Deep Reinforcement Learning for Aircraft Landing

A study on the use of Deep Reinforcement Learning techniques for automatic control of aircraft landing

MSc Thesis

T. H. A. van de Laar

Delft University of Technology

TU Delft

Deep Reinforcement Learning for Aircraft Landing

A study on the use of Deep Reinforcement Learning techniques for automatic control of aircraft landing

by

T. H. A. van de Laar

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday July 7, 2023 at 09:30 AM.

Student number:	4558871	
Project duration:	June, 2022 – June, 2023	
Thesis committee:	Dr. ir. E. van Kampen,	TU Delft, supervisor
	Dr. M. D. Pavel,	TU Delft, chair
	Dr. M. Lourenço Baptista,	TU Delft

Cover: ”An airplane flying in the sky” by Jorgen Hendriksen, Unsplash license, https://unsplash.com/pt-br/fotografias/dP4tgu5GGYw

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

TU Delft

Preface

The field of Reinforcement Learning is undoubtedly a fascinating one, which has seen tremendous increased interest in the recent years, and hence by default has also seen incredible advancements. Along the other Machine Learning paradigms, RL is heading towards solving problems with ever increasing complexity in numerous fields, such as engineering, robotics, medicine and economy. This research uses the latest technologies in the field of artificial intelligence and reinforcement learning to contribute to autonomous aviation by making aircraft more safe and fault-tolerant. In an effort to aid the growth of the fields of reinforcement learning and aviation, the code utilised in this thesis is made publicly available such to facilitate the repeatability of the results herein presented. [1]

This thesis marks the end of my studies at the Delft University of Technology, a place that sparked my curiosity and opened fascinating doors I thought would never be possible. It has been an extraordinary experience, and I would like to express my gratitude to those without whom this journey would not have been possible. First, I would like to thank Dr. ir. Erik-Jan van Kampen for his support and guidance throughout this research, and for introducing and giving me the opportunity to work on this fascinating field that is reinforcement learning. I would like to thank my friends which whom I have shared many highs and lows and spent long hours working together. Tank you for your support, all the discussions we have had and the beers we have shared. I would also like to thank my family, both in the Netherlands and in Brazil for all the support you have given me. Finally, I would like to express my utmost gratitude to my parents for all the effort you have made, for the time you have spent, the advices you have given and most of all for the opportunities you have offered me, without you more than anyone this would not have been possible. Muito obrigado!

T. H. A. van de Laar
Delft, June 2023

[1] https://github.com/tvandelaar/RLALS-SAC

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ABS	Adaptive Back-stepping
ACKTR	Actor-Critic with Kronicker-Factored Trust Region
ADC	Adaptive Critic Design
AD	Action-Dependent
ADDHP	Action-Dependent Dual Heuristic Programming.
ADGDHP	Action-Dependent Global Dual Heuristic Programming
ADHDP	Action-Dependent Heuristic Dynamic Programming
ADP	Approximate Dynamic Programming
AFCS	Aircraft Flight COntrol System
AGL	Above Ground Level
AI	Artificial Intelligence
ANN	Artificial Neural Network
ALS	Automatic Landing System
BLEU	Blind Landing Experimental Unit
BPN	Back-propagation network
CPN	Counter-Progpagtion Network
D4PG	Distributed Distributional Deep Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
DH	Decision Height
DHP	Dual Heuristic Programming
DME	Distance Measuring Equipment
DNN	Deep Neural Network
DP	Dynamic Programming
DPG	Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EASA	European union Aviation Safety Agency
FAA	Federal Aviation Administration
GDHP	Global Dual Heuristic Programming
GPI	Generalised Policy Iteration
GPS	Global Positioning System
GNC	Guidance, Navigation and Control
GNSS	Global Navigation Satellite System
G/S	Glide-Slope
HARV	High Alpha Research Vehicle
HDP	Heuristic Dynamic Programming
iADP	Incremental Approximate Dynamic Programming
IBPN	Improved Back-Propagation Network
ICAO	International Civil Aviation Organisation
IDHP	Incremental Dual Heuristic Programming
IGDHP	Incremental Global Dual Heuristic Programming

Abbreviation	Definition
IHDP	Incremental Heuristic Dynamic Programming
ILS	Instrument Landing System
JAA	Joint Aviation Authorities
LOC	Localiser
MADDPG	Multi-Agent Deep Deterministic Policy Gradient
MC	Monte-Carlo
MDP	Markov Decision Process
MLFN	Multilayer Functional-Link Network
ML	Machine Learning
NDI	Nonlinear Dynamic Inversion
NN	Neural Network
PI	Policy Iteration
PID	Proportional Integral Derivative
PPO	Proximal Policy Optimisation
RA	Radio Altimeter
RBFN	Radial Basis Functional Network
RL	Reiforcement Learning
RVR	Runway Visual Range
SAC	Soft Actor-Critic
SMC	Sliding Mode Control
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimisation
UAV	Unmanned Aerial Vehicle
VI	Value Iteration

