training time is the inverse of average training rate, where the training rate is defined as the inverse of the time required to pass a certain error threshold per run. In general, the number of trials are limited to a certain number at each run. The precision of the best control samples of each algorithm is determined by the offset error, rise time and/or settling time. Using the performance metric as shown in table 2.2 and recording the results for systems with different complexity (denoted by the number of states that are controlled) should give a clear lead to what controller is more beneficial for the VCCTEF design in the future. The three columns under learning performance in the list heavily depend on the design of the parametric function approximators. They are an essential part of the reinforcement learning controllers as described in section 4.1 and 4.2.

3

Reinforcement Learning

This chapter provides the relevant background information of Reinforcement Learning (RL) as introduced by Sutton and Barto [60], starting with a description of the RL problem framework and concluding with the most basic algorithm to solve RL optimization problems: Dynamic Programming (DP). It covers the discrete RL cases only. An extension to continuous state / action spaces is given in subsequent section 4.

3.1. Markov Decision Process

Reinforcement Learning is a framework that is inspired by the most intuitive way of how animals learn, namely by interacting with the environment (Lewis and Vrabie [34]). Generally speaking, an *agent* (decision-maker) is applying actions to the *environment* (process) over a period of time and receives a numerical *reward* for every action it is executing. The reward can be considered as an evaluative feedback, giving a measure of performance for each action. Eventually, the goal for the agent is to optimize its behavior such that it receives the maximum sum of rewards in the long run. The sum (total amount) of rewards is also referred to as the *return*. Figure 3.1 depicts the agent-environment interaction graphically.

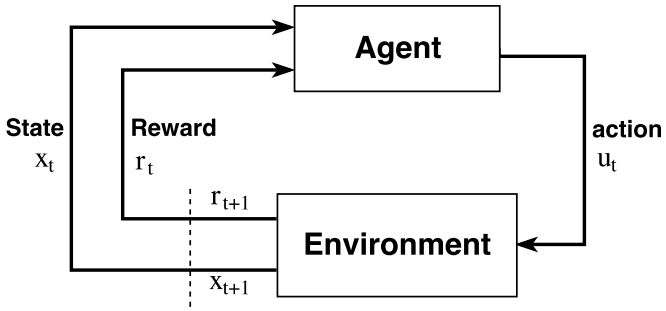
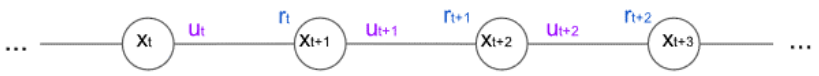


Figure 3.1: The agent-environment interaction in Reinforcement Learning (Sutton and Barto [60])

The time-variable is usually discrete, yielding in a sequential decision-making problem Busoniu et al. [11]. With every time-step t, the agent decides on an *action* $u_t \in U(x_t)$ based on the current *state* $x_t \in X$. The decision is therefore independent of states that have been visited beforehand. U and X are finite sets of possible actions and states respectively. Clearly, the information of the state variables that fully characterize the current state of the environment needs to be available and perceived by the agent at all times. Consider the following infinite-horizon task with states x, actions u and rewards r:



The sequence as shown above implies an important property of the agent-environment framework: The transition from state x_t to x_t+1 is independent of past events and only depends on the current state x_t and action u_t. That is, the one-step dynamics and expected reward at t+1 can be predicted solely given x_t and u_t. This

property is called the *Markov property*. If the agent-environment set-up satisfies the Markov property, it is called a *Markov decision process* (MDP)[60] Reinforcement Learning algorithms are tools to solve optimization problems that are modelled as a MDP. That is, a framework that is defined by the tuple $\langle X, U, f, \rho \rangle$, with

X	Finite state space with all $x \in X$
U	Finite action space with all $u \in U$
$f(x_t, u, x_{t+1})$	State transition function for state x_{t+1} based on state x_t and action u_t
$\rho(x_t, u_t, x_{t+1})$	Reward function of the transition $\{x_t, u_t, x_{t+1}\}$

Starting at an initial state x_0, the agent applies action u_0 to the environment. As a consequence, it finds itself in a new state x_1. The mapping from any state x_t and action u_t to a new state x_{t+1}, is determined by the state transition function $f : X \times U \rightarrow X$, that is:

$$x_{t+1} = f(x_t, u_t) \tag{3.1}$$

f is characterized by the environment or the system dynamics, while the specification of the environment is referred to as the *task*. After each transition, the agent receives a numerical reward $r \in R$ as an immediate evaluation of the action and the new state. It is defined by the finite reward function $\rho : X \times U \rightarrow R$:

$$r = \rho(x_{t+1}, u_t) \tag{3.2}$$

Being located in the subsequent state x_1, the agent choses a new action u_1 and the process starts over again. This loop can repeat ad infinitum *(continuous tasks)* or stops with specified terminal conditions and repeats a specified number of times (*episodic tasks*). Any goal-oriented learning problem that is set-up in this RL framework is therefore characterized by only three signals passing between the agent and the environment: the signal that fully describes the current state (state signal), which in turn serves as a base for the subsequent decision (action signal) and the evaluative feedback that defines the agents goal (reward signal). The behavior of the agent, i.e. its action-sequence, is dictated by its policy π; with π being a function that maps states to actions $\pi : X \times U \rightarrow [0, \infty)$. This can be expressed as:

$$u_t = \pi(x_t) \tag{3.3}$$

Applying u_t in x_t results in a transition to x_{t+1} under the condition that x_t is not a terminal state (usually denoted with subscript capital letter T). The overall goal in RL is to maximize the total return (sum of rewards) over the complete trajectory path generated by π.

The Agent-Environment Interaction (as it is depicted in figure 3.1) is the general framework of a *reinforcement learning problem* and can be shaped to many different applications. For example, in terms of control theory the agent is equivalent to the controller and the environment is the system that needs to be controlled. In RL it is essential to establish the learning problem-framework rather than defining a learning method. That is, designing the agent-environment interface and defining the reward function such that it unambiguously defines the ultimate goal of the task. In any case, the goal is to find the sequence of actions (control-law) that maximizes the sum of rewards over the complete time-period. As mentioned before, the time variable is usually discrete, but the framework can be extended to continuous time as well (Bertsekas and Tsitsiklis [7], Doya [15], Werbos [70]). However, continuous-time variants are not explicitly covered in this thesis as discrete-time cases have been developed to a far more mature level in the past decades.

Given an MDP with finite state and action space, transition probabilities and reward function, how does the agent learn then from the rewards it is receiving every time-step? This is done by memorizing the returns and linking them to values of the states; values that describe how "good" or "bad" a state (state-action pair) is with respect to other states. They are forming the so called value functions.

3.2. Value Functions & Rewards

The expected sum of rewards from any state $x_t \in X$ is described by the value (cost-to-go) function V (Sutton and Barto [60]):

$$V^{\pi}(x) = E_{\pi}\{\sum_{i=0}^{\infty} \gamma^i r_{i+1} | x_t = x\} \tag{3.4}$$

where $E_\pi\{\cdot\}$ is the expectation operator based on policy π. This estimated return is also called the value function. It is the agents memory that describes the relationship between the policy that the agent is following and the total amount of rewards it is collecting along the way. That is, $V^\pi(x_t)$ is the expected value of return when starting in state x_t and following a policy π thereafter. It averages the actual received returns R for any state, while the average converges to the real value as the number of visits to the states goes to infinity. Equation 3.4 can also be formulated in a different way, namely

$$Q^\pi(x,u) = E_\pi\{\sum_{i=0}^{\infty} \gamma^i r_{i+1} | x_t = x, u_t = u\} \tag{3.5}$$

This is the so called *Q-function* or *action-value function*, which expresses the expected return when starting in x_t, applying action u_t and then follow a policy π. Both, V^π and Q^π are usually unknown and can be estimated from experience. Knowing the value-function gives rise to the fundamental solution of RL problems as it indicates the maximum return that the agent could receive in the long run from any state. In order to weigh the importance of future rewards along the trajectory for the current estimation, the discount factor or forgetting factor γ is introduced, with $\gamma \in [0,1)$. When γ is near to 1, the agent is considering all rewards along the trajectory with the same "importance" while being near 0, only recent/near-future rewards give rise to the value of the current state. The total discounted return for an infinite horizon (H_{inf}) is then:

$$R = r_1 + \gamma r_2 + \gamma^2 r_3 ... = \sum_{i=0}^{\infty} \gamma^i r_{i+1} \tag{3.6}$$

With r being determined by the reward function r (eq. 3.2). In case of a N-step, finite horizon task, this equation would change to

$$R = \sum_{i=0}^{N} \gamma^i r_{i+1} \tag{3.7}$$

With N being a finite positive integer. In the 1950's, Richard Bellman expressed Equation 3.4 and 3.5 in a recursive form:

$$V^\pi(x) = E_\pi\{\rho(x_{t+1}, u_t) + \gamma V^\pi(x_{t+1})\} \tag{3.8}$$

$$Q^\pi(x,u) = E_\pi\{\rho(x_{t+1}, u_t) + \gamma Q^\pi(x_{t+1}, u_{t+1})\} \tag{3.9}$$

3.8 and 3.9 are called the *Bellman equations* or *dynamic programming equations* (Bellman [4]). They are functional equations, which means that the solution to this equation is a function. In terms of optimality, Bellman stated that "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision (Bellman [4])." Equation 3.8 and 3.9 can be modified and expressed in terms of *Bellman's principle of optimality*:

$$V^*(x) = \max_u E_\pi\{\rho(x_{t+1}, u_t) + \gamma V^*(x_{t+1})\} \tag{3.10}$$

$$Q^*(x,u) = E_\pi\{\rho(x_{t+1}, u_t) + \max_{u_{t+1}} \gamma Q^\pi(x_{t+1}, u_{t+1})\} \tag{3.11}$$

The $\max_u$ operator means to maximize the function (maximum return) by choosing the best actions u. Those actions are determined by the optimal policy, while the optimal policy is denoted by superscript '*'. Finding the optimal behavior and therefore finding the solution to this maximization problem can be found via means of RL algorithms. There exist many different methods, but all of them originated from the most basic form: *Dynamic Programming*.

3.3. Dynamic Programming

The first branch of methods to solve reinforcement learning problems was developed by Richard Bellman in the 1950's and is known today as *Dynamic Programming* (Bellman(1957a)Howard (1960). It is a solution method that assumes the environment to be a finite MDP. That is, the dynamics are defined by the transition function $f(x_t, u_t)$, with a given finite state-action space X and $U(x)$ as well as the expected immediate reward

r for all $x \in X$ and $u \in U$. The key element of DP is the symbiosis of policy evaluation - approximating the value function for a given policy - and policy improvement, i.e. altering the policy according to the maximum expected return. It is straight forward to find an optimal policy once V^* (or Q^*) are found that satisfy Bellman's principle of optimality given in equation 3.10 and 3.11. DP is a discrete method, meaning that it is only able to process discrete state and action spaces. Continuous problems can be formed into a 'grid world' by quantizing X and $U(x)$ which results in approximate solutions rather then exact solutions in most cases.
The DP solution is heavily based on recursion, i.e formulating the Bellman equation into an iterative update law such that the value functions and policy can be estimated. With the number of updates going to infinity, the estimation shall converge to the real value.

Policy Evaluation Starting with any arbitrary policy π the related value function V^π can be evaluated. This is called the *policy evaluation* (Sutton and Barto [60]). In context of computer programming, the bellman equation is translated into an iterative update rule:

$$V_{k+1}(x) = E_\pi\{\rho(x_{t+1}, u_t) + \gamma V_k(x_{t+1})\} \tag{3.12}$$

where k is the number of update sweeps to the value function at state x. A non-trivial solution is guaranteed if $\gamma < 1$ or the task is episodic. The value function is arbitrarily initialized first and then updated by means of recursion. If $k \to \infty$ the approximation is said to converge to V^π. The update rule applies to each state x and the new value of each state is updated based on the expected immediate reward when taking action u_t and the old value of the successor state $V_k(x_{t+1})$. Updating the value estimate based on the estimates of successor states is referred to as *bootstrapping*. Obviously, this requires to occupy a chunk of memory for an array with the size of the quanitzed state-space X. The algorithm is

Algorithm 1 Policy Evaluation, retreived from Sutton and Barto [60]

Initialize $V(x)$ and $\pi(x)$ arbitrarily for all states $x \in X$ and define error bound ϵ (small positive number)
while $\Delta V > \epsilon$ **do**
  for `each` $x \in X$ **do**
    $v = V(x_t)$
    $V(x_t) = E\{\rho(x_{t+1}, u_t) + \gamma V^\pi(x_{t+1})\}$
    $\Delta V = max(\Delta V, |v - V(x_t)|)$
  end for
end while

Policy Iteration Having determined the value function for a given random policy, it is time to find an improved policy if possible. This process is called *policy improvement*. The policy is changed at all states according to a new *greedy* policy $\pi'(x)$, which maps state to actions such that the total return is at least equal or greater than the return given under policy $\pi(x)$, that is:

$$V^{\pi'}(x) \geq V^\pi(x) \tag{3.13}$$

the update rule for the greedy policy gets:

$$\pi'(x) = \arg\max_u E\{\rho(x_t, u_t, x_{t+1}) + \gamma V^\pi(x_{t+1})\} \tag{3.14}$$

Where the $\arg\max_u$ term expresses to maximize expression by choosing the corresponding value of u. The originally estimated value function obtained by evaluating policy π, i.e. $V^\pi(x)$, is now kept constant in this update loop and provides the information to determine a better policy π'. As can be expected, the new policy is then evaluated once again, leading to new state-values and an approximation of the value function under policy π'. Finding $V^{\pi'}(x)$ may lead to another even better policy π'' and so on. This sequence, i.e. iteratively improving the value function and policy by policy evaluation and policy improvement, is called *policy iteration*. The complete algorithm is as follows:

As described in the previous section 3.3, the policy evaluation is an iterative loop itself and may require a couple of sweeps through the state-set to sufficiently approximate V^π. This requires computation time, but

Algorithm 2 Policy Iteration, retreived from Sutton and Barto [60]

Initialize $V(x)$ and $\pi(x)$ arbitrarily for all states $x \in X$ and define error bound ϵ (small positive number)
procedure POLICY EVALUATION
  while $\Delta V > \epsilon$ **do**
    for `each` $x \in X$ **do**
      $v = V(x_t)$
      $V(x_t) = E\{\rho(x_{t+1}, u_t) + \gamma V^{\pi}(x_{t+1})\}$
      $\Delta V = max(\Delta V, |v - V(x_t)|)$
    end for
  end while
end procedure

procedure POLICY IMPROVEMENT
  constantPolicy ← true
  for `each` $x \in X$ **do**
    $\pi' = \pi(x_t)$
    $\pi(x_t) = argmax \arg\max_u E\{\rho(x_{t+1}, u_t) + \gamma V(x_{t+1})\}$
    if $\pi' \neq \pi(x_t)$ **then**
      constantPolicy ← false
    end if
  end for
  if constantPolicy **then**
    Stop
  else
    go to Policy Evaluation
  end if
end procedure

alternately repeating function 2 and 3 in algorithm 2 eventually governs the optimal value function V^* and along with that the optimal policy. In theory it would need an infinite number of iterations to converge to the exact optimal value, but in practice the simulations are stopped whenever the change of value function updates is lower than a specified value ε, also called the terminal condition.
Dynamic programming is a very robust tool to solve optimization problems and finding the solution is guaranteed for finite MDP's. It is essential for the theory in RL. To give a better understanding, a simple example is given below.

Example 3-3-1: Suppose an agent needs to find the shortest path from any location to either point A or B on a fenced square. It is assumed that there are no obstacles.

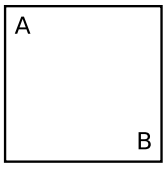


Figure 3.2: The agent shall reach A or B from any point on the square as fast as possible

In a continuous world, the fastest way to connect to A or B, is obviously a straight line. However, in order to solve this problem by using dynamic programming the state and action space needs to be discretized, which is:

The world is formed into a 4x4 grid with the allowed action set U at all non-terminal states X with $U = \{north, east, south, west\}$ and $X = \{1, 2, 3, ..., 14\}$. For any transition the agent receives a penalty of $r = -1$, while the transition function is known. At the terminal states (marked in gray) the simulation is reset. Whenever the agent is moving into the fence, it will receive a reward of -1 and will find itself in the same state. The learning process is shown in the figure below. The numbers in each grid denote

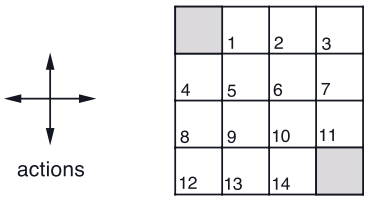
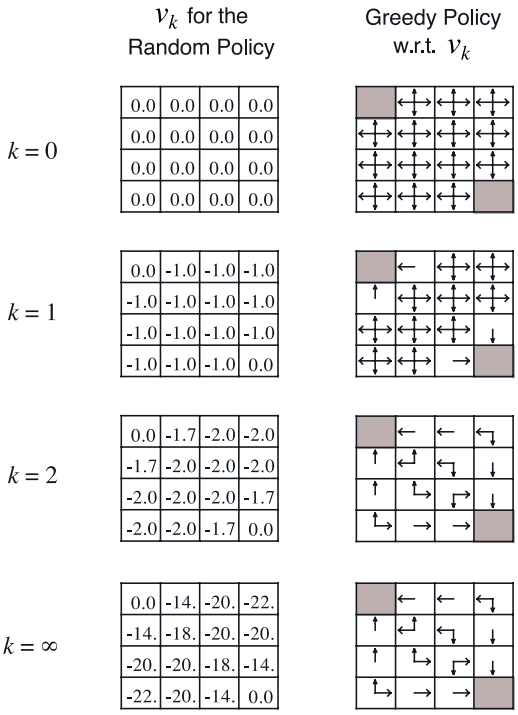


Figure 3.3: The discrete state-space with actions $u \in \{north, east, south, west\}$

the value of the state and the arrows show the greedy policy according to V at iteration k. It is an undiscounted, episodic task meaning that $\gamma = 1$ and t is finite. First V is initialized to be zero for all states and the random policy has equal probability of choosing an action from U at any state.
Assuming to be in position 5 at time step $k = 1$, the random policy denotes equal probabilities of choosing any action from U, e.g choosing to move north has the probability of one fourth. Performing an action leads to a reward of -1 and the agent ends up in a new state with an associated value of -1. Doing this for every possible action yields:

$$V_2^{\pi}(5) = \frac{1}{4} * \sum_{i=1}^{4} (r_i + V_1^{\pi}(x_{t+1})) = \frac{1}{4}(-2-2-2-2) = -2.0 \tag{3.15}$$

Where $x(t+1)$ denotes the successor states 1,6, 9 and 4. The rewards are determined by the reward function $\rho(x) = -1 \Leftrightarrow$ x is not a terminal state. This computation is repeated for every state until V_k is sufficiently estimated. After 3 iterations, the value function already provides enough information to find the optimal policy. That is, with $k \to \infty$ the value function does change slightly but the relations stay the same, i.e. the policy is already optimal and does not change anymore.



Sequence of policy evaluation (r) and policy improvement(l), retreived from Sutton and Barto [60]

This is a simple example. However, the disadvantages of DP become more apparent with the growing complexity of a problem. The execution time and/or memory requirements usually takes up to $O(n^2)$, as the value function needs to be estimated for every single state in several sweeps. That is, if the state-space X or action-space U is increasing in dimensions, the computational demand is exponentially growing as well due to the

two intertwined iterative loops. In literature this problem is referred to as the *curse of dimensionality* (Bellman [4]). In addition, DP works only for discrete problems and quantizing the continuous state-action space may lead to additional inaccuracies in the solution. In the example above, a better solution may be found by approaching more the continuous case with using a finer grid or provide more action choices, for example moving south-east. Doing so however, increases the complexity of the problem and the agent requires more time learn and find an optimal policy.
Real-life control problems are usually far more complex, working in continuous state and/or action spaces and may not provide full information of the environment. For those problems DP is not a practical solution. Another major drawback for dynamic programming solutions is namely, that it requires a complete and accurate model of the environment, which is seldom present.
In 1977 Werbos presented a new approach to approximate dynamic programming solutions by making use of function approximators and updating their parameters by gradient-descent. It is called 'Heuristic Dynamic Programming' and sets the starting line of 'Adaptive Critic Designs'.

3.4. Conclusions

This chapter discussed the concept of reinforcement learning and included a description of important terms such as markov decision processes, rewards and return, value function and policy as well as dynamic programming. All modern RL methods that evolved throughout the years rely heavily on that theory. With this chapter, research sub-question 2.(a). can be answered.

4

Approximate Dynamic Programming

Dynamic programming is a powerful tool to solve finite MDP's, given a perfect and discrete model of the environment. However, real-life control applications are usually comprised of large, continuous state-action spaces where the underlying dynamics of the system are usually unknown. This makes it computational infeasible, or even impossible to use DP. *Approximate Dynamic Programming* methods (ADP), try to overcome the curse of dimensionality by approximating the value function and the optimal policy with the use of parametric functions (Bertsekas and Tsitsiklis [7], Sutton and Barto [60]). The key feature is to cover the infinite continuous domain of the state-action space with a finite number of parameters. In that perspective, the heuristic process can be split into two entities, namely the *actor* and the *critic* (Barto et al. [3], Witten [71]). The actor is a parametric function that approximates the optimal control-law (policy) while the critic evaluates the current policy by estimating the corresponding value function. ADP methods are therefore often referred to as Adaptive – Critic Designs (ACD) or Adaptive Dynamic Programming (also ADP) in literature. Konda and Tsitsiklis [28] classified the ADP forms into three categories:

1. Critic-only methods exploit the best estimation of the optimal value function, instead of optimizing over the policy space. The idea is to approximate the optimal solution to the Bellman equation 3.10 or 3.11 and with that finding indirectly an optimal policy. The downside is that this method is not guaranteed to find an optimal solution.

2. Actor-only methods provide a continuous control signal using a parameterized policy, while the parameters are updated by adding current gradient estimates of the cost function with respect to the latest actor parameters. However, since only recent estimates are used to update the parameters, the actor does not incorporate learning from previously gained knowledge and the learning process is rather slow due to high variance in the policy gradient

3. Actor-Critic methods aim to combine the advantages of actor-only and critic-only methods. That is: a) generating a continuous control signal due to the actor and b) improving the convergence properties of the controller by using a critic to update the actors function parameters (limiting the large variance of the policy gradients).

Due to their beneficial properties are actor-critic methods preferred reinforcement learning algorithms to solve real-life control tasks. Their theory and architecture is the fundamental backbone to most ACD designs. The first ACD design family was presented by P. Werbos in the early 1990's [68–70] and has been widely employed to different applications ever since. His work includes the basic forms: Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP), Global Dual Heuristic Programming (GDHP) and their action depended versions (AD-). Prokhorov et. al. describes those ACD formulations in detail in his paper [51] and suggests two new modifications to the original GDHP design. All ACD forms have a similar structure but they can be distinguished by the input-output characteristics of the critic and the requirements of plant model derivatives (Lendaris [33]):

> **Critic Input:** The critic receives generally the complete state information by the plant (or plant model). In action-dependent cases the actor – output (control signal) is also linked to the critic and serves as an additional input.

Critic Outputs: For HDP the critic approximates the value function (usually denoted as J(t)) while for DHP the critic-outputs are the derivatives of the value function with respect to each state, i.e. $\frac{\partial J(t)}{\partial \mathbf{x}}$ (denoted as $\lambda_x(t)$). In the GDHP case the output contains both, J(t) and $\lambda_x(t)$.

Plant Model Derivatives: When employing artificial neural networks as function approximators (see appendix A), the backpropagation path differs with each ACD structure. Depending on the design, the update chain rule requires various derivatives where some derivatives must be obtained by a plant model.

ACD's have very beneficial properties. The complexity of a reinforcement learning problem is distributed and handled internally by two separate components, i.e. the learning and controlling component. Furthermore, the algorithms belong to the class of non-linear control which means that the plant does not need to be linearized around a certain operating point. The controller interacts with the plant directly and works in the continuous state-action domain. Other benefits are 1) there is no previous established complex model required 2) the controller is adaptive to changing operating conditions and 3) it is robust in a noisy environment.
This chapter explains the concept of Heuristic Dynamic Programming (section 4.1), Dual Heuristic Programming (section 4.2) and ACD applications that have been developed in recent years (section 4.3). For all ACD's the original implementation is based on Artificial Neural Networks (ANN). Their characteristics of universal function approximation and non-linear mapping have made them a popular tool in the field of reinforcement learning and artificial intelligence. A detailed overview of the concept of ANN's and the backpropagation algorithm is given in the appendix of this report.

4.1. Heuristic Dynamic Programming

Heuristic Dynamic Programming (HDP) is the fundamental form of ADP techniques. There are four entities that directly, or indirectly interact with one another: the actor, the critic, the plant and the model of the plant. It can be used as an online-learning algorithm. Starting in state $x(t)$, the critic estimates the associated value and the actor determines an action u(t) that is applied to the plant and the plant model. Having estimated the new state $\hat{x}(t+1)$ by the plant model and receiving a reward $\rho(x(t+1), u(t))$, the critic computes the associated costs for the new state. Actor, model and critic parameters are then updated according to their respective training error. The algorithm and the update laws are depicted graphically in figure 4.1.

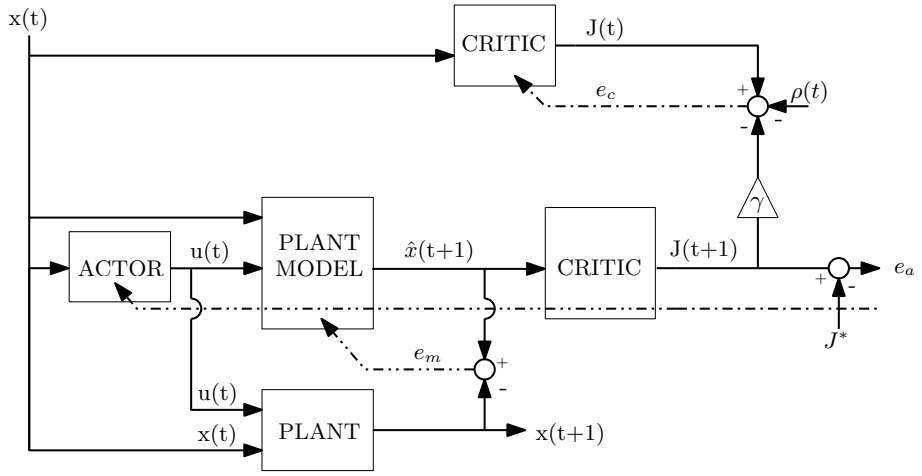


Figure 4.1: HDP algorithm at two consecutive moments in time. The backpropagation paths are indicated by the dotted line. e_a, e_c and e_m are the relative training errors for the actor, critic and model functions. $\rho(t)$ and $\hat{x}(t+1)$ are the reward and predicted one-step ahead state respectively.

Training Algorithm

The critic approximates the value function (discounted cumulative rewards) as it is described by equation 3.10:

$$J(x(t)) = f_c(x(t); w_c) \approx V(t) \tag{4.1}$$

Where J and V are the value function, x is the state vector and $f_c(\cdot, w_c)$ is the critic function approximator with w_c parameters. An error measure E_c needs to be minimized by the critic and is defined by the difference between the true value function and J(t). Although the optimal value function is mostly unknown, the recursive property as explained with equation 3.10 makes it possible to approximate it. For that, the temporal difference error (TD) is defined which is the difference between the left and right hand side of the Bellman equation:

$$e_c(t) = J(x(t)) - (\rho(x(t+1), u(x(t))) + \gamma J(x(t+1)) \tag{4.2}$$

$$E_c(t) = \frac{1}{2} {e_c}^2(t) \tag{4.3}$$

$J(x(t))$ is the output of the critic at time t while the desired value is expressed by the term $(\rho(x(t+1), u(x(t))) + \gamma J(x(t+1))$. The critic is therefore trained forward in time, which is of great importance for real-time applications (Prokhorov and Wunsch [51]). Minimizing this error concurrently relates to the optimal value function. The effect of each parameter on the respective training error is determined by the gradient of the error measure with respect to the parameters, i.e. $\frac{\partial E_c}{\partial w_c}$. The parameters are therefore updated with a small step size η_c opposite of the direction of the gradient (gradient descent):

$$w_c \rightarrow w_c(t+1) = w_c(t) - \eta_c \frac{\partial E_c(t)}{\partial w_c(t)} \tag{4.4}$$

Where η_c is the positive learning rate. The term $\frac{\partial E_c}{\partial w_c}$ is determined by applying backpropagation:

$$\frac{\partial E_c(t)}{\partial w_c(t)} = \frac{\partial E_c(t)}{\partial e_c(t)} \frac{\partial e_c(t)}{\partial J(x(t))} \frac{\partial J(x(t))}{\partial w_c(t)} \tag{4.5}$$

The dashed lines in figure 4.1 show the backpropagation paths. However, the main component that needs to be trained in HDP is the actor as it represents the actual controller of the the plant. Consider the optimal control signal u^* for any state x, the actor is denoted by $f_a : X \rightarrow U$ with parameters w_a

$$u(t) = f_a(x(t); w_a) \approx u. \tag{4.6}$$

As the critic serves as some sort of evaluation of the control signal, there must be an exchange of information between the actor and critic (Kampen et al. [25]), so the actor can learn. This is done by determining the gradient of the value function J(x(t+1)) with respect to actor parameters w_a. In other words, it simply backpropagates J through the critic and plant model function approximator to obtain $\frac{\partial J(x(t+1))}{\partial u(t)}$. Then the gradient and parameter update gets:

$$\frac{\partial E_a(t+1)}{\partial w_a(t)} = \frac{\partial E_a(t+1)}{\partial e_a(t+1)} \frac{\partial e_a(t+1)}{\partial J(x(t+1))} \frac{\partial J(x(t+1))}{\partial x(t+1)} \frac{\partial x(t+1)}{\partial u(t)} \frac{\partial u(t)}{\partial w_a(t)} \tag{4.7}$$

$$w_a \rightarrow w_a(t+1) = w_a(t) - \eta_a \frac{\partial E_a(t+1)}{\partial w_a(t)} \tag{4.8}$$

Where η_a is the positive learning rate of the actor. The training objective is to minimize the error measure E_a which is the squared error e_a and defined as:

$$e_a(t) = J(x(t)) - J^* \tag{4.9}$$

$$E_a(t) = \frac{1}{2} {e_a}^2(t) \tag{4.10}$$

By definition, the learning error of the actor is the difference between the estimated value $J(x(t+1))$ and the optimal value J^*, also referred to as the goal value. J^* heavily depends on the reward function $\rho(x,u)$ and must be defined by the control designer. One approach is to set the reward function to a maximum of zero when reaching an optimum and negative otherwise, so $\rho(x) \leq 0$ (Russell and Si [54]). Then the goal value J^* becomes zero as well. Keep in mind that ρ is the only information provided to the learning agent and it has to represent unambiguously the goal of learning process. Defining the reward function is therefore not always straight forward.

HDP is an algorithm that is based on approximated plant dynamics with which the derivatives $\frac{\partial J(x(t+1))}{\partial x(t+1)}$ and

$\frac{\partial x(t+1)}{\partial u(t)}$ can be determined. If there is sufficient sample data from previous tests and/or executions available, then the plant model may be trained offline already before implementing it in the control structure. It may also be trained online in series with the actor and the critic if there is no data available a priori. The structure of the model is denoted by

$$\hat{x}(t+1) = f_m(x(t), u(t); w_m) \approx f(x(t), u(t)) \tag{4.11}$$

where f_m is the plant model function with w_m model parameters and f are the actual plant dynamics. Like the other function approximators, the training objective is to minimize an error measure that usually comprises of the squared model error e_m. Where e_m is the difference between the model output and the output of the plant:

$$E_m(t) = \frac{1}{2} {e_m}^2(t) \tag{4.12}$$

$$e_m(t) = \hat{x}(t) – x(t) \tag{4.13}$$

And then, applying backpropagation the model parameters are updated as follows:

$$w_m \rightarrow w_m(t+1) = w_m(t) - \eta_m \frac{\partial E_m(t)}{\partial w_m(t)} \tag{4.14}$$

Although it has an effect on actor and critic learning, the model training performance itself is independent from them. Training all function approximators online makes the algorithm adaptive and robust to changing operating conditions. Obviously, the rate of change of the system dynamics is limited by the learning speed of the algorithm itself. The environment or plant dynamics may not alternate faster than the agent can learn. Another important learning property in HDP is the relation between actor and critic learning rate. The critic needs to be able to learn the value function with respect to a given policy before the policy is updated and changing. This means that the critic learning rate is a factor higher than the actor learning rate in order for the algorithm to converge. A summary of the algorithm is given in 3.

Algorithm 3 HDP

1: Initialize $t = 0$, w_a, w_c, (w_m) and $x(t_0)$
2: **procedure** START LOOP
3: Compute the output of critic at time t
4: $J(t) = f_c(x(t); w_c(t))$
5: Compute the actor output at time t
6: $u(t) = f_a(x(t); w_a(t))$
7: Estimate the one-step ahead state via plant model
8: $\hat{x}(t+1) = f_m(x(t), u(t); w_m(t))$
9: Compute expected reward
10: $r = \rho(x(t+1), u(t))$
11: Compute the output of the critic at t+1
12: $J(t+1) = f_c(x(t+1); w_c(t))$
13: Compute the critic error e_c at time t from equation 4.2
14: Backpropagate $J(t+1)$ and update actor parameters w_a with 4.8
15: Update the critic parameters w_c with 4.4
16: Plant proceeds to new state
17: $x(t+1) = f(x(t), u(t))$
18: if applicable, compute plant model error and update it's parameters w_m with 4.13 and 4.14
19: $t \leftarrow t+1$
20: **end procedure**
21: **goto** 2 and repeat N times

4.2. Dual Heuristic Programming

The dual heuristic programming optimization scheme is very similar to the HDP structure, except that the DHP critic parametric function is approximating the derivative of the value function with respect to each

state. This leads to more complicated update rules and backpropagation paths for the actor and critic as indicated with dashed lines in figure 4.2. Just like in HDP the actor receives the state information $x(t)$ and maps it to a control signal $u(t)$, which is observed by the plant and plant model. In the meanwhile, the critic estimates the current value gradients with respect to the states $\frac{\partial J(t)}{\partial x(t)}$ as well as for the predicted one-step ahead state $\frac{\partial J(t+1)}{\partial \hat{x}(t+1)}$. Then, actor, model and critic parameters are updated according to their respective training error. Figure 4.2 depicts the algorithm and it's update laws graphically.

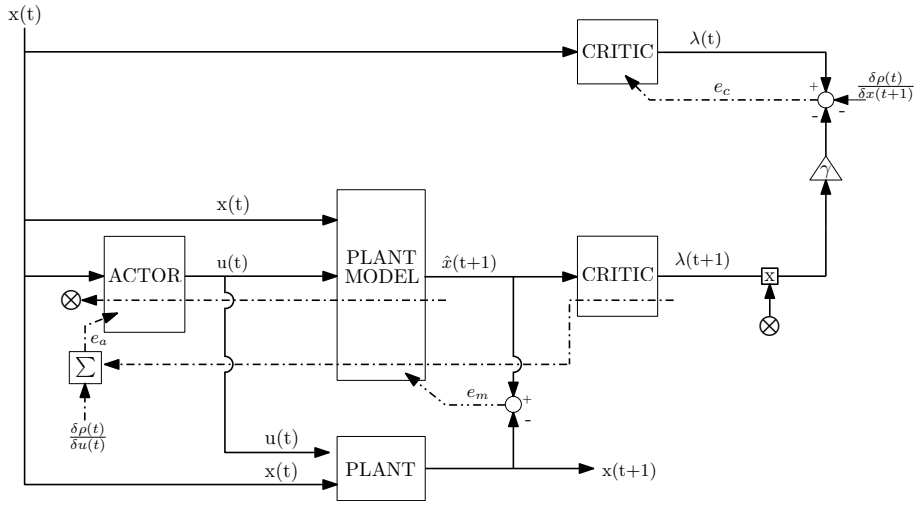


Figure 4.2: DHP algorithm at two consecutive moments in time. The backpropagation paths are indicated by the dotted line. e_a, e_c and e_m are the relative training errors for the actor, critic and model functions. The Symbol $\otimes$ represents the factors within the critic update scheme and are obtained from the actor and plant model. $\frac{\partial \rho(t)}{\partial(\cdot)}$ depicts the partial derivative of the reward function with respect to $(\cdot)$ and $\lambda(t)$ relates to the partial derivative of the value function with respect to *each* state $\frac{\partial J(t)}{\partial x(t)}$

Training Algorithm

Unlike in HDP the critic approximates the derivatives of the value function J with respect to the states x, usually denoted as λ:

$$\lambda(x(t)) = \frac{\partial J(x(t))}{\partial x(t)} = f_c(x(t); w_c) \tag{4.15}$$

This means that having a state vector with n elements, the critic outputs contains n elements as well, describing $\frac{\partial J(x(t)}{\partial x_1(t)}, \frac{\partial J(x(t)}{\partial x_2(t)} \ldots, \frac{\partial J(x(t)}{\partial x_n(t)}$. The update of the function parameters w_c gets more complicated as compared to HDP (Prokhorov and Wunsch [51]). Consider again the error measure E_c to be defined as the squared error:

$$E_c(t) = \frac{1}{2}\mathbf{e}_c{}^T(t)\mathbf{e}_c(t) \tag{4.16}$$

with

$$e_c(t) = \frac{\partial J(x(t))}{\partial x(t)} - \gamma\frac{\partial J(x(t+1))}{\partial x(t)} - \frac{\partial \rho(t)}{\partial x(t)} \tag{4.17}$$

where γ is the discount factor and ρ is the reward function, while the reward function derivatives depend on the design of the reward function itself. The terms $\frac{\partial(\cdot)}{\partial x(t)}$ are vectors containing all partial derivatives of $(\cdot)$ with respect to $x(t)$. The additional complexity of DHP lies in the added derivative terms and elongated backpropagation paths. Applying the chain rule gives us the unknown in equation 4.17:

$$\frac{\partial J(t+1)}{\partial x_j(t)} = \sum_{i=1}^{n} \lambda_i(t+1)\frac{\partial x_i(t+1)}{\partial x_j(t)} + \sum_{k=1}^{m}\sum_{i=1}^{n} \lambda_i(t+1)\frac{\partial x_i(t+1)}{\partial a_k(t)}\frac{\partial a_k(t)}{\partial x_j(t)} \tag{4.18}$$

where subscript j describes one of the n elements in the state vector, $\lambda_i(t+1) = \frac{\partial J(t+1)}{\partial x_i(t+1)}$ and n, m are the number of inputs to the critic and outputs of the actor respectively. The terms $\frac{\partial x_i(t+1)}{\partial x_j(t)}, \frac{\partial x_i(t+1)}{\partial a_k(t)}$ and $\frac{\partial a_k(t)}{\partial x_j(t)}$ are obtained following the dashed line in figure 4.2 with $\otimes$, while the first two are obtained by the plant model

and the last by the actor respectively. So, as the $e_c(t)$ vector contains n elements each element is computed by using equation 4.18 and:

$$e_{c_j}(t) = \frac{\delta J(x(t))}{\delta x_j(t)} - \gamma \frac{\delta J(x(t+1))}{\delta x_j(t)} - \frac{\delta \rho(t)}{\delta x_j(t)} - \sum_{k=1}^{m} \frac{\partial \rho(t)}{\partial u_k(t)} \frac{\partial u_k(t)}{\partial x_j(t)} \tag{4.19}$$

With that information the critic parameters are updated as follows:

$$w_c \rightarrow w_c(t+1) = w_c(t) - \eta_c \frac{\partial E_c(t)}{\partial w_c(t)} \tag{4.20}$$

Where

$$\frac{\partial E_c(t)}{\partial w_c(t)} = \frac{\partial E_c(t)}{\partial e_c(t)} \frac{\partial e_c(t)}{\partial \lambda(t)} \frac{\partial \lambda(t)}{\partial w_c(t)}. \tag{4.21}$$

This concludes the critic update scheme. The actor parametric function has the same objective as in HDP, namely approximating the optimal policy. It's parameters w_a are updated in a rather more complex form, by backpropagating $\lambda(t+1)$ through the critic and model network. The goal may be expressed as (Prokhorov and Wunsch [51]):

$$\frac{\partial \rho(t)}{\partial u(t)} + \gamma \frac{\partial J(t+1)}{\partial u(t)} = \frac{\partial \rho(t)}{\partial u(t)} + \gamma \frac{\partial J(t+1)}{\partial x(t+1)} \frac{\partial x(t+1)}{\partial u(t)} = 0 \tag{4.22}$$

$\frac{\partial J(t+1)}{\partial x(t+1)}$ and $\frac{\partial x(t+1)}{\partial u(t)}$ are obtained via the critic and plant model respectively and $\frac{\partial \rho(t)}{\partial u(t)}$ is provided by the reward function. Then the weight updates become:

$$w_a \rightarrow w_a(t+1) = w_a - \eta_a \left[\frac{\partial \rho(t)}{\partial u(t)} + \gamma \frac{\partial J(t+1)}{\partial x(t+1)} \frac{\partial x(t+1)}{\partial u(t)} \right]^T \frac{\partial u(t)}{\partial w_a(t)} \tag{4.23}$$

Also here, η_a is the positive learning rate of the actor and exponent T denotes the transpose. Prokhorov and Wunsch [51] states that the added complexity also relates to better performance as compared to HDP. The critic is provided more information during the training updates and it directly approximates the gradient of the value function with respect to the states. This is also the term that is found back in the actor update (i.e. $\frac{\partial J(t+1)}{\partial x(t+1)}$). The training loops are executed online and actor, critic and plant model parameters are updated simultaneously. If sufficient data samples are provided, the plant model may also be trained offline a priori. In general, the model training is exactly the same as it is for HDP. As a summary, the complete online algorithm is given by algorithm 4.