1

Introduction

1.1. Background & Motivation

According to [8], the final approach and landing phases of flight approximately represent 4% of the general flight time of an aircraft, however, it accounted for 54% of aircraft fatal accidents that happened between 2011 and 2020. Another study points out that in-flight Loss of Control (LOC-I) and Hard Landing accounted for 80% of the total fatalities in aviation in 2019 [31]. In view of these numbers, it is no surprise that approach and landing represent areas of concern in current aviation and it is critical that they are addressed and improvements are searched for. The thesis herein contained investigates a method to remedy accidents caused by the influence of unpredicted *disturbances* and *actuator failures* during landing by studying the implementation of an *Automatic Landing System (ALS)*. In the search for making the landing procedure more safe and secure, it is proposed to utilise *Reinforcement Learning (RL)* based controllers to explore novel, more robust and more adaptive control methods.

Automatic Landing Systems were initially designed as a tool that allowed landings in poor weather conditions, where visibility was the major imposing factor preventing the pilots to land the aircraft. Blind Landings were a hot topic in the 1940s and 1950s, having simultaneously, organisations in England, France and the USA developing systems to perform such manoeuvres. In England, the developments took place through the Blind Landing Experimental Unit (BLEU) which started its operations in 1945 and developed early versions of ALSs for both civil and military operations [14]. The main motivation behind the undertaking of such a project was the frequent occurrences of low visibility weather conditions in the North-West region of Europe, especially in the winter months. Nowadays, further development on ALS is a promising candidate for solving or minimising, the issues encountered with unexpected disturbances and in-flight loss of control during the most critical phases of flight.

Since the development of the early automatic landing systems in the 1940s and 50s, such systems have been further enhanced and nowadays are present in numerous aircraft, for example, the Lockheed L1011s and the Boeing B747s, performing landings routinely [33]. The current use of ALS however, is restricted to relatively calm weather conditions, the main reason for that is the fact that the vast majority of the planes rely upon classical control methods, e.i. PID controllers, or modification of it. As it is characteristic of such systems, they strongly (i) rely on mathematical models, (ii) often require lengthy tuning and are (iii) suited for linear systems, hence have limited performance on complex and variable systems, such as those encountered during the landing procedure. However, new control methods have been developed in the recent year in the hope of making flight control more *adaptable* and *robust*. Such methods include the use of Nonlinear Dynamic Inversion (NDI) [12] used in the F-35 aircraft, (Adaptive) Backstepping (ABS) [24], Sliding Mode Control (SMC) [55], Optimal control such as H_{inf} and H_2 [60], to mention a few.

The methods previously mentioned require knowledge of the dynamics of the aircraft and its surroundings, which becomes a problem when the dynamics change, for example, due to internal and/or external disturbances. This thesis proposes to investigate the feasibility of a control system based on *Reinforcement Learning* in an attempt to resolve/minimise the performance gaps encountered in automatic landing. In the realm of control engineering, RL is a framework that allows a controller to shape itself based on its interaction with the environment. These adaptations occur based on posi-

tive or negative feedback the controller receives based on its performance. Hence, learning occurs through a sophisticated process of trial and error, allowing the knowledge of the system dynamics to be completely unknown to the controller.

Historically, Reinforcement Learning is the conjunction of two areas of study that for long seemed to have no connection [63]. RL has bio-inspired status thanks to the first area of the two, which concerns *learning by trial-and-error* rooted to the study of animal psychology, also linked to the *"law-of-effect"* of Edward Thorndike [69]. Its *learning* process also sets Reinforcement Learning as an additional paradigm of Machine Learning (ML), within Artificial Intelligence (AI), alongside Supervised and Unsupervised Learning. The other area of study is more directly related to the topic in this thesis and has its roots in finding solutions for the *optimal control* problem through the use of value functions and Dynamic Programming (DP). Most of the work on RL is focused on, but not limited to, problems that can be described by the *Markovian Decision Process (MDP)* formalism [74]; A decision-making model where sequential decisions are made based on previous decisions and current state. According to [11], there are two main advantages of MDPs, and hence RL, that sets them apart: (i) their generality that allows for handling nonlinear and stochastic systems, and (ii) *model-independence*, meaning that they do not necessarily need a model of the environment's dynamics or even an expression for the reward function.

Additionally, RL-based controllers may *learn* how to best perform a task whilst performing it, when applied in such a manner RL is said to be *online* and allows for highly *adaptive* control. On the other hand, reinforcement learning can use the prevalent and effective technique called Artificial Neural Networks (ANNs), a powerful approximation tool that employs a flexible structure for generalisation. Hence, RL and ANNs can be used together to achieve *robustness* against nonlinear and highly dynamic systems, this requires adequate prior, or *offline*, training of the agent. These theoretical characteristics of reinforcement learning make it a promising candidate for finding solutions for the challenges currently faced in aerospace.