Algorithm 4 DHP

1: Initialize $t = 0$, w_a, w_c, (w_m) and $x(t_0)$
2: **procedure** START LOOP
3: Compute the output of critic at time t
4: $\lambda(t) = f_c(x(t); w_c(t))$
5: Compute the actor output at time t
6: $u(t) = f_a(x(t); w_a(t))$
7: Estimate the one-step ahead state via plant model
8: $\hat{x}(t+1) = f_m(x(t), u(t); w_m(t))$
9: Compute reward derivatives
10: $\frac{\partial \rho}{\partial \hat{x}}$ and $\frac{\partial \rho}{\partial u}$
11: Compute the output of the critic at t+1
12: $\lambda(t+1) = f_c(x(t+1); w_c(t))$
13: Compute the critic error e_c at time t from equation 4.17
14: Backpropagate $\lambda(t+1)$ and update actor parameters w_a with 4.23
15: Update the critic parameters w_c with 4.20
16: Plant proceeds to new state
17: $x(t+1) = f(x(t), u(t))$
18: if applicable, compute plant model error and update it's parameters w_m with 4.13 and 4.14
19: $t \leftarrow t+1$
20: **end procedure**
21: **goto** 2 and repeat N times

4.3. Applications

Adaptive critic designs are powerful algorithms to solve complex control problems. Especially their adaptive characteristics and online learning property make them a well respected tool for fast acting systems. In addition they are able to control non-linear plants directly without the need of linearization around an operating point and no complex model of the plant is required a priori. In the last decade there have been several applications of ACD's in research and industry. Some results of a few relevant examples are presented in this section.

Prokhorov and Wunsch [51] gave a concise overview of the existing ACD structures and presented all forms from simple ACD's such as ADHDP and HDP to advanced algorithms such as DHP and GDHP. In their experimental studies they applied those algorithms on a longitudinal autolanding problem of a simplified commercial aircraft as it is described in Miller et al. [37]. The aircraft itself is subjected to wind shear and turbulent wind gusts. Following the desired landing profile of the Instrument Landing System (ILS) and meeting the landing requirements, the controls achieved the following results:

Table 4.1: Autolanding of a commercial aircraft Prokhorov and Wunsch [51]

Gusts N(0,1,5)		% out of 600 test trials				
		GDHP	**DHP**	**HDP**	**ADHDP**	**PID**
Trained with wind shear only	tight success	73	71	50	1	0
	loose success	99	99	98	98	99
Trained with wind shear and wind gusts	tight success	71	70	45	0	0
	loose success	98	98	97	97	98
Average number of training attempts to land		1000	1000	100	100	N/A

Here, tight success relates to an application where the runway is shortened by 30%. The algorithms are trained for a number of trials (1000 or 100). It is claimed that HDP and ADHDP did not improve in performance after 100 trials in training significantly anymore. Table 4.1 presents clearly that GDHP and DHP achieve very similar performance and it seems that they obtain a much higher success rate compared to HDP and ADHDP.

In 2002, Venayagamoorthy et al. [66] compared HDP with DHP for neuro control of a turbogenerator where the control signal comprises of the deviation of the exciter voltage and the deviation of the turbine power. Turbogenerators are highly non-linear and fast acting systems with varying system dynamics when operating conditions are changing. As shown in figure 4.3, the results are promising. Both, HDP and DHP outperform a conventional AVR controller and Venayagamoorthy et al. concludes a better performance of the DHP algorithm as compared to HDP as well. For more details the reader is referred to the paper.

Ferrari and Stengel [17] applied DHP for flight control on a small business aircraft. The controller was pretrained offline with help of well-tuned, linear controllers. This way the researcher incorporates already existing knowledge to the design and ensures reliability. After the offline training phase the DHP algorithm is applied online and action and critic networks are further updated to improve performance. Implemented in a full-scale aircraft simulation the controller shows improved performance and the ability to adapt to unexpected conditions such as unmodeled dynamics, control failures, and parameter variations. However, it is also mentioned that adaptive critic methods do generally converge to an optimal policy over time, but in practice it is difficult achieve quick enough convergence to affect real-time performance (Ferrari and Stengel [17], Howard [22], Venayagamoorthy et al. [66]). Also the problem of tuning is still an issue.

There have been several other applications, like HDP for torque and air-fuel ratio control of a V8 engine (Liu et al. [35]) or direct HDP forms for helicopter stabilizations (Russell and Si [54])and damping oscillations of large power systems (Lu et al. [36]). All show promising results and motivate current research to investigate more.

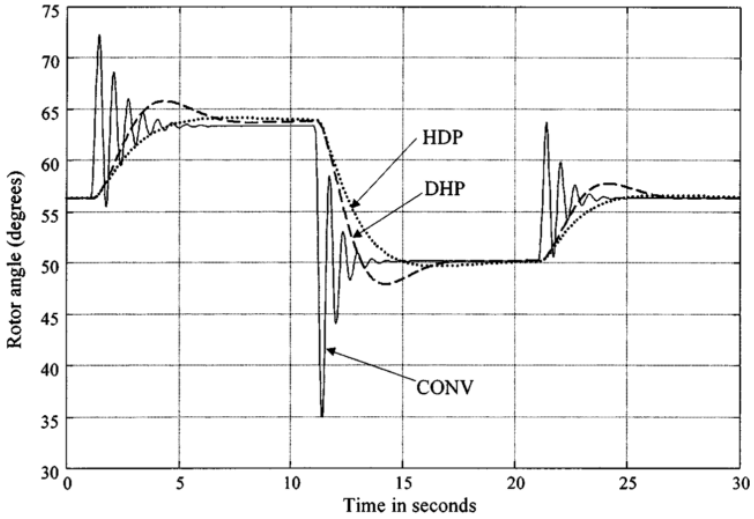


Figure 4.3: Rotor angle variation for ±5% step changes in the desired terminal voltage

4.4. Conclusions

ADP methods are a powerful tool to solve highly non-linear control tasks without the need of a complex model of the plant a priori. The first family of adaptive critic reinforcement learning has been presented by Werbos [70] in the early 1990's and has been widely employed to different applications in industry and research ever since. The two fundamental forms named Heuristic Dynamic Programming and Dual Heuristic Programming have been presented in detail in this chapter, revealing the theoretical difference between the two algorithms. Some details are and conclusions are quickly summarized here.

Prokhorov and Wunsch [51] mentioned that it is more valuable to obtain the derivatives of an optimization criterion then the optimization criterion itself. In a simple form of ACD, such as HDP, the actor receives this information only indirectly via backpropagation of the value function J(x) through the critic and model network. This may result in a too coarse actor behavior with a slight off-set to the desired point. One approach to improve the results is suggested by Borghese and Arbib [10], where the actor explores additional trajectories around the nominal one.

As part of advanced ACD's, the DHP method directly estimates the derivatives of J with respect to the states and is explicitly trained by the reward function derivatives $\frac{\partial \rho(t)}{\partial u(t)}$ and $\frac{\partial \rho(t)}{\partial x(t)}$. This has the advantage that the derivatives are directly provided to the actor rather than the value function itself. According to Prokhorov and Wunsch [51], the quality of such direct approximation is noted to be better than obtaining the indirect approximations. Furthermore the model network is used for both, the critic and the actor, in the training process. The update process itself is more complex, involving more derivatives and update rules which makes it more tedious to set-up the algorithm for a given problem.

In terms of practical differences it seems that DHP usually provides better results, with higher rate of convergence and learning. An example of direct comparison is given by the examples of neurocontrol of a turbo generator (Venayagamoorthy et al. [66]) and an autolanding of a commercial aircraft in [51].

The different modules, i.e. the actor, critic and plant model are parametric functions that approximate the optimal solutions. This way, the algorithm works directly with the (non-) linear plant in continuous state-action space. The functions itself can be any parametric function, but the most common method in adaptive critics is to use artificial neural networks. According to the theorems of Barron [2], it is known that artificial neural networks can approximate processes with many variables and less errors than other, more traditional approximators. A short summary about ANN's and it's working principles is given in the appendix A.

It is expected, given different levels of complexity and control tasks, that DHP is the better option to chose for aircraft flight control equipped with the VCCTEF. The preliminary analysis in the next chapter provides first results. With this chapter, research sub-question 2.(b) and 2.(d) can be answered. The appendix Acovers sub-question 2.(c)

5

Preliminary Analysis

In order to compare the performance of Heuristic Dynamic Programming (HDP) and Dual Heuristic Programming control (DHP) in practice, they are applied to the task of learning to stabilize an inverted pendulum in the upright position. Pole-balancing tasks or inverted pendulums are common practice for benchmarking advanced control system designs, because of multiple reasons. First of all the system dynamics are of low dimensions with few state and action variables. Next to that it is highly non-linear and inherently unstable. That is, without active control, the pole would not stay in the upright position, but fall to the stable point, i.e. hanging down. Second, it is easy to understand and visualize and can be modified in terms of complexity. The pendulum may be hinged to a torque motor directly while the torque can be sufficient to push the pendulum to the upright position, or it is under-actuated and the pendulum needs to swing up. In this case the pendulum is controlled directly, while it can also be controlled indirectly when placed on a cart (cart-pole-problem), where the cart is pushed by a force and so on. There have been many different designs and they all vary in terms of complexity. Another reason to use inverted pendulums for performance analysis of modern controllers is the easy set-up.
Section 5.1 explains in detail the experimental set-up, the system dynamics and the state-action variables. Before designing the HDP and DHP controller, a plant model is build that represents the dynamics of the inverted pendulum. The performance of such model including the design of the function approximator is described in section 5.2. Due to their characteristics of universal function approximation and non-linear mapping, artificial neural networks (ANN) are used for the control design. A detailed description of ANN including their update law and performance improvements is given in appendix A. HDP and DHP control structures are summarized in section 5.3 and 5.4 respectively. The simulation was conducted, using 3 different controllers: HDP, DHP and Proportional-Iterative-Differential control (PID). PID belongs to the class of linear classical control and is commonly used in industry. This shall provide an additional benchmark with respect to common linear control methods. Results are listed in section 5.5, followed by the conclusions in section 5.6.

5.1. Experimental Setup

The experiment is a pole balancing task where the controller needs to stabilize an inverted pendulum in the upright position. A picture of the set-up can be seen in figure 5.1. It contains an electric torque motor that is connected directly to a weightless link with length l. Attached at the end of the link is a point mass m. The motor itself is subjected to motor resistance r and a torque constant τ. Regarding the system dynamics, the pendulum can be described in the following state-space form:

$$I\ddot{\theta} = mgl\sin(\theta) - (c + \frac{\tau^2}{r})\dot{\theta} + \frac{\tau}{r}u \tag{5.1}$$

The corresponding variables are copied from Grondman [19] and are listed in table 5.2. It is assumed that the state information is completely available at all times, with the state vector $\mathbf{x}$ containing two variables. With θ being the angle of the link measured from the upright position and $\dot{\theta}$ being the angular rate that is positive when the pendulum is turning clock-wise.

$$\mathbf{x} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} \tag{5.2}$$

The action space is the voltage that is send to the motor and is limited to $u \in [-3.5, 3.5]$ V. With that voltage the controller is able to directly control the pendulum to the upright position from any angular position.

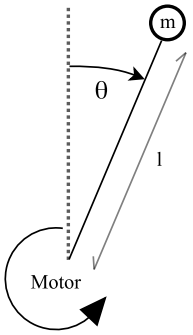


Table 5.1: The inverted pendulum experiment

Parameter	Symbol	Value	Units
Pendulum inertia	J	$1.91 \cdot 10^{-4}$	kgm^2
Pendulum mass	m	$5.50 \cdot 10^{-2}$	kgm^2
Gravity	g	9.81	m/s^2
Length of pendulum	l	$4.20 \cdot 10^{-2}$	m
Friction coefficient	c	$3 \cdot 10^{-6}$	$Nm\,s/rad$
Torque constant	τ	$5.36 \cdot 10^{-2}$	Nm/A
Rotor resistance	r	9.50	Ω

Table 5.2: Inverted pendulum parameters

5.2. Plant Model Structure

Since HDP and DHP are both model-based learning algorithms, a neural network model of the plant dynamics is needed. This is due to the fact that the actor and critic parameters are updated based on the model derivatives (upon others), which are:

HDP	DHP
$\frac{\partial \hat{x}(t+1)}{\partial u(t)}$	$\frac{\partial \hat{x}(t+1)}{\partial u(t)}$
	$\frac{\partial \hat{x}(t+1)}{\partial x(t)}$

Table 5.3: Model derivatives for HDP and DHP update rule

$\hat{x}(t+1)$ is the estimated state at the subsequent time-step, x(t) is the state at time t and u(t) is the control signal. A more detailed description of the update law per algorithm follows in the next sections. In a complete online learning structure the model is not pre-trained, but rather learned simultaneously with the actor and the critic while engaged in a task. However, this directly comes in hand with a poor estimation of the derivatives in table 5.3 which impedes convergence and delays the learning process. Therefore, the model is pre-trained before it is connected to the HDP and DHP learning algorithm.

Data-Set: First the data-set with input-output pairs [x,y] is created to which the model can be fitted on. This is done by randomly exciting the inverted pendulum itself for three seconds in a 100 runs. Each run starts with random initial conditions and is stopped at terminal time t_{end} or as soon as the angular velocity exceeds a value of 25π.
The final data-set consists of more than 20 thousand samples and is split into a training-set (70%) and a validation-set (30%) randomly. This way, the neural network model can be validated while detecting and preventing over-fitting. In order to prevent early saturation and slow learning of the neural networks, the whole data-set and the sample data during simulation is normalized / mapped from $\theta_{min-max}$, $\dot{\theta}_{min-max}$ and $u_{min-max}$ to an interval of W = [-1 1] (see more in appendix A.3). The max values of each variable are later used again to denormalize the outputs of the network to the real values.

Model Network Structure: The model is a feed forward neural network and consists of one hidden layer with ten sigmoidal neurons. Additionally, the input layer contains three linear neurons, one for the angle $\theta(t)$, the angular rate $\dot{\theta}(t)$ and the control signal u(t), while the output layer has 2 linear neurons for the one-step ahead prediction of $\hat{\theta}(t+1)$ and $\hat{\dot{\theta}}(t)$. The neural network structure can be seen in figure 5.1.

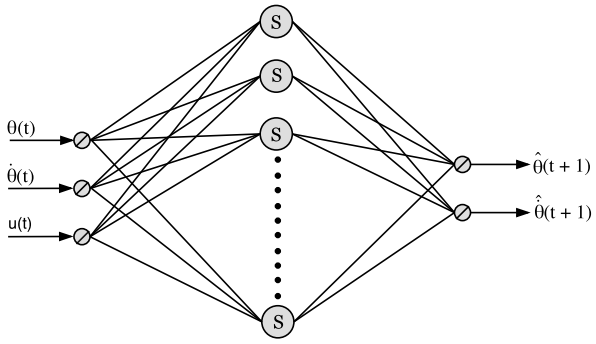


Figure 5.1: The model network architecture with three inputs, 10 sigmoidal hidden layer neurons and two linear output neurons

The model network is trained to identify the input-output behavior of the inverted pendulum. The ANN identifier tries to minimize the following error measure:

$$\mathbf{E}_M = \frac{1}{2N}\sum_x \mathbf{e}_M^T(\mathbf{x})\mathbf{e}_M(\mathbf{x}) \tag{5.3}$$

$$\mathbf{e}_M = \tilde{f}_M(\mathbf{x}, u; W_M) - f(\mathbf{x}, u) \tag{5.4}$$

with N being the total number of samples, $\tilde{f}_M(\mathbf{x}, u; W_M)$ the ANN output vector for input $\mathbf{x}$ and u and $f(\mathbf{x}, u)$ the desired states, i.e. the actual output of the plant given input $\mathbf{x}$ and u. Then, the input and output layer weights W_M and $b_M$are updated using the partial derivative of the error measure with respect to the weights, see appendix A for more details. The update law is performed by momentum-based Stochastic Gradient Descent (SGD), which includes the momentum factor μ_M and the regularization L_M. Each epoch sweeps through the data set with batch-size n and shuffles the data-samples thereafter:

$$V_M \rightarrow V_M(t+1) = \mu_M V_M(t) - \frac{\eta_M}{n}\sum_n e_M \frac{\delta e_M}{\delta W_M} \tag{5.5}$$

$$W_M \rightarrow W_M(t+1) = (1 - \eta_M L_M) W_M(t) + V_M(t+1) \tag{5.6}$$

and

$$b_M \rightarrow b_M(t+1) = b_M(t) - \frac{\eta_M}{n}\sum_n e_M \frac{\delta e_M}{\delta b_M} \tag{5.7}$$

V_M is a temporary variable to determine the momentum and is initialized to be zero. Using grid search for different hyperparameters such as the learning rate η_M, the momentum factor μ_M and the regularization rate L_M lead to the optimal settings as listed in table 5.4. The learning rate itself is adaptive, meaning that it decreases in time. Note that grid search is a very basic form to search for the best hyperparameter settings and may lead to sub-optimum. It does its job for a small amount of tweaking variables, but exponentially increases in computation time with an increasing number of hyperparameters and decision space. Therefore, it is recommended to use faster and more advanced optimization algorithms in the next phase of this thesis such as bayesian optimization.