Moreover, RL features a wide variety of fields, from engineering to economics, and from AI to neuroscience, and it has been given considerable attention in recent years, resulting in rapid progress. In the field of Computer Science, for example, RL has accomplished incredible feats with DeepMind's AlphaGo [46] defeating a professional GO player, and Deep Q-Networks (DQN) [48] surpassing human performance in several Atari games. In the field of engineering reinforcement learning is extensively applied in the research for the development of autonomous vehicles [20] [78]. In the field of aerospace engineering, even though RL applications to aerospace guidance, navigation and control (GNC) are a novelty, it has been successfully applied to a multitude of Flight Control applications. For example, [62] implemented an RL technique called Incremental Global Dual Heuristic Programming (IGDHP), based on Approximate Dynamic Programming (ADP) to generate a self-learning adaptive flight controller. Dual Heuristic Programming (DHP) has been applied to a 6-DoF business jet flight controller by [19]. RL algorithms such as Deep Deterministic Policy Gradient (DDPG) and Proximity Policy Optimisation (PPO) techniques have been applied to flight control of Unmanned Aerial Vehicles (UAVs) in [70] and [9], respectively. Lastly, DDPG has also been applied to fixed-wing aircraft automatic landing in [67], where it shows that DDPG may perform better at landing than other control techniques based on supervised learning.

To conclude, reinforcement learning is a growing field that has promising applications in aerospace. Even though it is a novelty within aerospace GNC, it is a fast-growing field and it has been successfully applied in multiple systems. RL is not a perfect solution and it contains limitations, however, there are numerous benefits to its application in Automatic Landing Systems (ALS).

1.2. Objectives & Research Questions

The research herein proposed aims to remedy the LOC-I problems during the final stages of flight by means of advancing the field of autonomous systems, this is proposed to be accomplished through a study on the development and implementation of Reinforcement Learning techniques applied to the control of Automatic Landing Systems. Hence the goal of the thesis is framed as:

> "The main research goal is to contribute to the development of Automatic Landing Systems that are capable of performing under unforeseen circumstances with enhanced robustness and control accuracy, to increase repeatability and safety by means of exploring the use of different Reinforcement Learning frameworks applied to control techniques"

From the research goal, the main research questions and sub-questions are derived as:
The main research question is defined as:

> '**How can Reinforcement Learning techniques be applied to Automatic Landing Systems to improve robustness and control accuracy?**'

The sub-questions are defined such that they aid in answering the main question and give direction to the research performed.

- RQ1: What are the state-of-the-art methods that are currently employed/proposed in Automatic Landing Systems and what are their problems/challenges?
  - RQ1.1 What are the control methods utilised?
  - RQ1.2 Under what circumstances do these systems become faulty/present issues?
  - RQ1.3 What are the requirements for such systems? (airfield, weather conditions, onboard equipment...)
- RQ2: What characteristics of Reinforcement Learning are suitable to mitigating ALS's issues (from RQ1)?
  - RQ2.1 What are the current state-of-the-art RL algorithms?
  - RQ2.2 Which RL framework is suitable for resolving ALSs issues?
  - RQ2.3 How well does the proposed algorithm perform when applied to simple flight control systems?
- RQ3: How can the proposed RL framework be implemented in an ALS control design to increase robustness and control accuracy?
  - RQ3.1 What are the characteristics of the landing environment and how can it be modelled?
  - RQ3.2 What are the architectural characteristics of the RL controller that allow optimal implementation?
- RQ4: How does the proposed Automatic Landing System method perform compared to current methods?
  - RQ4.1 How does the simulated landing performance of the proposed method compare to those of classic methods? (with respect to relevant touchdown variables)
  - RQ4.2 How much training is required for the proposed algorithm to achieve the landing conditions requirements? And how consistent are the results?
  - RQ4.3 How robust is the proposed system to environment changes?

1.3. Outline

This document is divided into 7 parts, which not only have the objective of informing the reader about the results of the performed research but also guiding them through the methodologies employed and the rationale behind the choices taken in the development process. **Chapter 2** presents the final product of this research, a scientific article written to showcase its main outcomes and final results. **Chapter 3** contains a literature review in which the topic of (automatic) landing systems by describing landing phases, instruments and regulations, previously developed automatic landing systems and the challenges that they currently face. The information presented in the chapter is gathered in the last section

in order to answer research question **Q1**. The topic of reinforcement learning is also described in **Chapter 3** where the basic elements of the RL problem, common approaches, design choices, as well as examples of how RL has been applied in Flight control and the current challenges of the field, are presented. Additionally, it dives into more specific details of reinforcement learning through an analysis of possible algorithms, partially answering research question **Q2**. **Chapter 4** presents a series of tests in a well-known bench-marking environment, namely Gym's *Lunar Lander* in order to evaluate the performance of different RL algorithms. Once more, the information presented in the chapter is gathered in the last section in order to completely answer research question **Q2**. Additional results that were not included in the scientific article are presented in **chapter 5**, which contains a brief study on the effects of the use of action smoothening parameters, additional failure cases and the performance of alternative controllers. **Chapter 6** contains the processes used for verification and validation of both, the environment model and algorithm itself. Finally, **Chapter 7** gives a brief summary of the information presented throughout the report answering the research questions and presents recommendations for future research on the topic.