Model Parameters	**Symbol**	**Value**
Learning rate	η_M	0.3
Momentum factor	μ_M	0.7
Regularization rate	L_M	0.01

Table 5.4: Model ANN parameters

Simulation & Results: The learning process is done by using SGD with batch size of 300 while the elements of all batches are shuffled after each episode. As the weights of the ANN are randomly initialized, the learning session is repeated 10 times with the same settings (i.e. same hyperparameters). See figure 5.2 to 5.3 for the

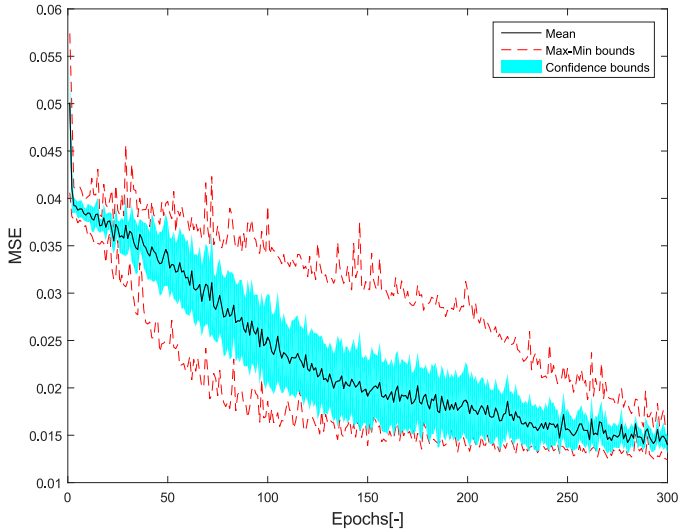


Figure 5.2: Results of Model-learning using the validation data-set, with 95% confidence and max-min bounds computed from 10 learning sessions, keeping the hyper parameters constant

results of convergence and final performance, using the hyper parameters from table 5.4. The mean squared error (mse) is computed with respect to the validation data-set and is averaged over the 10 learning sessions.

It can be seen that the SGD training successfully updates the model ANN parameters in 300 epochs. The mean squared error is gradually, but slowly decreasing with each epoch (see figure 5.2) and eventually converges to a roughly constant value at around $1 \cdot 10^{-2}$ with lower variance. The tendency is still decreasing, but the performance will not improve significantly anymore. Figure 5.3 displays a sample simulation and shows that the model accuracy is satisfying. Note that the sharp jumps in the graph are due to the discontinuity at π (hanging down) as the angle is normalized around that point, so it does not exceed $\pm\pi$. The discontinuity introduced some bias in the estimation and the approximation power of the model ANN diverges slightly at that point, but the differences are minor.

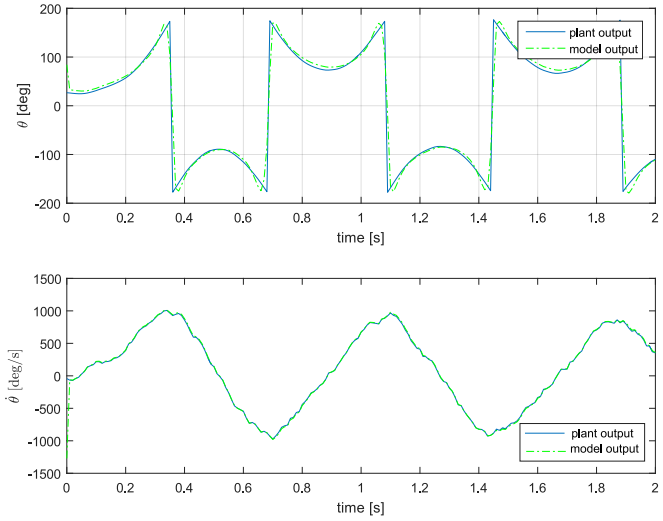


Figure 5.3: Performance of the model in simulation when subjected to pseudorandom inputs

Looking at the performance, the model network is accurate enough to give a good approximation of the derivatives listed in table 5.3. The model that achieved the lowest MSE, is used for the HDP and DHP algorithm.

5.3. Heuristic Dynamic Programming Control

HDP is a sample-based algorithm that can be trained online. The actor and critic parameters are updated each time-step according to the update law as explained in section 4.1. This section gives an overview of the applied HDP structure.

The Reward Function: A reward function is evaluating the performance of the actions at every time-step t. It is a continuous function that quadratically penalizes the angular distance of the pendulum to the desired upright position $\theta = 0$. Furthermore it takes the angular rate and control signal into account, by penalizing non-zero values of $\dot{\theta}$ and u. For HDP the reward function gets:

$$\rho(t) = -\mathbf{x}Q\mathbf{x}^T - Pu^2 \tag{5.8}$$

with

$$\mathbf{x} = \begin{bmatrix} \theta(t) & \dot{\theta}(t) \end{bmatrix} \qquad Q = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad P = 0.2$$

The Critic Neural Network: As mentioned in section 4.1, the critic tries to estimate the value function $J(\mathbf{x})$. In order to train the neural network and update its parameters, the temporal difference error is backpropagated through the network. The TD error is the difference of the left and right side of the Bellman equation 3.10. Eventually, the objective of the critic is to minimize $E_C(t)$ over time t:

$$E_C(t) = \frac{1}{2}\sum_t e_c^2(t) \tag{5.9}$$

$$e_c(t) = TD = J(\mathbf{x}(t)) - (\rho(\mathbf{x}(t+1), u(\mathbf{x}(t))) + \gamma J(\mathbf{x}(t+1)) \tag{5.10}$$

The critic neural network consists of one hidden layer with eight sigmoidal neurons, 2 linear input neurons for $\theta(t)$ and $\dot{\theta}(t)$ and one linear output neuron for $J(t)$. The structure can be seen in figure 5.4. Just like the model ANN the hyper-parameter settings are found via grid-search during multiple simulations. This time both, the critic and actor hyper-parameters, have to be tuned simultaneously. The convergence and performance of one neural network effects the convergence and performance of the other and makes tuning very difficult. The found hyper parameters are

Critic Parameters	Symbol	Value
Discount factor	γ	0.97
Learning rate	η_C	0.05
Momentum factor	μ_C	0.1
Regularization rate	L_C	0

Table 5.5: HDP Critic ANN parameters

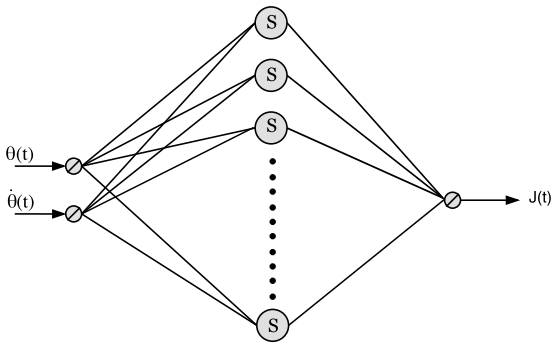


Figure 5.4: Critic neural network with 2 inputs, 8 sigmoidal neurons in the hidden layer and one linear output layer.

The Action Neural Network: The actor ANN is the actual controller of the system and maps the current angle and angular rate to a voltage signal u, which is then sent to the plant and the model. As part of reinforcement

learning it is approximating the optimal policy and learns based on the information of the critic and the model ANNs. Figure 5.5 shows the neural network structure. The output layer has one sigmoidal neuron that bounds the output of the neural network between -1 and 1. This is the bounded and normalized control signal and can be scaled to the actual voltage signal by:

$$u(t) = y_{actor}(t)u_{max} \tag{5.11}$$

Sigmoidal transfer functions are also present in the hidden layer, which contains 10 neurons. The input layer has two linear neurons, one for each state.

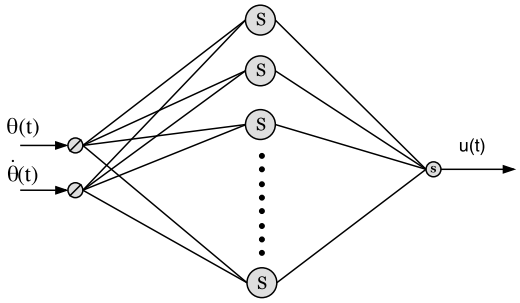


Figure 5.5: Actor neural network with 2 inputs, ten sigmoidal neurons in the hidden layer and two sigmoidal output neurons.

During simulation, the actor receives the state-information each time step and projects it to an action. The action is evaluated by means of the value function $J(\mathbf{x})$, which is the output of the critic neural network. J represents the expected total return from any state and since the maximum reward is zero ($\rho(t) \leq 0$) the optimal sum of reward would be zero as well, i.e. $J^* = 0$. Therfore, the actor tries to minimize the following squared error over time:

$$E_A(t) = \frac{1}{2}\sum_t e_A^2(t) \tag{5.12}$$

$$e_A(t) = J(\mathbf{x}) - J^* = J(\mathbf{x}) \tag{5.13}$$

The neural network updates are done in small steps into the direction of the gradient $\nabla E_A = \frac{\delta E_A}{\delta W_A}$, or $\nabla E_A = \frac{\delta E_A}{\delta b_A}$ with W_A and b_A being weights and biases of the actor ANN respectively. The step size itself is defined by the learning rate, momentum factor and regularization. Table 5.6 lists the tuned parameters. Note that the actor learning rate is smaller than the critic learning rate. This is necessary due to the fact that the critic requires some time to estimate the value function for a given policy, so it needs to learn faster than the actor.

Actor Parameters	**Symbol**	**Value**
Exploration rate	ϵ_0	0.1
Learning rate	η_A	0.01
Momentum factor	μ_A	0
Regularization rate	L_A	0

Table 5.6: HDP Actor ANN parameters

Exploration: In order to force the pendulum to visit different states the initial angle θ_0 is randomized per trial while the angular rate $\dot{\theta}_0$ is always zero. This supports the learning procedure, as the pendulum would otherwise visit mostly the states in the lower half of the circle (stable point) and the chances of visiting the optimal point [0,0] is relatively low. Having a surplus of data points in a small region of the state-space leads to overfitting of the ANN in that region while having a poor fit otherwise.
To stimulate exploration during a simulation run, the actor is doing a random action every ϵ steps. The so called exporation rate is initialized (ϵ_0 and degrades with the number of trials, such that the actor explores more at the start of the simulation and by the end, when learned the optimal policy, only follows greedy actions.

5.4. Dual Heuristic Programming Control

The DHP controller has a very similar set up as HDP. This section shortly reviews the actor-critic structure and learning parameters.

The Reward Function: Just like for HDP, the reward function quadratically penalizes the difference of the angular deflection and angular rates with respect to the nominal values [0 0] using equation 5.8 and

$$Q = \begin{bmatrix} 0.9 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad P = 0.2$$

With DHP, the reward function becomes a more active part in the actor and critic training. That is $\frac{\delta\rho(t)}{\delta\mathbf{x}(t)}$ is part of the derivatives to update the critic neural network weights and $\frac{\delta\rho(t)}{\delta u(t)}$ for the actor weights. The individual partial derivatives are as follows:

$$\frac{\delta\rho(t)}{\delta\theta(t)} = 1.8\cdot\theta(t) \qquad \frac{\delta\rho(t)}{\delta\dot{\theta}(t)} = 0.2\cdot\dot{\theta}(t) \qquad \frac{\delta\rho(t)}{\delta u(t)} = 0.2\cdot u(t)$$

The Critic Neural Network: The critic is approximating the derivative of the value function $J(\mathbf{x})$ with respect to the states. Those partial derivatives are denoted as λ_x, with x being any state of the state vector. So, in this example there are two partial derivatives λ_θ and $\lambda_{\dot{\theta}}$. The error measure for the minimization problem becomes:

$$E_C(t) = \frac{1}{2}\sum_t \mathbf{e}_c^T(t)\mathbf{e}_c(t) \tag{5.14}$$

$$e_{c_\theta}(t) = \frac{\delta J(t)}{\delta\theta(t)} - \gamma\frac{\delta J(t+1)}{\delta\theta(t)} - \frac{\rho(t)}{\delta\theta(t)} \tag{5.15}$$

$$e_{c_{\dot{\theta}}}(t) = \frac{\delta J(t)}{\delta\dot{\theta}(t)} - \gamma\frac{\delta J(t+1)}{\delta\dot{\theta}(t)} - \frac{\rho(t)}{\delta\dot{\theta}(t)} \tag{5.16}$$

with

$$\lambda_\theta(t) = \frac{\delta J(t)}{\delta\theta(t)} \qquad \lambda_{\dot{\theta}}(t) = \frac{\delta J(t)}{\delta\dot{\theta}(t)}$$

and

$$\frac{\delta J(t+1)}{\delta\theta(t)} = \lambda_\theta(t+1)\frac{\delta\theta(t+1)}{\delta\theta(t)} + \lambda_{\dot{\theta}}(t+1)\frac{\delta\dot{\theta}(t+1)}{\delta\theta(t)} + \lambda_\theta(t+1)\frac{\delta\theta(t+1)}{\delta u(t)}\frac{\delta u(t)}{\delta\theta(t)} + \lambda_{\dot{\theta}}(t+1)\frac{\delta\dot{\theta}(t+1)}{\delta u(t)}\frac{\delta u(t)}{\delta\theta(t)}$$

$$\frac{\delta J(t+1)}{\delta\dot{\theta}(t)} = \lambda_{\dot{\theta}}(t+1)\frac{\delta\dot{\theta}(t+1}{\delta\dot{\theta}(t)} + \lambda_\theta(t+1)\frac{\delta\theta(t+1)}{\delta\dot{\theta}(t)} + \lambda_{\dot{\theta}}(t+1)\frac{\delta\dot{\theta}(t+1)}{\delta u(t)}\frac{\delta u(t)}{\delta\dot{\theta}(t)} + \lambda_\theta(t+1)\frac{\delta\theta(t+1)}{\delta u(t)}\frac{\delta u(t)}{\delta\dot{\theta}(t)}$$

Critic Parameters	Symbol	Value
Discount factor	γ	0.97
Learning rate	η_C	0.1
Momentum factor	μ_C	0
Regularization rate	L_C	0

Table 5.7: DHP Critic ANN parameters

Note that the derivatives of the one-step-ahead state with respect to the current state ($\frac{\delta\hat{x}(t+1)}{\delta x(t)}$) as well as $\frac{\delta\hat{x}(t+1)}{\delta u(t)}$ is given by the model network. The hat symbol above $\theta(t+1)$ and $\dot{\theta}(t+1)$ has been omitted for readability. The derivative of the action u with respect to the current state is coming from the actor network.
Figure 5.6 shows the critic neural network. There are eight sigmoidal neurons in the hidden layer, 2 linear

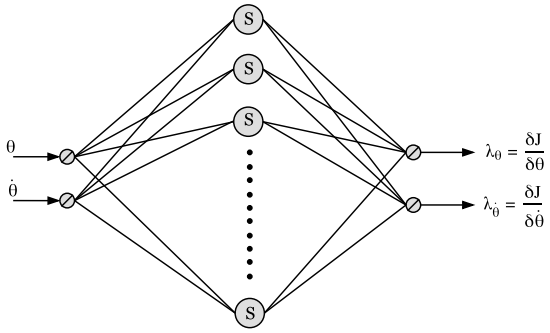


Figure 5.6: Critic neural network with two inputs, eight sigmoidal neurons in the hidden layer and two linear output layer.

input neurons and 2 linear output neurons. Table 5.7 lists the optimal hyper parameters as they have been found by grid-search.

The Actor Neural Network: The actor neural network for DHP is exactly the same as it is for HDP in terms of structure, i.e. same number of neurons and hidden layers as well as one sigmoidal output layer in order to include saturation limits of the control signal. The hyper parameter settings change slightly and are listed in table 5.8.

Actor Parameters	Symbol	Value
Exploration rate	ϵ_0	0.1
Learning rate	η_A	0.05
Momentum factor	μ_A	0
Regularization rate	L_A	0

Table 5.8: DHP Actor ANN parameters

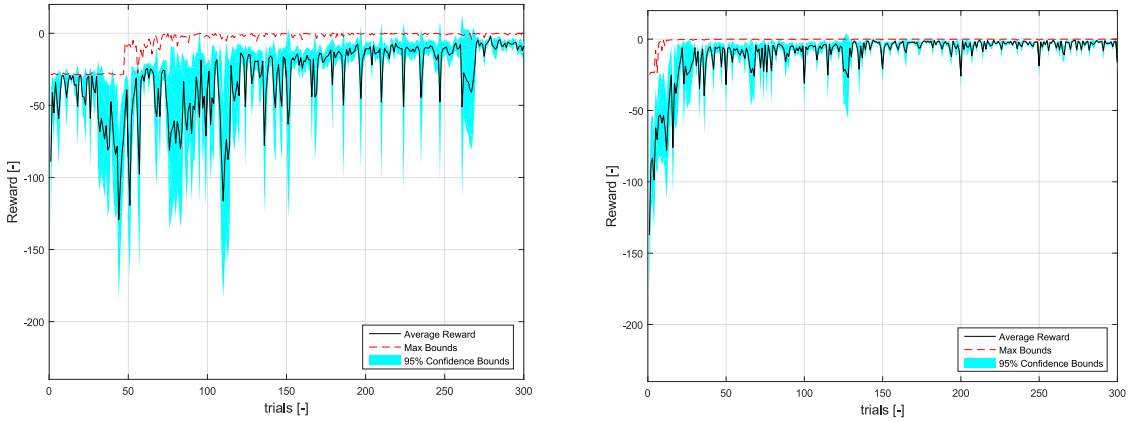
Also the exploration is done in the same way as it is for HDP. This provides some consistency in the results.

5.5. Results

This sections shows the results of the experiments. The simulation time is discrete with time-step of $\delta t = 0.01s$. First, the HDP and DHP algorithms are compared with respect to their learning performance. Then, subsection 5.5.2 presents the control characteristics for the inverted pendulum with the two RL algorithms as well as an additional linear controller namely PID.