2

Article

Soft-Actor Critic Deep Reinforcement Learning for Automatic Control of Aircraft Landing

T. H. A. van de Laar*
Delft University of Technology, P.O. Box 5058, 2600GB Delft, The Netherlands

Aircraft accidents mainly occur in the final phases of flight and the majority of fatalities in recent years can be rooted to in-flight loss of control (LOC-I) and hard landings. Therefore, the need for better fault-tolerant control systems for these phases of flight is evident. This research explores the use of reinforcement learning techniques for automatic control of aircraft landing. The proposed cascaded Automatic Landing System (ALS) structure consists of a combination of PID controllers on the outer loop and Deep Reinforcement Learning controllers in the inner loop for more robust guidance of the control surfaces. The state-of-the-art offline trained model-free Soft Actor-Critic algorithm is used to train pitch and roll attitude controllers. Simulation tests are made under sensor and actuator failures to evaluate the performance of the proposed ALS regarding robustness and fault-tolerance compared to classical methods.

I. Introduction

The final approach and landing phases of flight approximately represent 4% of the general flight time of an aircraft, however, it accounted for 54% of the aircraft fatal accidents that happened between 2011 and 2020 [1]. Another study points out that in-flight Loss of Control (LOC-I) and Hard Landing accounted for 80% of the total fatalities in aviation in 2019 [2]. Hence, the need for better fault-tolerant control systems in aviation is evident, especially during the final phases of flight.

Automatic Landing Systems (ALS) were first developed in the 1940s and 1950s in order to aid pilots to land in the poor weather conditions of northwestern Europe, which often did not allow complete visibility of the landing path. In England the Blind Landing Experimental Unit (BLEU) started its operations in 1945 and developed early versions of ALSs for both civil and military operations [3]. Nowadays all large commercial jets are equipped with ALS, also known as autoland, allowing landings to happen in conditions where it would be dangerous or completely impossible otherwise. The use current use of ALS however, is restricted to a relatively limited flight envelope, the main reason for that is the fact that the vast majority of the planes rely upon classical control methods, e.i. PID controllers, or modification of it.

Current flight control systems, such as those in ALS, rely on classical control theory, which is widely used in the field of control due to its straightforward implementation and reliability. To operate in non-linear systems such methods require the use of gain scheduling to switch between different linear controllers that have been optimized for each point in the flight envelope. The use of classical control theory methods has two drawbacks, the first being that gain scheduling can be a tedious and gruelling process, and more importantly, the second is that these controllers are linear and strongly rely on mathematical models. The latter implies that the generated controller is only accurate if the plant dynamics model is accurate, which poses great challenges for complex coupled dynamics systems.

Since the 1950s new control methods have been developed in the hope of making flight control more adaptive and robust. Nonlinear Dynamic Inversion (NDI), for instance, has been used in the F-35 aircraft to enhance performance qualities [4] [5]. An (Adaptive) Backstepping (ABS) controller was designed using a non-linear aircraft model to hold a constant angle of attack during the aircraft's final descent [6]. A Neural-aided Sliding Mode Control (SMC) ALS method was proposed and tested under actuator failure and severe wind conditions [7]. Moreover, a wide range of ALS methods using optimal control have been proposed, such as [8] with a H_2 based controller that is fault-tolerant and takes into account wind disturbances, [9] with robust H_∞ based controllers, and notable the works of [10], where H_2 control is used to determine the optimal trajectory, taking into account the predefined and the actual trajectories of the aircraft, subsequently, H_∞ is used to minimise the effects of disturbances in the performance output.

However, the methods previously mentioned still require, at one point or another, the knowledge of the dynamics of the aircraft and its surroundings, which becomes a problem when the dynamics change, for example, due to internal and/or external disturbances. Therefore, better adaptive and robust control systems must be developed in order to solve

*M.Sc. Student, Faculty of Aerospace Engineering, Department of Control & Operations, Section Control & Simulation, Delft University ofTechnology.

the current lack of fault tolerance and robustness to constantly changing plant dynamics encountered in aviation and other fields.

Reinforcement Learning (RL) is a paradigm of Machine Learning (ML) that allows computers to learn from their own interaction with the environment, this concept applied to control engineering results in a framework that allows a controller to shape itself based on its interaction with the environment, regardless of its own concept of it. These adaptations occur based on positive or negative feedback the controller receives from its performance. Hence, learning occurs through a sophisticated process of trial and error, allowing the knowledge of the system dynamics to be completely unknown to the controller.