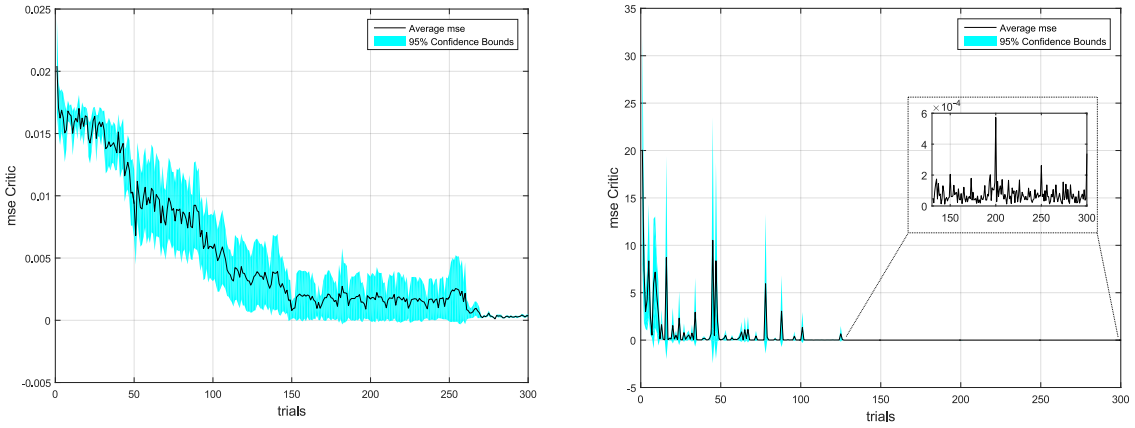
5.5.1. Training Performance

A good measure of learning performance is the cumulative reward over the set of experiments. Each experiment is repeated 20 times, each containing 300 trials. The maximum cumulative reward that can achieved is 0. See figure 5.7a and 5.7b for details. These results also include failed runs. HDP failed to converge to an optimum in 5 experimental runs out of 20 while for DHP it was only one training session that did not converge. This gives a success rate of 75% and 95% respectively. HDP convergence rate seems much more coarse and still contains a lot of variance at trial 200. DHP on the other hand is rather steady and with low variance near the optimum at approximately 150 trials. The red lines mark the maximum of all runs per trial and shows that HDP does not visit the optimum before 60 attempts while for DHP it takes about 20 trials.
The critic-training performance as shown in 5.8a and 5.8b is much more clear for HDP. Starting with a maximum mean squared error of about 0.018 and converging to a stable, steady behavior at $1 \cdot 10^{-3}$ at 150 trials. The DHP algorithm contains high peaks of up to 13 on average in the first attempts, but quickly reduces to a minimal value. At trial 100 the mse is below 1 and after 150 attempts the critic error approaches values in the scale of $1 \cdot 10^{-6}$.



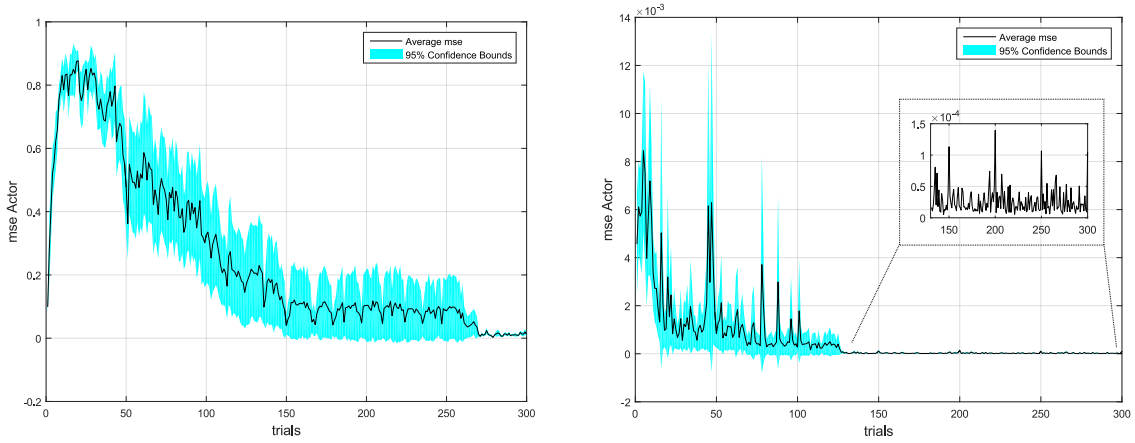
(a) HDP (b) DHP

Figure 5.7: Average cumulative reward and confidence bounds per trial of HDP and DHP control



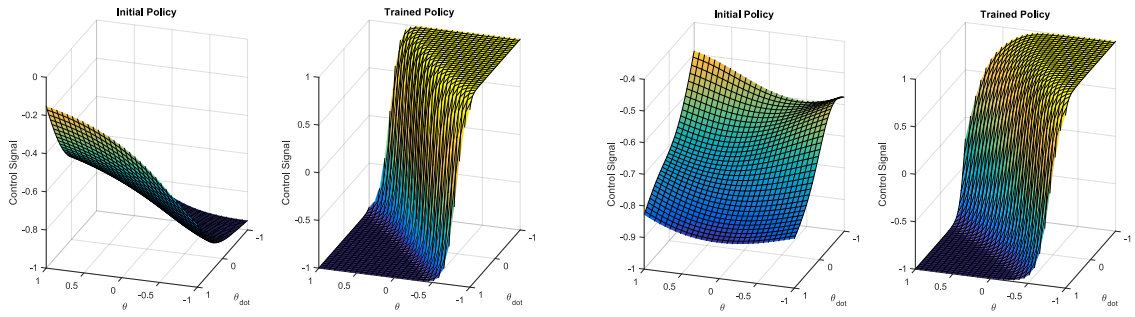
(a) HDP (b) DHP

Figure 5.8: Critic mean squared error



(a) HDP (b) DHP

Figure 5.9: Actor mean squared error



(a) HDP (b) DHP

Figure 5.10: Initial and trained policy after 300 trials with normalized state-action space. The max-min values of -1 and 1 correspond to the control signal [-3.5, 3.5] volt; θ $[-\pi, \pi]$ radians and $\dot{\theta}$ $[-25\pi, 25\pi]$ radians per second.

The actor is the actual control system of the plant. Figure 5.9a and 5.9b shows the training process for HDP and DHP respectively. In HDP the mean squared error is reduced to a rather noisy but stable minimum of around $1 \cdot 10^{-2}$ after approximately 150 trials. DHP, on the other hand approaches a minimum rather quickly at around 100 attempts and approximates with an mse of less than $1 \cdot 10^{-5}$. Though the values differ significantly by being a factor of 10^{-2} smaller when compared to HDP, both algorithms also work in a different scale. So the value itself shall not be compared in the conclusive remarks. It is more the convergence rate, variance and consistency that is of importance.

When looking at the trained policy in figure 5.10a and 5.10b, it can be seen that it consists of very flat regions near $\theta = \pm\pi$, i.e hanging down position. So the controller learns to apply maximum torque in either direction, whenever the pendulum is located in the lower half. If the pendulum reaches exactly $\pm pi$ (in the figure noted as ± 1) with $\dot{\theta} = 0$, then the controller can barely chose between a max/min control scheme as both options are desired.
Getting closer to the optimal point of [0, 0] the policy shape is a steep slope, with very fine adjustments at the optimum, i.e. fine control signals near zero. Figure 5.10a and 5.10b also shows clearly the transformation of an initial, random policy to the optimal policy.

5.5.2. Control Performance

Chosing the actor of both algorithms with the best control behavior from all experimental runs, the response to a disturbed pendulum is shown in figure 5.11.

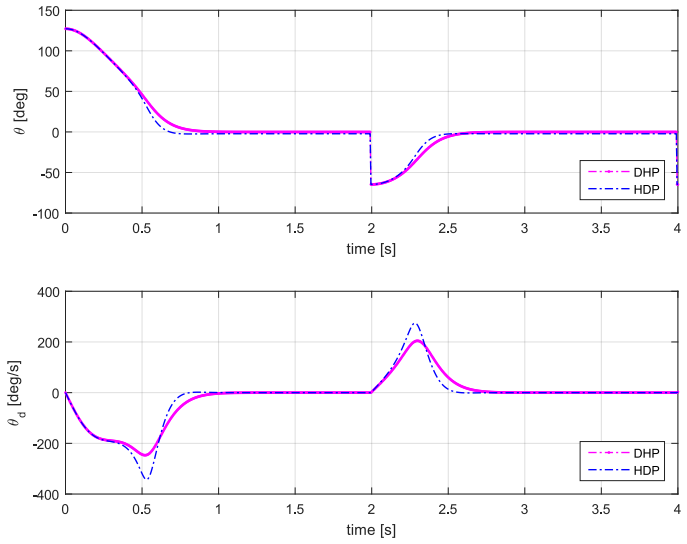


Figure 5.11: Plant response with HDP and DHP control. Disturbances are applied at t=0s and t=2s at 125 and -60 degrees respectively

Disturbance was initiated at $t_0 = 0[s]$ and $t_1 = 2[s]$. Both controller successfully stabilize the pendulum from 125 degrees within 0.7 to 0.8 seconds after the disturbance occurred. The rise-time to reach ±5 degrees around the nominal point is approximately 0.6 and 0.7 for HDP and DHP respectively. HDP often showed an offset error of about 2-5 degrees while DHP usually controls the pendulum very closely to 0. Also for -60 degrees deflection both controllers push the pendulum to an optimum within 0.5 s.

Comparison with Linear Control: In order to benchmark HPD and DHP against classical control theory, a third controller is included. Proportional-Iterative-Differential control (PID) is one of the most commonly used industrial control system. It would be desirable to stabilize the inverted pendulum in the upright position using linear control theory, but as mentioned before the pendulum is highly non-linear. For that matter, the system is linearized around the desired operating point, i.e. $\theta = 0\,\text{deg}$, assuming small-angle theorem, with $|\theta| \leq 12\,\text{deg}$. The PID control parameters are tuned using the Matlab simulink PID function. The tuning objective is to create a fast acting PID while limiting overshoot and transient response time. The specific control parameters are listed in table 5.9.
It can be seen that all controls manage to stabilize the pendulum. PID has a fast rise-time of 0.2 seconds, but with a slight overshoot of 2-3 degrees. It settles to a steady state after around 0.6 seconds. DHP is slightly slower in rise time (0.4 s) but settles faster to a steady state in 0.5 s at zero. The previously mentioned offset error with HDP control is now clearly visible. HDP pushes the pendulum to the nominal point in 0.3 seconds, but overshoots and 'gets stuck' at a steady state of ±2 degrees.

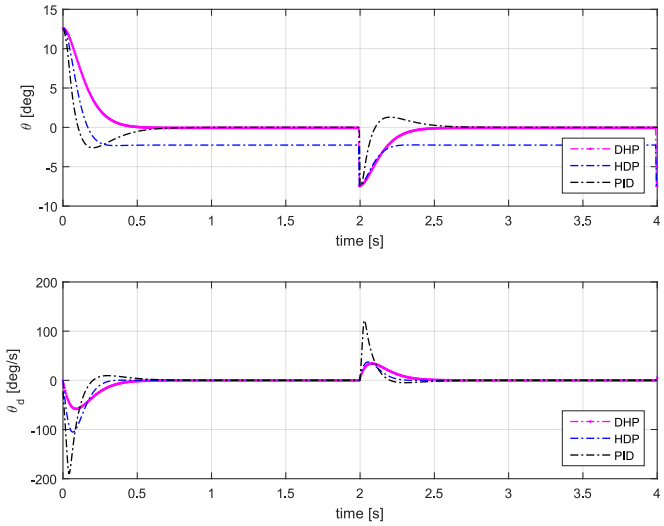


Figure 5.12: Performance of HDP and DHP with respect to classic linear control. Disturbances are applied at t=0s and t=2s at 12 and -8 degrees respectively

Table 5.9: PID control parameters

PID Parameters	Value
Proportional (**P**)	1.6273
Interal (**I**)	7.3649
Derivative (**D**)	0.0859

5.6. Conclusions

In this chapter, HDP and DHP have been applied on an inverted pendulum experiment. First a model network was pre-trained on a data-set, matching validation data with a mean squared error lower than $1 \cdot 10^{-2}$. Then, the actor and critic networks were connected and trained. The hyperparameters for all modules were found using grid search. It was soon clear that the hyperparameter values had a significant effect on the overall performance of the algorithms. Grid search is a cumbersome and time-consuming method, but eventually

gave a set of parameters that provided satisfying results. Nevertheless, the results as shown in table 5.10 and 5.11 change completely when tuning the parameters differently. A good tuning is therefore essential. The listed results have been found to be the best results for both algorithms.

During training DHP showed better performance by faster convergence and learning speed as well as more stable behavior. It was able to train the modules within 100 trials with lower variance than HDP. The success rate was 95% which shows more reliable behavior as compared to HDP with 75%. Especially the failed trials introduced more variance and coarse results of the HDP algorithm.

Looking at the control performance itself, both algorithms achieve very good results by stabilizing the pendulum in less than a second from any operating point. HDP reached the nominal values by 0.1s faster than DHP, but showing a slight offset error of 2-5 degrees. Although the offset error may be introduced by some integration/computation errors, it was noticeable that HDP mostly stabilized the pendulum in a range near the optimum in *all* experimental runs. This was even the case when penalizing the action to a higher degree, i.e. increase the P value in the reward function 5.8. Exploration near the optimum helped to converge to a slightly better set-point, but still an offset was present. The reason for this is due to the fact that the agent can not distinguish the minor differences between the optimal and near optimal points when approaching $\theta = 0$. The error becomes unnoticeably small. This is also an explanation of the rather coarse training behavior, as the agent is still updating the actor which leads to jump-arounds around the nominal trajectory.
DHP instead, uses the approximation of the derivative of the cost function with respect to the states for updates and does not seem to have this problem. It always converged to the exact optimum ($\pm 10^{-6}$ difference on average). This is due to the derivatives as they can still pose high values near the optimum, especially when the optimum is a very sharp, narrow point. On the other hand, HDP seems to result in a rather more aggressive control policy, leading to shorter rise-times. On average the HDP controller approached the nominal region 0.1 to 0.2 seconds faster than DHP. These characteristics should be kept in mind as some plants may require a more aggressive and fast acting control solution.

When compared to linear control, both controllers showed similar if not better results. DHP steered the pendulum to the optimum in 0.1 s faster then the PID, without any overshoot. HDP, on the other hand, controlled the pendulum to a steady-state even in 0.3 s (0.2 s faster than DHP), but with an offset error.

All in all it can be concluded that both algorithms show strong behavior, but DHP provided a better solution with faster convergence and learning speed. It's results were more reliable and stable as compared to HDP. A summary of the results is given in table 5.10 and 5.11 .

Table 5.10: Learning and control performance of Actor and Critic for HDP and DHP. These are the results for controlling to the nominal point from 120 degree initial deflection

ACTOR	LEARNING PERFORMANCE			CONTROL PERFORMANCE	
	Convergence rate in trials	∅ **Training time** in sec	**Success rate** in %	**Accuracy** offset error	**Rise time** in sec (θ_0 = 125deg)
HDP	150	338	75	± 5	0.6
DHP	120	256	95	0	0.7
CRITIC	**Convergence rate** in trials	∅ **Training time** in sec	**Success rate** in %		
HDP	150	338	75		
DHP	100	213	95		

Table 5.11: Comparison of DHP and HDP with a linear PID controller after 12 degrees deflection

	CONTROL PERFORMANCE	
	Accuracy offset error	**Settling time** in sec (θ_0 = 12deg)
PID	0	0.6
HDP	±2	0.3
DHP	0	0.5

III

Additional Results

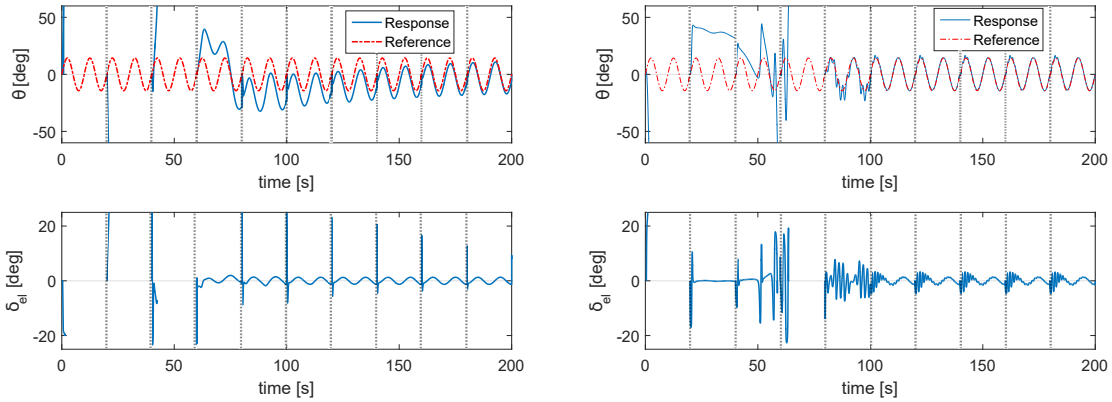
6

Offline Learning Behavior

This chapter gives an example of the offline learning progress of HDP and DHP. It shows, how the two agents are developing with each trial and how the initial policy evolves to the final control law. The last section gives a short sensitivity analysis on the initial weights.

6.1. Policy evolution

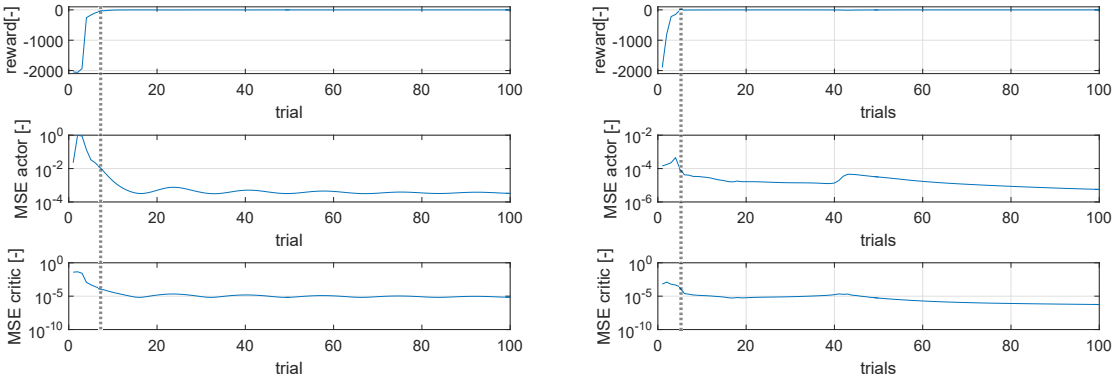
The following graphs visualize the pitch response of the F-16 aircraft together with the desired reference. They are depicted on a time-series plot, while a trial ends at every 20 second interval and the simulation is resetted to the trim condition. With 100 trials this means that there is 2000 seconds of total simulation time. In order to make it more readable, only the first 200 seconds are shown, which corresponds to 10 trials. The controls are not perturbed in this case.
Figure 6.1a and 6.1b show an example of the learning progress, where both algorithms finalize in an optimal policy. Clearly there is a learning process with every trial. HDP starts with extreme elevator deflections from ±25[deg] and then slowly converges to follow the reference signal. As there is no exploration, HDP is only slowly re-learning that the initial spike in the control-signal is not favorable. DHP also starts with quite strong elevator deflections near the saturation points, but quickly adapts.
Similarly, Figure 6.2a and 6.2b show the learning progress with respect to the reward, actor and critic mean squared error (MSE). Note that only the first 10 trials correspond to the response plots of Figures 6.1a and 6.1b. It can be seen that the actor error increases slightly in the first 2 trials of the experiment. This may be caused by the critic learning progress. The actor parameters rely on the convergence and information from the critic and if that is not trained well yet, the actor wont find a proper solution or even diverges. That is why the critic needs to learn faster (converge faster) than the actor. This explains also some oscillations that are visible in the learning process of the actor. If the critic finds itself in a new state-sequence (by exploration or by coincidence) and receives a higher reward/less penalty, then it will immediately update the value function approximation. The actor, then, will receive higher error measures and needs to adapt all over. Depending on the learning rate, this may take several trials again, or in some cases may even lead to divergence.
The optimal policy shapes after training can be seen in figure 6.3a and 6.3b. HDP shows a much more extreme control behavior, with mostly highly negative or positive elevator deflections. Only in a small region of q and $\Delta\theta$ the elevator deflection gets near zero. Additionally, it is noted that the control behavior is shifted. It would be expected to give a 0 elevator deflection at $q = 0\,\text{deg}$ and $\Delta\theta = 0\,\text{deg}$, but instead the actor suggests $\delta_{el} = -25\,\text{deg}$. The actor after DHP training shows a much more smooth surface with less extreme control deflections (±20 deg maximum). Also the control at state [$q = 0$, $\Delta\theta = 0$] is nearly zero, which indicates that the actor shape is correctly aligned.



(a) HDP response for the first 10 trials. The dotted line marks a new trial.

(b) DHP response for the first 10 trials. The dotted line marks a new trial.

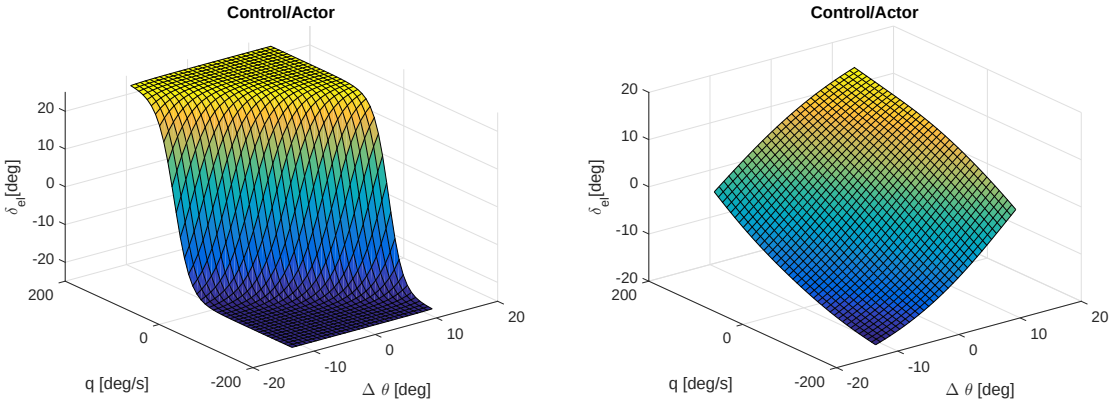
Figure 6.1: Offline learning history. Every $t = 20[s]$ the simulation is resetted. If the pitch angle exceeds 60 degrees, the trial is considered as a fail and the simulation stops.



(a) HDP

(b) DHP

Figure 6.2: Reward and error history for the complete experimental run. The error plots are on a log scale and the dotted line denotes the trial at which the reward approaches near 0.



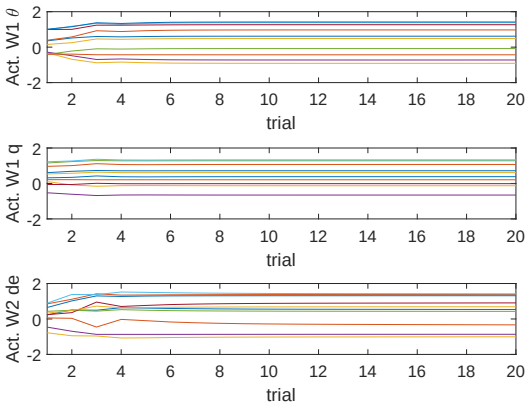
(a) HDP

(b) DHP

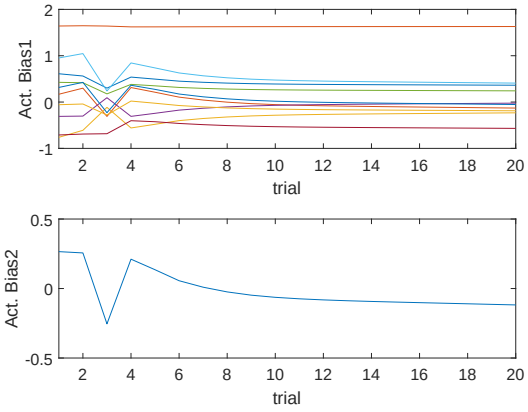
Figure 6.3: Actor input-output behavior after training

6.2. Neural network weight-updates

A good indicator for the learning progress is the evolution of the neural network. If it is a stable process, the weights of the actor and critic are converging to a constant and don't diverge or oscillate. Figure 6.4a to 6.7b show the update of the weights within the first 20 trials of the same experimental run as given in the previous section. Clearly, the weights and biases are heavily changing in the first four to five trials and then stay almost constant with small adjustments.

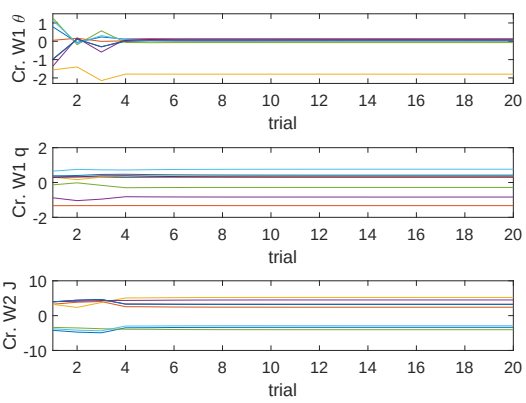


(a) HDP actor weights

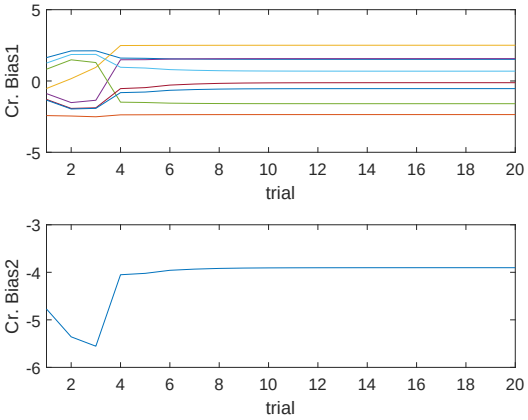


(b) HDP actor biases

Figure 6.4: Evolution of the actor weights and biases during the first 20 trials.

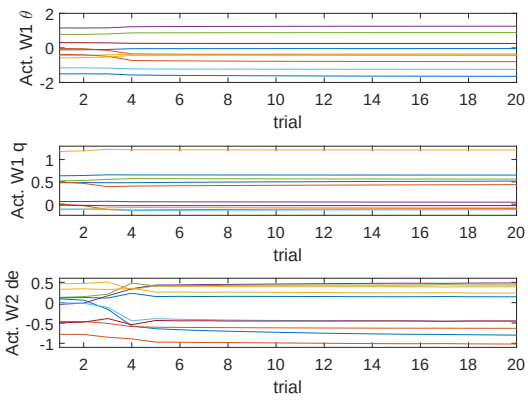


(a) HDP critic weights

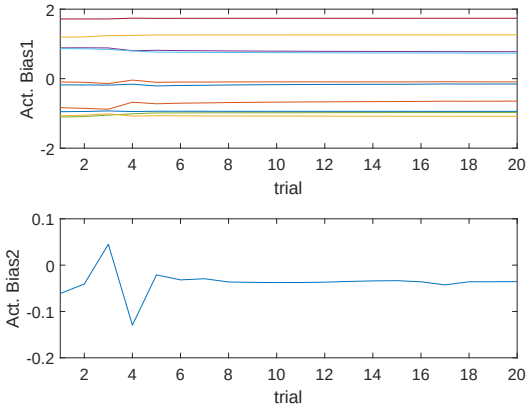


(b) HDP critic biases

Figure 6.5: Evolution of the critic weights and biases during the first 20 trials.

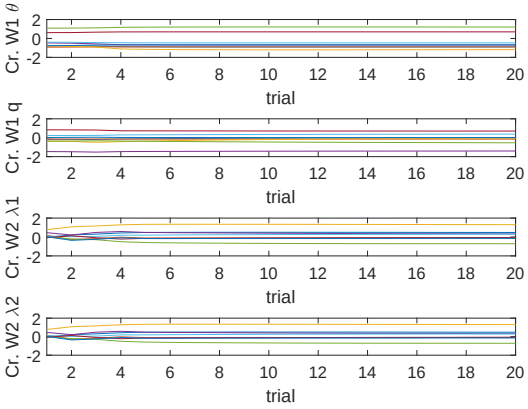


(a) DHP actor weights

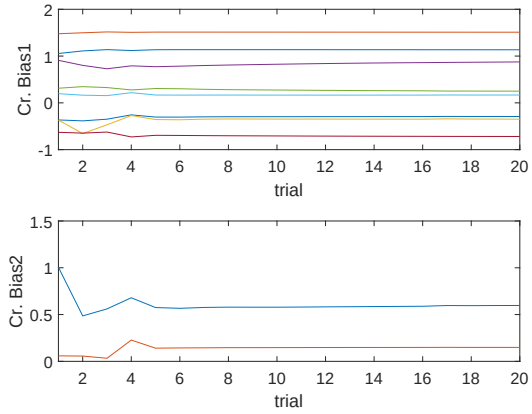


(b) DHP actor biases

Figure 6.6: Evolution of the actor weights and biases during the first 20 trials.



(a) DHP critic weights

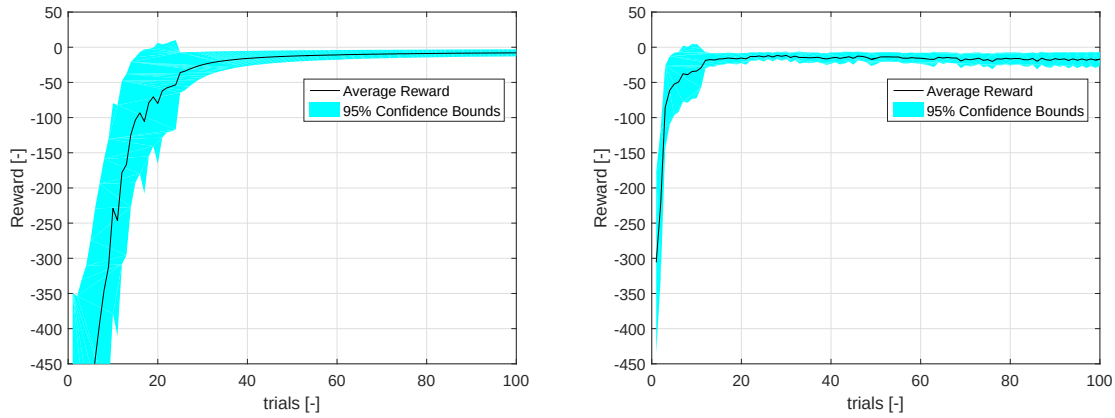


(b) DHP critic biases

Figure 6.7: Evolution of the critic weights and biases during the first 20 trials.

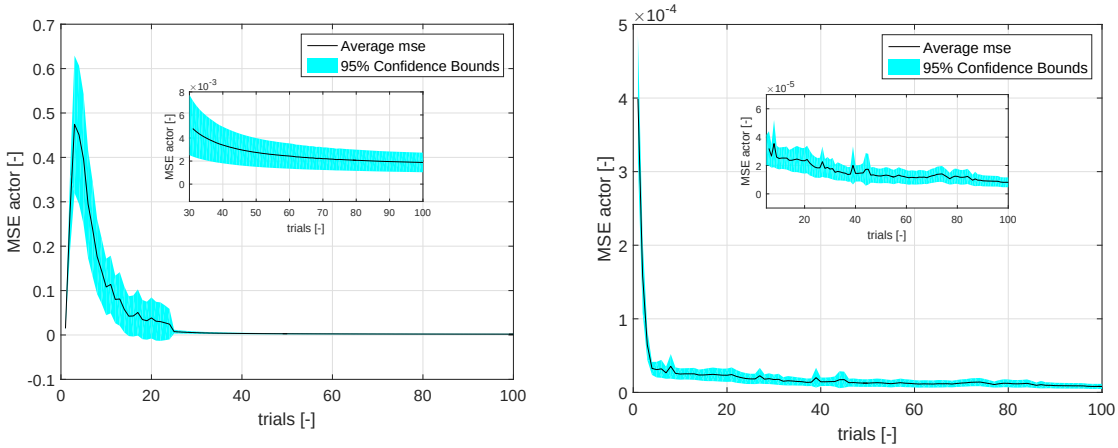
6.3. Sensitivity to initial weights

The results of the previous experimental run do not only depend on the chosen network structure and /hyperparameters, but also on the initial weights. Figure 6.8a to 6.10b, show the effect of different initial weights on the evolution of the learning process. The hyperparameters have been fixed as well as the network structure and only the weights are randomly initialized at the beginning of each experimental run. The magenta shaped area in the plots denotes the 95% confidence bounds of the successful experiments. Especially at the beginning of the experiments, the confidence bounds are wider, indicating a more coarse learning behavior and convergence rate. In the later stage, the bounds become significantly smaller. It is also noted that critic and actor error within HDP first start to get worse. This may be related to the higher amount of exploratory steps at the beginning of the experimental run. Rather then following the current (sub-optimal) behavior, the exploratory steps may push the agent to undesired states, leading to higher penalty and errors. In DHP, the learning process seems not at all to be prone to exploration. It consistently converges with each trial.



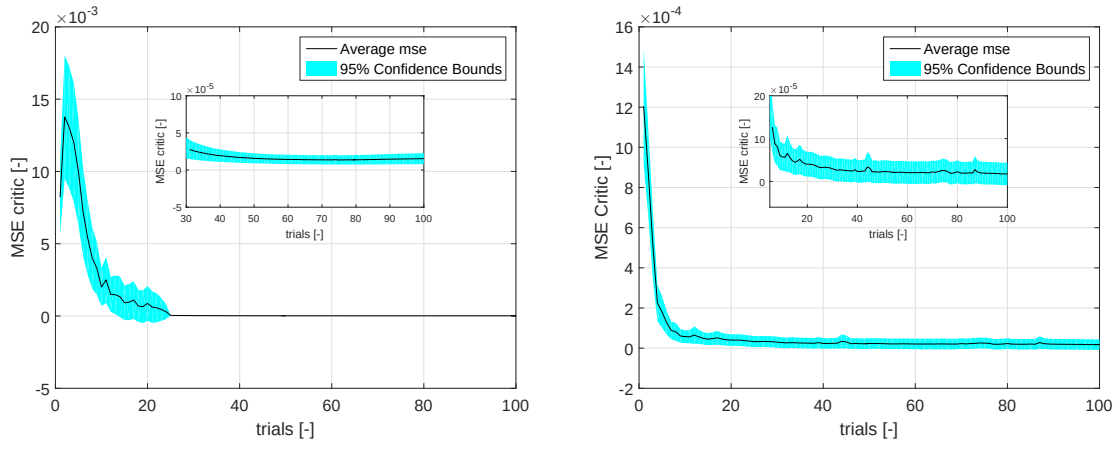
(a) HDP (b) DHP

Figure 6.8: Collected rewards over 100 experimental runs, with exploration. All failed runs are excluded



(a) HDP (b) DHP

Figure 6.9: MSE of the actor network over 100 experimental runs. All failed runs are excluded



(a) HDP (b) DHP

Figure 6.10: MSE of the critic neural network over 100 experimental runs. All failed runs are excluded

7

Hyperparameter Selection

This section gives a short overview of the selection process for finding the best hyperparameters.

7.1. Importance of tuning

There are many parameters in a artificial neural network that need to be defined such as the number of layers, weight regularization, layer size, learning rate, momentum factor etc. This can be a tedious task especially when multiple networks are linked and trained simultaneously. In HDP and DHP there are three artificial neural networks that depend on each other and on each others convergence. Therefore, also the tuning of the hyperparameters for each entity is interconnected. In order to improve convergence, the different networks can be trained offline, with a given data set, or a stable policy. As has been presented in the paper, a slight change in the hyperparameter settings can have a major impact on the convergence of the algorithm. It is therefore important to surely find the correct settings for the parameters, i.e. optimize the learning rate, momentum factor etc.

7.2. Optimization

Grid Search: The idea of grid search is simple and is considered to be the most widely used strategy for hyperparameter optimization next to manual search (LeCun et al. [32], Rumelhart and McClelland [53]). Consider a set of hyperparameters $\lambda = \{\mu, \eta, L...\}$ that effects the learning algorithm and each parameter has an associated set of values $\{K^{(1)}...K^{(s)}\}$. Then, the number of experiments is directly related to every possible combinations of those hyperparameters values, i.e. the number of trials is $H = \prod_{s=1}^{S} |K^{(s)}|$. Obviously, the more elements in λ or the more possible values K, the bigger the grid and the more elements H need to be tested. That is why grid search usually performs poorly in practice, as it requires a lot of time and test runs (Bergstra and Bengio [6]). In addition, the random initialized weights also have an effect on learning performance which means that each experiment with a certain parameter-set would need to be repeated a certain number of times with different weight-sets in order to determine the performance within some confidence bounds. This method has been applied in this thesis. It is a rather simple brute-force algorithm to find the right parameter settings. Almost 10000 different settings have been applied, each for 5 experimental runs in order to exclude the effect of random initial weights on the final performance. The selected search space can be seen in table 7.1. Note that the search space differs per algorithm. DHP seemed to perform better with slightly higher values for the learning rate and momentum factor. The critc learning rates have generally been higher as this entity needs to converge faster than the actor. If a certain setting has given good results, additional experiments have been conducted in the neighborhood of that setting. The search itself is computational expensive, requires a lot of simulation time and does not guarantee the best selected sets for the final runs. For future research it is recommended to use rather more advanced methods to find the right settings, such as Bayesian optimization.

Bayesian Optimization: In contrast to the previous two methods, Bayesian optimization is an automated process of finding the optimal parameters. It assumes the unknown function (in this case the maximum return) to be sampled from a Gaussian Process (GP). Snoek et al. [58] presents the framework of Bayesian optimization in machine learning and describes an algorithm that also takes into account the variable cost

(duration) of the learning process. Basically, after each experiment the hyperparameters are changed according to the Expected Improvement over the current best results or the Expected Improvement per second. It has been shown in Snoek et al. [58] that the results with Bayesian optimization outperforms current optimization benchmarks. That is why this tool is a considerable feature for this thesis. It has not yet been included for the results obtained in chapter 5, but it is planned to use it for the upcoming experiments.

Table 7.1: Search space of the hyperparameter settings for grid search

Variable	**Tested values HDP**	**Tested values DHP**
$n_{neurons}$	4 to 14 in steps of 1	4 to 14 in steps of 1
η_a	0.001 - 0.02 in steps of 0.001	0.01 - 0.2 in steps of 0.01
η_c	0.01 - 0.05 in steps of 0.01	[0.1 - 0.5 in steps of 0.1
μ_a	[0, 0.1, 0.2]	[0, 0.1, 0.2]
μ_c	[0, 0.1, 0.2]	[0, 0.1, 0.2]

8

Plant Model Performance

DHP and HDP are model-based RL methods, meaning that they require an on-board model of the plant. This chapter reviews the quality of the pre-trained plant model and also shows the online training behavior.

8.1. Offline training

The plant-model is an artificial neural network with 10 sigmoidal neurons in the hidden layer. It is trained offline at first, meaning that the parameters are updated in several epochs with previously gathered data. Before training, the data set is split into a training and validation sub-set. At the start of each epoch, the data samples of the training set are (re-)shuffled and divided into batches. Each batch is then individually presented to the neural network, for training. After an epoch, the mean squared error is determined with respect to the complete training and validation set. Figure 8.1 shows the MSE history throughout the training process. Clearly, the model error drops below 10^{-4} within less than 10 epochs. Since the neural network is fitted on the training set, the error continues to decline, while the validation MSE nearly gets constant. This also shows the significance of having an additional validation set, in order to detect if overfitting occurs.