Recent advancements and the increase in popularity of Deep Learning (DL) and Artificial Neural Networks (ANNs) have made their way to RL fomenting the field and creating a new branch referred to as Deep Reinforcement Learning (DRL). The use of ANNs allows RL to efficiently transition to continuous state and action problems, which was not easily done previously due to the tabular nature of early RL methods. In 2015 DeepMind presented a Deep Q-Network (DQN) algorithm that was capable of achieving human-level performance in a number of classic Atari games [11]. In 2016 the company followed up its previous DRL achievements with AlphaGo, a DRL-trained computer program that was capable of defeating a professional human Go-player [12]. Many of the earliest RL methods employed an **Actor-Critic** agent structure, and to this day they still remain the state-of-the-art in RL control techniques due to their intuitive task division where *learning* is solely reserved for the critic and *controlling* for the actor. The most common and effective actor-critic methods are the Adaptive Critic Designs (ADC), from the field of Approximate Dynamic Programming (ADP), and what is commonly referred to as Actor-Critic Deep Reinforcement Learning (AC-DRL), the latter although a general and unclear term will be used in this paper to differentiate from ADP and because it is the most commonplace in literature.

ADP methods are model-based, hence are limited to the quality of the model used. However, recent research on a branch of the field called Incremental ADP (iADP) implemented the use of online incremental model-learning techniques to eliminate model dependence and provide solid adaptive control [13]. Incremental Heuristic Dynamic Programming (IHDP) and Incremental Dual Heuristic Programming (IDHP) are two of these methods that have been applied to Nonlinear Adaptive Flight control in [14] and [15], respectively. The advantage of iADP methods is their high sample efficiency when compared to other methods, which allow for full-online learning without the need for the initial offline training phase. Flight control research on a high-fidelity aircraft model showed that body-rate control with coupled dynamics can be done using IDHP [16], and late also altitude, and attitude control [17]. However, validation and high-fidelity simulations still have to be performed before real-world flight tests are possible.

Whilst ADP methods tend to provide adaptive control, DRL methods generally provide robust control. After the appearance of the DQN by DeepMind control application of DRL methods have been extended with multiple algorithms, such as Deep Deterministic Policy Gradient (DDPG), Twi-Delayed DDPG (TD3), Soft Actor-Critic (SAC), Proximal Policy Optimisation (PPO), Trust Region PO (TRPO), Advantage Actor-Critic (A2C), Asynchronous A2C (A3C), and many others. Flight control research on a model of the Cessna Citation 500 aircraft showed that attitude and altitude can be controlled using SAC [18], and its fault tolerance was evaluated under a series of failure cases for an altitude tracking task. Additionally, a hybrid controller utilizing SAC and IDHP was also developed and tested on the same aircraft model [19].

The contribution of this paper is an Automatic Landing System that employs DRL techniques. Achieved through an investigation into the use of modern model-free DRL flight controllers for general aviation. The research focuses on the most troublesome phases of flight, namely, the final approach and landing, which suffer the most from LOC-I. More specifically, this paper presents a study on the use of DRL flight controllers in an Automatic Landing System (ALS), implemented in a high-fidelity simulation model of a Cessna Citation 500.

The theoretical basis for this paper is described in section II through a description of the (automatic) landing and reinforcement learning problems, followed by a description of the SAC algorithm, the DRL algorithm explored in this research. Section III presents an ALS base controller structure and the design of a PID and RL controllers for it. Sections IV and V present the discussion of the results and the conclusions, respectively.

II. Fundamentals

This section briefly introduces the (automatic) landing problem and the general reinforcement learning problem and provides a more detailed description of the RL algorithm explored in this research, the Soft Actor-Critic.

A. (Automatic) Landing problem

Generally, the landing procedure is considered to be composed of four phases: *Glide slope descent* (or final approach), *Flare*, *Touchdown*, *roll-out* [20], all of which are illustrated in figure 1. The glideslope path ensures that the aircraft lands at a stable descent rate by following a descent path usually with a $-3°$ angle with the ground, although the angle might slightly change depending on the airport and its surroundings. The flare occurs at a height of around 10 to 5 $[m]$ above ground level and it is meant to change the aircraft's descent attitude to a landing attitude by pitching up and reducing the sink rate of the aircraft, setting it up for a soft touchdown, improving passenger comfort and minimising landing gear impact. As mentioned previously, the majority of fatal accidents in the past decades have happened in these two flight phases, hence these are the most critical ones. The touchdown and roll-out phases are the aircraft's gentle touch onto the landing surface and the deceleration to a controlled speed for taxiing.

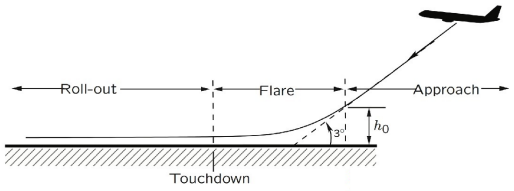


Fig. 1 Aircraft Landing phases, retrieved from [21]

As with many procedures in aerospace, landing requires the use of multiple instruments and aiding systems. There are two main ground-based systems that are required to perform landing: 1. the *Instrument Landing System (ILS)*, which is divided into *localiser - LOC*, *glideslope - G/S* and possibly *beacon markers*, and 2. *Radio Altimeters* [22]. The onboard equipment is limited to suitable receivers and computers that are able to process the information provided by the ground-based equipment.