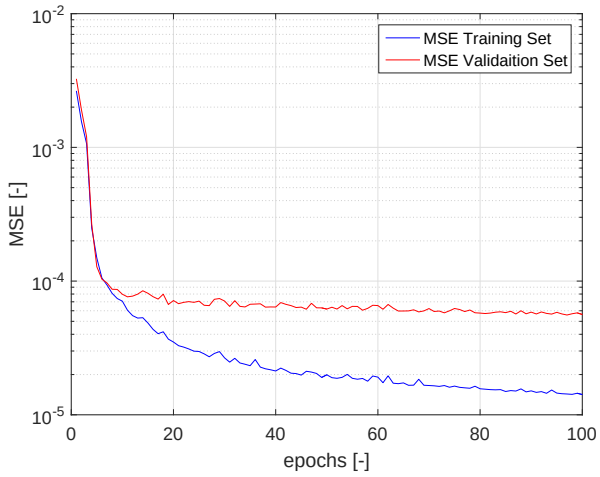


Figure 8.1: Plant model MSE over 100 epochs

8.2. Online adaption

In an online simulation, the plant model dynamics change instantaneously. For this particular example the change in center of gravity position is shown, to give the reader an idea what happens with the plant model during online adaptation. Figure 8.2 shows the learning process of the plant model at $t = 30[s]$, which is the time the change in plant dynamics occur. It can be seen that the plant model learns quickly, but with high oscillations at the beginning. After about 10 seconds it converges to a minimum of $1 \cdot 10^{-4}$ on average, with

small oscillations. The oscillations seem to coincide with the aircraft motion, which is following a sinusoidal reference with a 1 seconds period.

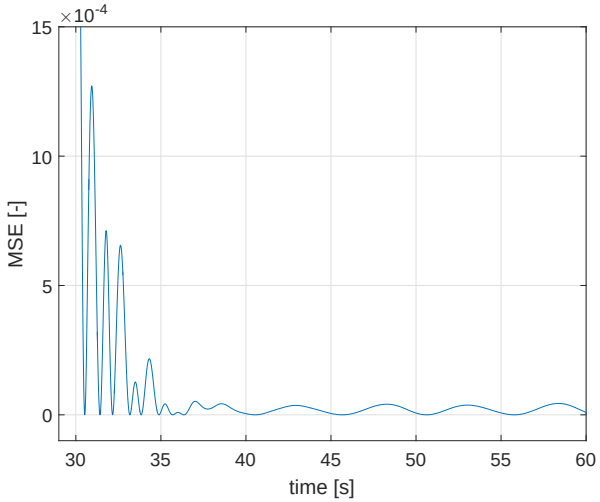


Figure 8.2: Online adaption of the plant model due to a shift in center of gravity position at $t = 30[s]$

9

Direct Online Learning Approach

This chapter shows an example of a direct online training with DHP. The agent has no knowledge in memory at the start of the simulation. Figure 9.1 shows the response of 100 seconds. It takes the algorithm about 18 seconds to converge to a near optimum and then refines until about 55 seconds, where the difference between the response and the reference gets nearly zero (with an off-set error of ± 1[deg]). The chances of convergence are relatively low of about 15% and depend on the initial weights. HDP on the other hand, diverged quickly in *all* the experimental run and was unable to learn an optimal control law (0% success).

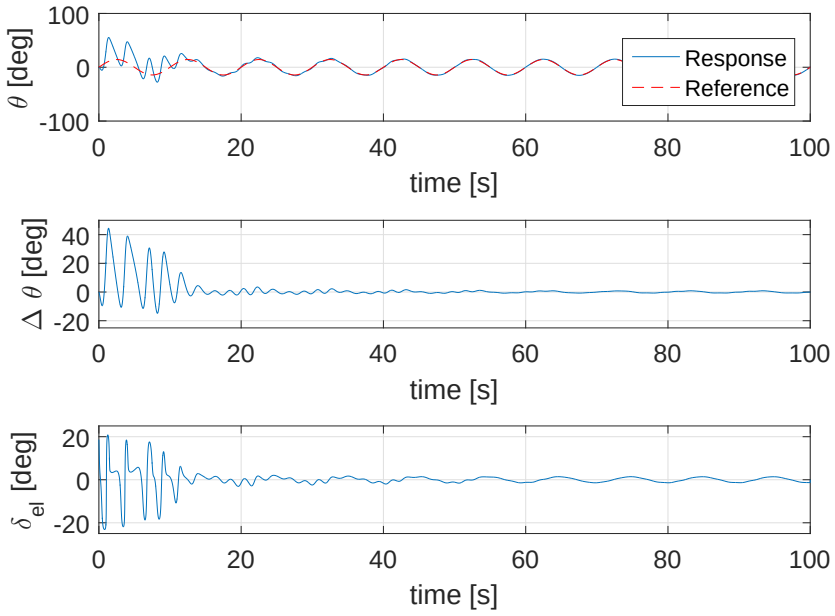


Figure 9.1: Direct online learning with DHP over 100 seconds and zero exploration.

IV

Closure

10 Conclusions

This thesis work has provided the results to determine the theoretical and practical differences of HDP and DHP for aircraft control. This chapter summarizes all results and conclusions that have been found throughout the study and gives some recommendations for future research.

10.1. Properties of VCCTEF

The idea of the VCCTEF system is to actively control and reshape flexible wings of an aircraft in any flight condition in order to achieve optimal aerodynamic performance, improve handling qualities and neutralize negative aeroelastic effects. VCCTEF is a design of Performance Adaptive Aeroelastic Wings that provides spanwise load tailoring with continuous trailing edge flaps, whereby adjacent flap sections are connected via an elastomer material. There are in total 16 flap sections spread along the complete wing trailing edge with 15 sections being attached to the outer wing and 1 section being attached to the inner wing. Each flap consists of three chordwise segments that can actively shape the chamber. As the VCCTEF is a multifunctional type of aerodynamic control surfaces, it provides multi-axes control options. In that perspective a multi-objective control architecture has been developed that intents to suppress aeroelastic modes, alleviates gust and maneuver loads, stability augmentation and drag optimization. However, there are multiple challenges that the controls are facing:

1. High amount of control surfaces that need to be steered. In total there are 48 surfaces *per wing* that can be controlled.

2. Model inaccuracies, meaning that the established model never matches 100% with the real-life case and designed controller need to

3. Certain level of dependency between each flap segment, which causes a problem as soon as malfunctions occur. Failure of a single section has an immediate effect on all other sections.

4. Linearization of the control systems may lead to additional inaccuracies

The controller needs therefore be robust, adaptive and accurate for large state-space systems. One potential approach for VCCTEF control is to use Adaptive-Critic Designs. ACD is a promising branch of reinforcement learning, where the controller learns itself the optimal control law (online or offline). It is an adaptive, non-linear control approach that does not require a complex model of the system a priori.

10.2. Comparison of HDP and DHP for aircraft control

The fundamental designs in ACD are heuristic dynamic programming and dual heuristic programming. HDP uses the approximation of the value function J(x) as the information for updating the policy while in DHP the derivative of the value function with respect to the states is directly used for the actor parameter-updates. Both algorithms have been applied to various applications in industry already and have shown promising results.
The contribution of this study is the application and comparison of HDP and DHP for aircraft control and

their potential for the VCCTEF design. During the preliminary analysis, both algorithms have been applied on an inverted pendulum task in over 20 experimental runs, each consisting of 300 trials. It is shown that DHP provided more reliable results with higher success rate of 95% then 75% for HDP. It also showed a faster convergence rate, as it required less than 100 trials to converge, while HDP needed 150 trials on average. The learning behavior for HDP was also much more coarse and the final control performance showed a slight offset error of ±5 degrees. Nevertheless, the rise-time was up to 20% faster when compared to DHP. The preliminary research was the first stage of analysis and the results show clearly the learning effect for the two algorithms, which verifies the developed code. Evidently, DHP seems to be a better choice for aircraft control with the VCCTEF design. This is due to the fact that it showed higher reliability, faster learning and better control performance.

During the main thesis work, HDP and DHP have been compared and evaluated on a 2D pitch-tracking task in a F-16 aircraft model. The experimental study involved an offline and online learning phase. The results have shown that during the initial offline phase, both controllers were able to learn the correct behavior and control the baseline model of the F-16 aircraft. DHP showed higher success ratios and converged to an optimal control-law in almost half the time as HDP. Also the tracking performance was more accurate, with an overall lower RMS error.

In the online learning phase, both controllers could handle most changes in the longitudinal aircraft dynamics even without adapting their parameters. This reveals their robustness as the baseline controller could cope with some changes in the dynamics, sometimes with better performance than without those changes. Especially HDP improved in the tracking performance for *all* the different modes. DHP seemed less robust. It was able to follow the pitch reference signal very well for changes in the C_{mq}, but was unable to handle the shift in the center of gravity position, resulting in an unstable behavior.

However, for simulations with parameter adaptation DHP showed a clear performance improvement, as it could adapt quickly to all the changes in plant dynamics, while HDP seemingly learned an unstable behavior. It can be concluded that DHP is a better method for aircraft control than HDP. It showed higher success rates, faster learning, decent robust behavior as well as quick and stable adaptation to changes in the plant dynamics. In general, the adaptability and self learning behavior makes it a promising solution for the VCCTEF design. However, a major drawback of these RL methods is that the learning process is not transparent and it is unknown what exact changes are made to the network parameters with every update cycle. This especially becomes a problem when the controller is learning a wrong policy, getting the system unstable as it was the case with HDP. Furthermore, the high sensitivity to the initial learning parameters makes it unclear whenever the best parameter combinations are chosen. Just a slight change in the settings has lead to a drastic change in performance in this study. It was also noted that the RL methods are well able to control complex systems with two states and one control variable, but as soon as the amount of states and actions increases the individual networks may have more trouble to converge. As in the online learning phase the overall convergence depends on the convergence of three separate entities (actor, plant-model and critic network) the probability of convergence of all entities to an optimum gets very low. Looking at the VCCTEF and its complexity with more than 48 control surfaces that can be actuated, a direct control of the effector using RL methods may not be feasible.

11

Recommendations

For future research it is recommended to apply HDP and DHP for aircraft control in a bigger state-action space. This shall reveal how the algorithms can perform with an increased complexity of the tasks. Next to that, it would be interesting to see how well HDP and DHP performs when the state information is provided by sensors that include sensor noise. Furthermore, brute force methods and grid search for defining the hyper parameters of the neural networks is a tedious task and does not guarantee the best settings. A way to improve this, is to use machine learning methods such as Bayesian optimization to search in the hyperparameter space and solve for the best settings. This makes also the comparison between the algorithms more fair as can be guaranteed that the networks perform with the best settings.
Another idea, would be to test other candidates from the design family such as GDHP, which supposedly combines the benefits of HDP and DHP.

V

Appendices

A

Artificial Neural Networks Function Approximators

Inspired by the functionality of a human brain Artificial Neural Networks (ANN) have gained a lot of attention in recent research especially in the field of machine learning and artificial intelligence. In general they can be seen as parametric functions that imitate the input-output behavior of any process. The typical feed forward neural network is therefore a mapping of parameter space to function space (Busoniu et al. [11]). Consider a dynamical process with the following property:

$$x(t+1) = f(x(t), u(t)) \tag{A.1}$$

This process can be represented by an ANN approximator such that

$$\hat{x}(t+1) = \tilde{f}(x(t), u(t); w) \approx x(t+1) \tag{A.2}$$

where $\tilde{f}(x(t), u(t); w)$ is the function approximator with parameters w, using the same inputs as the original system and approximating the original output. w are real numbers, also called the *network weights* and express the importance of respective inputs to the output. A feed forward neural network is a typical example of a parametric function that is non-linear in its parameters (Bertsekas and Tsitsiklis [7], Hassoun [21]). The form and number of parameters are typically defined before tuning and implementation. Looking at $\tilde{f}(x(t), u(t); w)$, the structure or such ANN can be described as follows (see figure A.1).

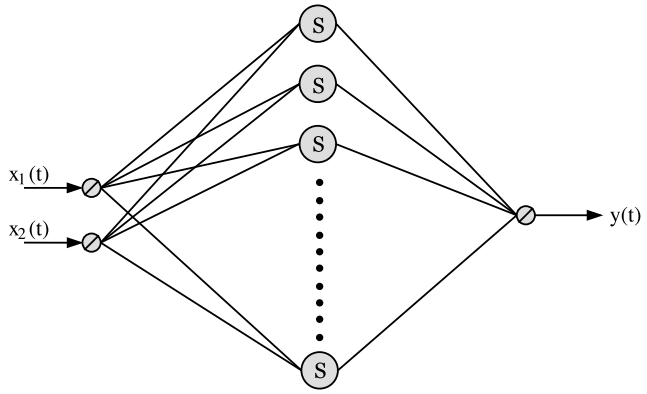


Figure A.1: Artificial neural network structure with 2 linear input neurons in the input layer, n neurons in the hidden layer and 1 linear output neuron. Each line indicates a connection between two neurons, while the connection contains a specific weight w

$\tilde{f}$ consists of multiple sections, the so called *layers* and each layer contains a certain number of *neurons*. The Neurons itself are linear or non-linear functions, called *activation-function* or *transfer-functions*. The number of input variables of the process determines the number of neurons in the *input layer*. Those are usually linear, meaning that they do not change the input value. The connections between each neuron

in one layer to a neuron in the subsequent layer is subjected to a weight w and denoted by lines in figure A.1. After the input layer follows a number of *hidden layers*, although one hidden layer is usually enough to capture most plant dynamics (Nielsen [48]). The number of hidden layers and number of neurons in each layer are design variables that need to be defined by the engineer.

A.1. Feed Forward Path

Processing the input and mapping it to an output signal is done by feed forward computation. Consider the network as shown in figure A.1. The states x_1 and x_2 pass the input neurons which are linear. In general, the output of neuron i in layer j is described by h_{ji}. So, for the input layer it gets:

$$h_{11} = x_1$$
$$h_{12} = x_2$$

h_{11} and h_{12} are then multiplied by n number of weights (denoted by black lines in figure A.1), where n is the total number of neurons in the next layer. The inputs to the neurons in the hidden layer is therefore a weighted sum of x_1 and x_2 with an additional bias b_{ji} and is denoted by z_{ji}:

$$z_{ji} = \sum_{k=1}^{m} x_k w_{ji\,k} + b_{ji} \tag{A.3}$$

Where k is the neuron from the previous layer with maximum m neurons. These are observed by the respective neuron transfer function in the hidden layer. Transfer-functions $\phi(z)$ within the neurons are typically *sigmoidal functions* such as hyperbolic tangent (see figure A.2). They compress the input z to an interval between 0 and 1 (or -1 and 1). The output therefore gets

$$h_2 = \phi(z) \tag{A.4}$$

which is a vector containing n elements, i.e. one for each neuron. The final outputs y_i are then determined by the weighted sum of h_2

$$\hat{y}_i = \sum_{k=1}^{m} h_{2i\,k} w_{2i\,k} + b_{2i} \tag{A.5}$$

If the output neuron is a sigmoidal function as well (necessary when saturation of outputs is desired), then the output gets

$$\hat{y}_i = \phi\left(\sum_{k=1}^{m} h_{2i\,k} w_{2i\,k} + b_{2i}\right) \tag{A.6}$$

This completes the feed forward path. Eventually, the network weights w need to be tuned such that the ANN output $\hat{y}$ approximates the desired output y. The update of the network parameters is done by backpropagation.

A.2. Backpropagation

Backpropagation is the most common way to update the parameters in a feed forward artificial neural network (Riedmiller and Braun [52]). The basic idea is to determine the influence of each weight in a neural network with respect to a specified error function E. This is done by means of the chain rule and finding the partial derivatives of E with respect to each weight (Rumelhart and McClelland [53]):

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_i}\frac{\partial z_i}{\partial w_{ji}} \tag{A.7}$$

$$\frac{\partial E}{\partial b_{ji}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_i}\frac{\partial z_i}{\partial b_{ji}} \tag{A.8}$$

where $w_{i\,j}$ and $b_i\,j$ are the network weights and biases respectively, for the connection between neuron j and i, y_i is the output of the i-th neuron and z_i the weighted sum of the inputs to that neuron. In most cases this error function is the squared difference between the output of the network y and the desired value y_D:

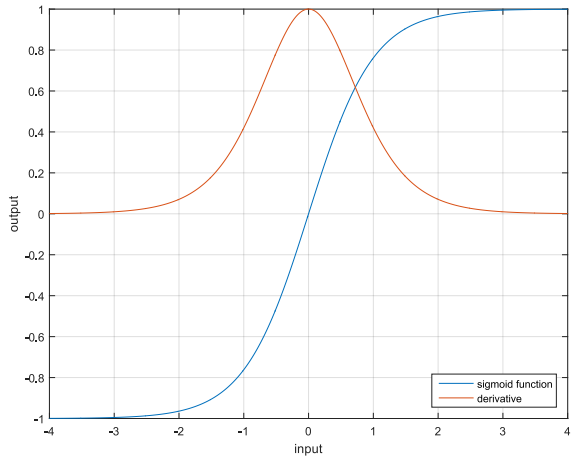


Figure A.2: Sigmoid transfer function and its derivative

$$E = \frac{1}{2}\sum_k e_k^2$$
$$e_k = y_k - y_D$$

with k being the number of outputs of the ANN. Having computed all partial derivatives from equation A.7 the minimum of E can be found by applying Stochastic Gradient Descent (SGD) rule:

$$w_{ji} \rightarrow w_{ij}(t+1) = w_{ji}(t) - \frac{\eta}{n}\eta\sum_n \frac{\partial E}{\partial w_{ji}} \tag{A.9}$$

and

$$b_{ji} \rightarrow b_{ij}(t+1) = b_{ji}(t) - \frac{\eta}{n}\eta\sum_n \frac{\partial E}{\partial b_{ji}} \tag{A.10}$$

With n being the size of the data-samples. Basically the weight is shifted in the direction of the negative gradient with step-size η, where η is the so called learning rate. The learning rate has a crucial role with respect to convergence and performance of the neural network. Choosing a value too high may lead to oscillations and performance will diverge in time rather than converge to an optimal solution. Some other settings lead the ANN to fall into a local optimum and if the learning rate is too low, the process takes too much time-steps to learn such that it may not even be apparent in the results. Tuning the parameters such as learning rate, number of nodes and hidden layers as well as the training and test sets can be very tedious and choices are seemingly arbitrary LeCun et al. [32]. With a high-dimensional, non-convex cost cost surface that contains multiple local minima, backpropagation may be very slow and convergence is not guaranteed. In recent years there have been some developments and extensions to improve convergence and find the optimal hyperparameters settings. A small review of those methods is given in the following sections A.3 and 7.2.

The backpropagation algorithm may be applied to each input-output pair seperately or in batches. In the example where a whole batch of data-pairs is presented to the network before the weights are adjusted is considered as batch-learning. SGD is a method that can be employed to update the weights online, i.e. each time a single data-pair {x, y} or mini-batch is presented. This method is preferred because 1) it converges much faster especially for data-sets with similar patterns, 2) results are often more accurate and 3) the online learning applicability.

Although feed forward ANN's have strong properties such as global approximation, non-linear approximation, black-box modeling and the possibility to represent high state-space dimension, usually linear parameterized approximators are preferred. This is due to the fact that those functions have better locality, meaning that it is possible to improve approximation power in a small region of the input space without effecting values outside of that region (Coulom [12]). Examples of such linear parameterized function approximators are

radial basis function networks (Bertsekas and Tsitsiklis [7]), tile coding (Sherstov and Stone [57], Watkins [67]), multilinear interpolation (Davis [14]) and Kuhn triangulation (Munos and Moore [38]).

A.3. Convergence Improvements

There have been multiple ways to improve the convergence and performance of artificial neural networks such as modified error functions, improved weight initialization, momentum co-efficient, adaptive learning rate and different transfer functions.