The ILS is a navigation-aid system that provides short-range lateral and vertical guidance for aircraft during the landing procedure, and it consists of a localiser antenna (LOC), a glide-slope antenna (G/S), and a beacon marker. The LOC is placed at the end of the runway and provides *lateral guidance* by informing the aircraft about its position relative to the runway's centre line. The G/S is placed at the start of the runway and provides *longitudinal guidance* by informing the aircraft about its position relative to the glideslope path. The LOC and G/S beam lobes that provide the transmitted signals are represented in figure 2. The remaining component of the ILS is the beacon markers, which provide the aircraft with information about its distance to the runway, however, these have been slowly substituted by increasing accuracy of Distance Measuring Systems (DMEs) and Global Navigation Satellite Systems (GNSS). Finally, the RA is required to provide accurate information about its altitude above ground level. It is mainly used once the aircraft has reached altitudes lower than 600 $[m]$, and it provides more accurate information than other altitude systems, such accuracy is mainly required such that the aircraft is able to initiate the flare at the correct height.

The ALS was initially developed in the 1950s to aid pilots to land in difficult weather conditions such as those encountered in North-western Europe, which generally result in low visibility. The idea of the Automatic Landing System stems from the concept that the ILS provides a tracking signal that can be followed by an autopilot controller. Already in 1959, it was stated that the ALS will not only be capable of landing the aircraft in situations where a human pilot would not be able to, but it will be able to do so with more accuracy and consistency [3].

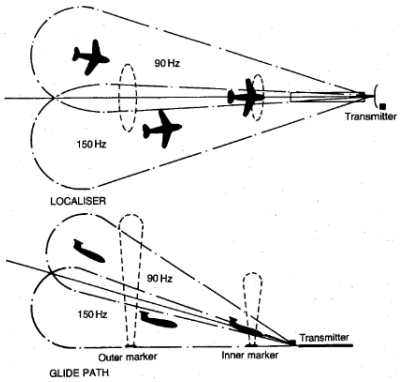


Fig. 2 ILS Working principle, retrieved from [20].

B. Reinforcement Learning problem

The majority of Reinforcement Learning problems can be mathematically described by the Markov Decision Process (MDP), where at a given *state* s, an *agent*, or controller, interacts with the *environment*, or plant, through *actions* a, and in turn, receives information about the environment's *new state* s' and a *reward* r characterizing the quality of the action taken. This formulation assumes the process to be Markovian, that is to have the Markov Property, meaning that future states solely depend on the current state, not past ones, as in equation 1. That also means that the current state-action pair contains all the required information to predict the next state. Ultimately, the agent uses the information acquired from its actions and their responses on the environment to create a series of laws that allow it to maximise the amount of reward gathered over time G_t, equation 2, also known as the policy.

In RL the state-action function illustrates how beneficial it is to be in a given state s_t, in other words, how much reward r can be expected in the future, when an action a_t is taken and the policy is followed thereafter. A discount factor γ is used to determine how relevant immediate rewards are with respect to future rewards. The state-of-the-art algorithm structure employs an entity that is solely responsible for learning the policy to be followed, and a separate entity that critiques the learned policy, which in turn is adapted accordingly, these entities are referred to as Actor and Critic, respectively.

$$s_{t+1} = f(s_t, a_t) \tag{1}$$

$$G_t = \sum_{k=0}^{N} \gamma^k r_{t+k+1} \tag{2}$$

The agent's policy can be deterministic or stochastic, meaning that the agent will always take the exact intended action or that there is a probability of each action in each state. This distinction is interesting and both of these methods are used in this research. Off-policy DRL algorithms often times make use of randomly-sampled *action noise*, to enhance the learning process of the agent [23], that is because it forces the agent to take actions that it would otherwise have not taken, expanding the learning scope and it may happen that the forced action might actually be more beneficial. Hence, stochastic actions are used during training, for better learning, and deterministic actions are used during testing, such that the policy is evaluated in its raw form.

C. Soft Actor-Critic

The Soft Actor-Critic (SAC) is an off-policy DRL algorithm that introduces stochastic policies and double Q-learning to the already successful DDPG algorithm, initially presented in [24]. The former is introduced to enhance exploration, which is especially important for offline-trained algorithms, such as AC-DRL algorithms. The latter is a characteristic shared with TD3 and Double DQN (DDQN), and it is introduced to remedy the value function over estimation issues encountered in previous algorithms (DDPG and DQN). Another key feature of the SAC algorithm is the use of entropy regularisation to balance the maximisation of the returns over time and the randomness of in the policy, in other words, its entropy.

1. Actor - Policy

The role of the actor in the actor-critic agent structure is to learn a policy, in the case of the SAC, this policy is stochastic and it is represented by π_θ, where the addition of the θ term refers to the parameters of a Deep Neural Network (DNN). The actor is modeled by an m-dimensional DNN that outputs two values: the standard deviation σ_θ and the mean μ_θ. These values are then used to normally sample a value from a Gaussian distribution, this value is then squashed by a Tanh function and then scaled according to the action boundaries of the environment it is in. This normal sampling of the unscaled action is what characterizes the policy as stochastic and ensures adequate exploration, provided the hyperparameters have been appropriately tuned. This stochasticity of the sampling means that the policy objective is non-differentiable, which is a problem for policy gradient calculations. This issue is solved with the "parametrization trick" that is proposed by the authors of the SAC paper, where the action is sampled using the mean, the standard deviation, and a Gaussian noise vector ϵ as shown in equation 3.

$$a_t = f_\theta(\epsilon_t, s_t) = \pi_\theta(s_t) + \sigma_\theta * (s_t)\epsilon_t \tag{3}$$

The Loss function of the actor is presented in equation 4 and is not only dependent on the states, sampled from batch B, and actor's actions, but also on the Value function Q, retrieved from the Critic, and the entropy coefficient η, both of which are touched upon on the next sections. The log probabilities term is also derived from the normal distribution and is used in both, the actor and the critic.

$$L_\pi = E_{s_t \sim B, a_t \sim \pi}[min_{i=1,2}(Q'_{w'_i}(s_t, a_t)) - \eta * log\pi_\theta(a_t|s_t)] \tag{4}$$

A downside of the stochasticity of the SAC algorithm is the fact that the selected actions can be oscillatory and noisy, complicating the identification of a correlation between one action and the next, and consequently a state and the next. In the past, this was remedied by letting the actor control the increment to the current action, instead of the entire action itself, making therefore the correlation between them clear and reducing the variation between subsequent actions. This situation, however, is not ideal and can hinder the learning process. Fortunately, recent developments in smoothing of action policies lead to the development of the CAPS parameters, or Conditioning for Action Policy Smoothness (CAPS) [25]. This innovative method employs two regularisation terms: Temporal and Spatial. In short, the former is based on the distance between the current and previous action and it penalises the actor by taking subsequent actions that are far from each other. The latter, computes the distance between the action taken and the previous action taken in a similar state, which is normally sampled from a nearby state $\bar{s}$. The CAPS are implemented as additional terms in the Actor's loss function, allowing the smoothness to occur inside the algorithm and be completely independent of the environment and any changes that may happen to it. The Temporal regularisation loss term is shown in equation 5 and the Spatial loss in equation 6, in this case, the overall loss function can be seen in equation 7 where the relevance of the CAPS parameters is balanced through the scaling parameters λ_T and λ_S.

$$L_T = ||\pi(s_t) - \pi(s_{t+1})||_2 \tag{5}$$

$$L_S = ||\pi(s) - \pi(\bar{s})||_2 \tag{6}$$

$$L_\pi^{CAPS} = L_\pi + L_T\lambda_T + L_S\lambda_S \tag{7}$$

2. Critic - Value

The role of the critic in the actor-critic agent structure is to criticize the policy the actor has learned, in the case of the SAC, this is done through the estimation of the state-action function Q_k, where the addition of the k term refers to the parameters of a Deep Neural Network (DNN). SAC also features a double Q-network, where two Q-functions are estimated, but only the one with the lowest value is used to estimate the value and gradient updates, this is done to avoid the overestimation of the Q-function, a common issue in previous algorithms. The use of twin state-action value functions speeds up training in complex tasks and increases learning stability. The target network is updated with the value-network with an exponentially weighted moving average as a soft update mechanism, in such a way that the former is a delayed version of the latter, this ensures a smoother direction of the updates.

$$L_Q(k_i, B) = E_{(s_t, a_t, s_{t+1}) \sim B, a_t \sim \pi_\theta}[(Q_{k_i}(s_t, a_t) - \bar{r}(s_t, a_t) + \gamma(min_{i=1,2}(Q_{\bar{k}_i}(s_{t+1}, a_{t+1}) - \eta * log\pi_\theta(a_t|s_t))))^2] \tag{8}$$

The critic loss function is shown in equation 8 and it includes a modified version of the Bellman equation, that includes the entropy term η. Additionally, SAC is an off-policy algorithm, therefore, the state-value function is estimated

with no regard to the current policy. This allows the algorithm to store transition samples (s_t, a_t, r_t, s_{t+1}) in a memory buffer, which in turn can be used to sample mini-batches of transition samples and reused for more optimal sample efficiency.

3. Entropy & Temperature

Entropy is a term that measures the randomness of a variable in its probability distribution. High entropy means high disorder/high uncertainty, low entropy means order/certainty. In addition to maximising the expected return, the SAC algorithm also maximises the entropy, which means that not only the algorithm learns optimal policies but also gives priority to the most stochastic policies that still achieve high returns. By choosing the most stochastic policy, the algorithm creates an inherent exploration mechanism that reduces the risks of local optima convergence and ultimately leads to better performance and a more robust controller. The trade-off between entropy and future rewards is done through the temperature parameter η in equations 4 and 8. The entropy function is shown in equation 9.

$$H(\pi(\cdot|s_t)) = E_{a_t, a_{t+1}\ \pi_\theta}(-log\pi(a'|s_t)) \tag{9}$$

SAC was shown to be unstable with respect to the temperature parameter, this is because the optimal trade-off between entropy and rewards is not constant throughout the training process, which is what is assumed when the temperature parameter is kept constant. Exploration should be much lower at the end of training than at the beginning. Therefore an automatic control of said parameter was later proposed by the authors of the original SAC paper in [26]. The loss function presented in equation 10 is used to optimise the temperature parameter. The $\bar{H}$ is a constant entropy target that is set to be the same as the negative of the action space size.

$$L_\eta = E_{s_t \sim B, a_t \sim \pi_\theta}(-\eta log\pi(a_t|s_t) - \bar{H}) \tag{10}$$

4. Overview

Figure 3 presents an overview of the SAC algorithm, containing the Actor, the Critics, the entropy, and the environment, as explained in the previous sections.