Momentum Factor: An early approach was to introduce a momentum co-efficient μ which modifies the current gradient descent update by adding a scaled gradient term from the previous step (Nielsen [48]):

$$v_{ij} \rightarrow v_{ij}(t+1) = \mu v_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}} \tag{A.11}$$

$$w_{ij} \rightarrow w_{ij}(t+1) = w_{ij}(t) + v_{ij}(t+1) \tag{A.12}$$

The temporary term v_{ij} is initialized to be zero and $\mu \in [0\ 1)$. Momentum-based gradient descent algorithms have the advantage to include the knowledge of the past gradient and shall accelerate convergence. However, in practice the problem of tuning is still present and not always leads to a more stable learning process (Riedmiller and Braun [52]). The choice of α and μ essentially relates to the training success and speed. The momentum factor is only applied to the weights and not to the biases.

Adaptive Learning Rate: Another way to improve weight-updates is to adapt the learning rate during the learning process (Darken and Moody [13], Schiffmann et al. [56]). A high learning rate is desirable to make bigger steps down the slope of a long, continual decreasing cost-function and reducing the rate leads to a good asymptotic behavior and prevents oscillations at the later stage. There are different strategies for learning rate adaption and some algorithms and their performance are presented in Schiffmann et al. [56]. The majority uses the information of the error function and its gradient to update the learning rate. The most common strategies can be described as follows (Plagianakos et al. [49]):

- The learning rate increases exponentially with every epoch that resulted in a reduced error and decreases significantly whenever the error increases
- Starting with a small α, the learning rate increases when the gradient of successive epochs points into the same direction, but is reduced as soon as the direction of the gradient fluctuates strongly
- Each weight is receiving an individual learning rate which increases whenever the ∂W has the same sign as the ∂W in the previous epoch. An example of such method is the Delta-Bar-Delta rule (Wu et al. [72])
- With each iteration the learning rate is calculated using a specified formula such as the "running average" or the "search and converge" (Darken and Moody [13])

Those algorithms may take into account the effect of the learning rate, but this effect can be disturbed by the behavior of the derivative $\frac{\partial E}{\partial w_{ij}}$ itself (Riedmiller and Braun [52]). An approach to overcome this issue is to apply Resilient Propagation (RPROB) where the weight update is determined only depending on the sign of the derivative and not on the value of $\frac{\partial E}{\partial w_{ij}}$ (Riedmiller and Braun [52]). it has also been demonstrated by various experiments that the algorithm promises a significant decrease in required learning epochs. In Schiffmann et al. [56] this algorithm is said even to be the fastest with very good performance and independent of the initial step-size settings.

Weight-initialization: There is one problem in particular with sigmoidal neural network structures that hampers the learning process. The input to each neuron is the weighted sum of input-signals and due to the properties of sigmoid functions, this signal is mapped to an interval between -1 and 1 (or 0 and 1). Whenever the neuron receives high or low values, the output saturates to the one of the values near the interval bounds. As explained in the previous section the derivative of the sigmoid function is an important addend in the backpropagation algorithm, but the derivative becomes nearly zero in those boundary regions. That is why different weight initialization of the neural network lead to different learning behavior & results. There are

multiple ideas to deal with this issue. One example is to add simply a small constant to the derivative such that it never gets zero (Fahlman [16]). However, Riedmiller and Braun [52] noted that this did not lead to a better and stable convergence property. Nielsen [48] described a better way to improve weight initialization by simply dividing all pseudo-random weight values by the square root of the number of input-nodes. The weights itself are initialized with normalized Gaussians.
In any case, this problem makes it obvious why the data-set needs to be normalized as high values for in- and outputs would lead to immediate saturation and slow learning.

Regularization: A neural network with a large number of free parameters can describe very complex systems, but a good fit with the available data may not necessarily conclude that the ANN is a good model (Nielsen [48]). It just means that it fits well with the given data, but it may not capture the genuine dynamics of the system itself. It is possible that the ANN is trained too well on the given set and thus fails to generalize to new situations and data points. In literature, this is referred to as overfitting and it is one of the reasons why the data-set is split into a training and test-set. So the performance of the ANN is evaluated based on a data-set that the ANN has not seen before. Another way to restrict overfitting is by regularization techniques. There are several methods such as dropout, artificially expanding the training-set, or L1 and L2 regularization (Nielsen [48]), while the later one is the most common method. L2 regularization, also called weight decay, adds a regularization term to the error function E:

$$E = \frac{1}{2n}\sum_{x}\mathbf{e}(x)^{T}\mathbf{e}(x) + \frac{L}{2n}\sum_{w}w^{2} \tag{A.13}$$

where L is known as the *regularization parameter*, n is the total amount of samples in the data-set and w are the network weights excluding the biases. With this addend, the weight update gets:

$$w_{ij} \rightarrow w_{ij}(t+1) = (1-\frac{\mu L}{n})\,w_{ij}(t) + {v_{ij}}' \tag{A.14}$$

and the term $(1-\frac{\eta L}{n})$ is considered to be the weight decay.

All aforementioned tools show promising results in application, which is why most of them are included within this research as well. Due to the circumstances of HDP and DHP algorithms, an ANN class was developed which includes SGD backpropagation, momentum factor, L2 regularization, normalized input-output data-pairs and improved weight initialization. The SGD property makes it possible to train the ANN's online. It is recommended to include the RPROB algorthim in combination with advanced adaptive learning rate capabilities in the next step such that convergence and performance are improved even further. All methods that are mentioned in this section may help to achieve better results, but the problem of tuning of each parameter is still present. Some methods for finding the best hyper-parameter settings are discussed in the next section.

Bibliography

[1] Airbus. For a better bottom line. URL http://www.a350xwb.com/cost-effectiveness.

[2] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inf. Theory*, 39(3):930–945, 1993.

[3] Andrew Barto, Richard S Sutton, and Charles W Anderson. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. In *Transactions on Systems, Man, and Cybernetics*, number 5, pages 834–846. Bäuerle, 1983.

[4] Richard Ernest Bellman. The Theory of Dynamic Programming. Princeton University Press, 1954.

[5] Richard Ernest Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[6] James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. ISSN 1532-4435.

[7] Dimitri P Bertsekas and John N Tsitsiklis. Neuro-Dynamic Programming: An Overview. *Proceedings of the 34th Conference on Decision & Control*, (December):1–34, 1996.

[8] Boeing. Development of Variable Camber Continuous Trailing Edge Flap System. *Report No 2010X0015*, 2012.

[9] Boeing. Development of Variable Camber Continuous Trailing Edge Flap System for B757 Configured with a More Flexible Wing. *NASA*, Report No.(September), 2014.

[10] N Borghese and M. Arbib. Generation of temporal sequences using local dynamic programming. *Generation of temporal sequences using local dynamic programming*, 1:39–54, 1995.

[11] Lucian Busoniu, Robert Babuska, Bart de Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Funciton Approximators*. 2010.

[12] Remi Coulom. Feedforward Neural Networks in Reinforcement Learning Applied to High-Dimensional Motor Control. *Algorithmic Learning Theory: 13th International Conference*, November(November), 2002.

[13] Christian Darken and John Moody. Note on Learning Rate Schedules for Stochastic Optimization. *Neural Information Processing Systems*, 1991.

[14] S Davis. Multidimensional triangulation and interpolation for reinforcement learning. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing systems 9*, pages 1005–1011. MIT Press, 1997.

[15] K Doya. Temporal difference learning in continuous time and space. *Advances in Neural Information Processing systems 8,*, (1994):1073–1079, 1996.

[16] Scott E. Fahlman. An Empirical Study of Learning Speed in Back Propagation Networks. *Technical Report CMU-CS-88-162*, 1988.

[17] Silvia Ferrari and Robert F. Stengel. Online Adaptive Critic Flight Control. *Journal of Guidance, Control, and Dynamics*, 27(5):777–786, 2004. ISSN 0731-5090. doi: 10.2514/1.12597.

[18] Yvonne Ferrier, Nhan T. Nguyen, and Eric Ting. Real-Time Adaptive Least-Squares Drag Minimization for Performance Adaptive Aeroelastic Wing. *34th AIAA Applied Aerodynamics Conference*, (June):1–34, 2016. doi: 10.2514/6.2016-3567. URL http://arc.aiaa.org/doi/10.2514/6.2016-3567.

[19] Ivo Grondman. *Online Model Learning Algorithms for Actor-Critic Control*. 2015. ISBN 9789461864321.

[20] Justin Hale. Boeing 787 from the Ground Up. *AERO Magazine*, pages 17–23, 2006. URL http://www.boeing.com/commercial/aeromagazine/articles/qtr{_}4{_}06/AERO{_}Q406.pdf.

[21] M Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Hassoun1995, 1995.

[22] R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.

[23] Corey Ippolito, Nhan T. Nguyen, Joseph Totah, Khanh Trinh, and Eric Ting. Initial Assessment of a Variable-Camber Continuous Trailing-Edge Flap System on a Rigid Wing for Drag Reduction in Subsonic Cruise. *AIAA Infotech @ Aerospace*, AIAA2013-5(August), 2013.

[24] Thomas Jordan, William Langford, Christine M Belcastro, J.V. John Foster, Gautam Shah, Gregory Howland, and Reggie Kidd. Development of a Dynamically Scaled Generic Transport Model Testbed for Flight Research Experiments. *AUVSI's Unmanned Systems North America 2004 Symposium and Exhibition*, page 16, 2004.

[25] E Kampen, Q P Chu, and J A Mulder. ONLINE ADAPTIVE CRITIC FLIGHT CONTROL USING APPROXIMATED PLANT DYNAMICS. (August):13–16, 2006.

[26] Upender K Kaul and Nhan T Nguyen. Drag Optimization Study of Variable Camber Continuous Trailing Edge Flap (VCCTEF) Using OVERFLOW. *32nd AIAA Applied Aerodynamics Conference*, (June):1–20, 2014. doi: 10.2514/6.2014-2444. URL http://arc.aiaa.org/doi/abs/10.2514/6.2014-2444.

[27] Upender K Kaul and Nhan T Nguyen. Drag Optimization Study of Variable Camber Continuous Trailing Edge Flap (VCCTEF) Using OVERFLOW. *32nd AIAA Applied Aerodynamics Conference , 16-20 June 2014, Atlanta, GA*, (June):1–20, 2014. doi: 10.2514/6.2014-2444. URL http://arc.aiaa.org/doi/abs/10.2514/6.2014-2444.

[28] Vijay R Konda and John N Tsitsiklis. On Actor-Critic Algorithms. *Control Optim*, 42(4):1143–1166, 2003. ISSN 0363-0129. doi: 10.1137/S0363012901385691.

[29] S. Lebofsky, E. Ting, and N. Nguyen. Aeroelastic Modeling and Drag Optimization of Aircraft Wing with Variable Camber Continuous Trailing Edge Flap. *32nd AIAA Applied Aerodynamics Conference*, (June):1–34, 2014. doi: 10.2514/6.2014-2443.

[30] Sonia Lebofsky, Stinger Ghaffarian Technologies, and Moffett Field. Multidisciplinary Drag Optimization of Reduced Stiffness Flexible Wing Aircraft With Variable Camber Continuous Trailing Edge Flap. 2015. doi: doi:10.2514/6.2015-1408.

[31] Sonia Lebofsky, Stinger Ghaffarian Technologies, and Moffett Field. Multidisciplinary Drag Optimization of Reduced Stiffness Flexible Wing Aircraft With Variable Camber Continuous Trailing Edge Flap. (January):1–27, 2015. doi: doi:10.2514/6.2015-1408.

[32] Yann A. LeCun, Lion Bottou, Genevieve B. Orr, and Klaus Robert Miller. Efficient backprop. *Neural Networks: Tricks of trade*, pages 9–48, 2012. ISSN 03029743. doi: 10.1007/978-3-642-35289-8-3.

[33] George G. Lendaris. A retrospective on Adaptive Dynamic Programming for control. *Proceedings of the International Joint Conference on Neural Networks*, pages 1750–1757, 2009. doi: 10.1109/IJCNN.2009.5178716.

[34] Frank Lewis and Draguna Vrabie. Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3):32–50, 2009. ISSN 1531-636X. doi: 10.1109/MCAS.2009.933854.

[35] Derong Liu, Hossein Javaherian, Olesia Kovalenko, and Ting Huang. Adaptive critic learning techniques for engine torque and air-fuel ratio control. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):988–993, 2008. ISSN 10834419. doi: 10.1109/TSMCB.2008.922019.

[36] Chao Lu, Jennie Si, and Xiaorong Xie. Direct Heuristic Dynamic Programming for Damping Oscillations in a Large Power System. 38(4):1008–1013, 2008.

[37] W. T. Miller, R. S. Sutton, and P. J. Werbos. *Neural Networks for Control. Cambridge*. MIT Press, Cambridge, 1990.

[38] R. Munos and A. Moore. Variable resolution discretization in optimal control. In *Machine Learning*, pages 291–323. 2002.

[39] N Nguyen and J Urnes Sr. Aeroelastic modeling of elastically shaped aircraft concept via wing shaping control for drag reduction. *AIAA Atmospheric Flight Mechanics Conference*, (650), 2012. doi: 10.2514/6.2012-4642. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84880811460{&}partnerID=40{&}md5=daceaacb5516dfd2e4ba860abfcff123.

[40] N. Nguyen, U. Kaul, S. Lebofsky, E. Ting, D. Chaparro, and J. Urnes. Development of Variable Camber Continuous Trailing Edge Flap for Performance Adaptive Aeroelastic Wing. 2015. doi: 10.4271/2015-01-2565.

[41] Nhan Nguyen, Moffett Field, Ilhan Tuzcu, Tansel Yucelen, and Anthony Calise. Longitudinal Dynamics and Adaptive Control Application for an Aeroelastic Generic Transport Model. pages 1–22.

[42] Nhan Nguyen, Upender Kaul, Sonia Lebofsky, Eric Ting, Daniel Chaparro, J. Urnes, Moffett Field, Sonia Lebofsky, Eric Ting, Daniel Chaparro, and St Louis. Development of Variable Camber Continuous Trailing Edge Flap for Performance Adaptive Aeroelastic Wing. 2015. doi: 10.4271/2015-01-2565.

[43] Nhan T. Nguyen. Elastically Shaped Future Air Vehicle Concept. In *NASA Innovation Fund 2010 Project*, 2010.

[44] Nhan T. Nguyen and Greg Hornby. Trajectory Optimization Using Adjoint Method and Chebyshev Polynomial Approximation for Minimizing Fuel Consumption During Climb. *AIAA Infotech@Aerospace (I@A) Conference*, pages 1–12, 2013. doi: 10.2514/6.2013-5142. URL http://arc.aiaa.org/doi/abs/10.2514/6.2013-5142.

[45] Nhan T Nguyen and Ezra A Tal. A Multi-Objective Flight Control Approach for Performance Adaptive Aeroelastic Wing. *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, (650), 2015. doi: doi:10.2514/6.2015-1843. URL http://dx.doi.org/10.2514/6.2015-1843.

[46] Nhan T Nguyen and Ezra A Tal. A Multi-Objective Flight Control Approach for Performance Adaptive Aeroelastic Wing. *56th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, (650), 2015. doi: doi:10.2514/6.2015-1843. URL http://dx.doi.org/10.2514/6.2015-1843.

[47] Nhan T. Nguyen, Sonia Lebofsky, Eric Ting, Daniel Chaparro, Upender Kaul, and James Urnes. Development of Variable Camber Continuous Trailing Edge Flap for Performance Adaptive Aeroelastic Wing. In *SAE Technical Paper*, volume 1, 2010.

[48] Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[49] V. P. Plagianakos, G. D. Magoulas, and M. N. Vrahatis. Learning Rate Adaptation in Stochastic Gradient Descent. *Advances in Convex Analysis and Global Optimization*, 54, 2001.

[50] Nathan Precup, Marat Mor, and Eli Livne. Design, construction, and tests of an aeroelastic wind tunnel model of a variable camber continuous trailing edge flap (VCCTEF) concept wing. *32nd AIAA Applied Aerodynamics Conference*, (June), 2014. doi: 10.2514/6.2014-2442.

[51] Danil V Prokhorov and Donald C Wunsch. Adaptive Critic Designs - Neural Networks, IEEE Transactions on. 8(5):997–1007, 1997.

[52] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *IEEE International Conference on Neural Networks - Conference Proceedings*, (January):586–591, 1993. ISSN 10987576. doi: 10.1109/ICNN.1993.298623.

[53] D. E. Rumelhart and J. McClelland. *Parallel Dis- tributed Processing*. 1986.

[54] Enns Russell and Jennie Si. Apache Helicopter Stabilization Using Neural Dynamic Programming. *Guidance, Control and Dynamics*, 25(1):19–25, 2002.

[55] Richard S. Russell. Non-Linear F-16 Aircraft Model. *University of Minnesota*, 2003.

[56] W Schiffmann, M Joost, and R Werner. Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrons. *Physics*, (September):1–36, 1994. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.137.4028.

[57] A Sherstov and P Stone. Function approximation via tile coding: Automating parameter choice. In *6th international symposium on Abstraction Reformulation and Approximation*, pages 194–205, 2005.

[58] Jasper Snoek, Hugo Larochelle, and Rp Adams. Practical Bayesian Optimization of Machine Learning Algorithms. *Nips*, pages 1–9, 2012.

[59] S. S. Stahara. Operational manual for two-dimensional transonic code TSFOIL. 1978.

[60] R S Sutton and A G Barto. *Reinforcement learning : an introduction*. MIT, 1998. ISBN 9780262193986. doi: 10.1109/TNN.1998.712192.

[61] A Tal. *Coupled Flight Dynamic and Unsteady Aeroservoelastic Modeling and Control of a Flexible Wing Aircraft*. PhD thesis, 2015.

[62] Brian Taylor. Performance Adaptive Aeroelastic Wing. In *the University of Minnesota Digital Conservancy*. University of Minnesota, 2015.

[63] G. Tesauro and B. Janssens. Scaling Relationships in Back-Propagation Learning. *Complex Systems*, 2: 39–44, 1988.

[64] Eric Ting, Stinger Ghaffarian Technologies, and Moffett Field. Static Aeroelastic and Longitudinal Trim Model of Flexible Wing Aircraft Using Finite-Element Vortex-Lattice Coupled Solution. (January):1–29, 2014. doi: 10.2514/6.2014-0837.

[65] James Urnes and Nhan Nguyen. A Mission Adaptive Variable Camber Flap Control System to Optimize High Lift and Cruise Lift to Drag Ratios of Future N+3 Transport Aircraft. *AIAA-2013-214, Aerospace Sciences Meetings*, (650):1–24, 2013. doi: 10.2514/6.2013-214. URL http://dx.doi.org/10.2514/6.2013-214.

[66] Ganesh K. Venayagamoorthy, Ronald G. Harley, and Donald C. Wunsch. Comparison of heuristic dynamic programming and dual heuristic programming adaptive critics for neurocontrol of a turbogenerator. *IEEE Transactions on Neural Networks*, 13(3):764–773, 2002. ISSN 10459227. doi: 10.1109/TNN.2002.1000146.

[67] C. J. C. J. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Oxford, UK, 1989.

[68] Paul Werbos. Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence. *General Systems*, (XXII):25–38, 1977.

[69] Paul Werbos, Richard S Sutton, and W. T. Miller. A menu of designs for reinforcement learning over time. *Neural Networks for Control*, MIT Press:67–95, 1990.

[70] Paul J Werbos. Approximate Dynamic Programming for Real-Time Control And Neural Modeling. In D A White and D A Sofge, editors, *Handbook of Intelligent Control: Neural, Fuzzy and Adaptive Approaches*, page ch 13, New York, 1992. Van Nostrand.

[71] Ian H Witten. An Adaptive Optimal Controller for Discrete-Time Markov Environments. In *Information and Control 34*, pages 286–295, 1977.

[72] Jun Wu, Xin Xu, Chuanqiang Lian, and Yan Huang. Adaptive Dual Heuristic Programming Based on Delta-Bar-Delta Learning Rule. pages 11–20, 2011.