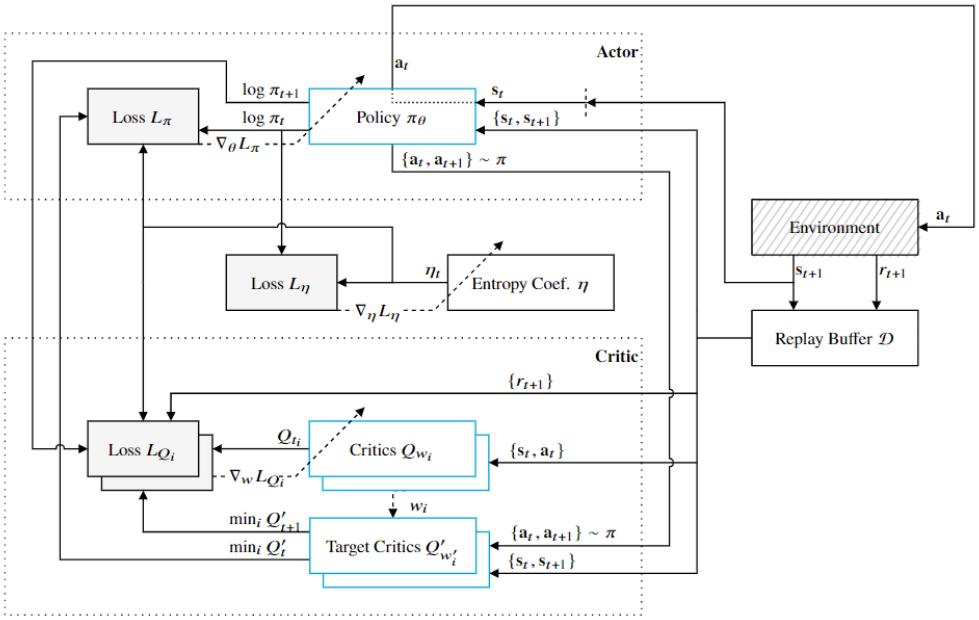


Fig. 3 Schematics of the SAC framework, retrieved from [19]

III. Methodology

This section introduces the environment to be used for this research, a high-fidelity simulation model of the Cessna Citation 500, and the ALS controller structure, including a series of variations that incorporate PID and RL controllers. Finally, the training configuration and strategy for pitch and roll attitude RL controllers are also described in this section.

A. Environment - Cessna Citation Model

In reinforcement learning the terms agent, action and environment are analogous to the terms controller, control input, and plant in control engineering. Additionally, for the agent, the environment is considered to be anything that is not the agent itself, that includes the aircraft actuators, ILS, and other controllers.

The environment used in this research is a high-fidelity non-linear model of the Cessna Citation 500. The simulation model was built using the Delft University Aircraft Simulation Model and Analysis Tool (DASMAT). Model validation was done in [27] where system validation was performed on flight test data recorded on a Cessna Citation 550, an aircraft of a similar size and engine power. The specific aircraft is also known as the PH-LAB and it is owned by the TU Delft Faculty of Aerospace Engineering in conjunction with the Netherlands Aerospace Centre (NLR) for research purposes. The use of this aircraft model gives continuity to previous research and allows for information and techniques to be drawn from previous papers [17] [18] [19] [28] that studied the use of SAC and other RL-based fault-tolerant controllers the same simulation model.

The Citation model consists of a nonlinear 6-DOF model that combines rotation and translation equations of motion for a rigid-body aircraft that has been trimmed to a specific flight condition. The environment is simulated with a refresh rate of $100Hz$ and outputs a complete set of 12 states shown in equation 11, where p, q, r are the roll pitch and yaw rates, V_{TAS}, α, β, are the true airspeed, angle of attack and the sideslip angle, θ, ϕ, ψ, are the pitch, roll and yaw angles, and H, X, Y are the distances in Z, X, and Y directions respectively. Furthermore, the velocity of the aircraft is controlled by an auto-throttle and the trim tab and flap deflections are kept at zero, only the elevator , ailerons and rudder deflections, δe, δa, δr respectively, can be used to control the aircraft, yielding the set of actions in equation 12.

$$\bar{x} = [p, q, r, V_{TAS}, \alpha, \beta, \theta, \phi, \psi, H, X, Y]^T \tag{11}$$

$$\bar{x} = [\delta e, \delta a, \delta r]^T \tag{12}$$

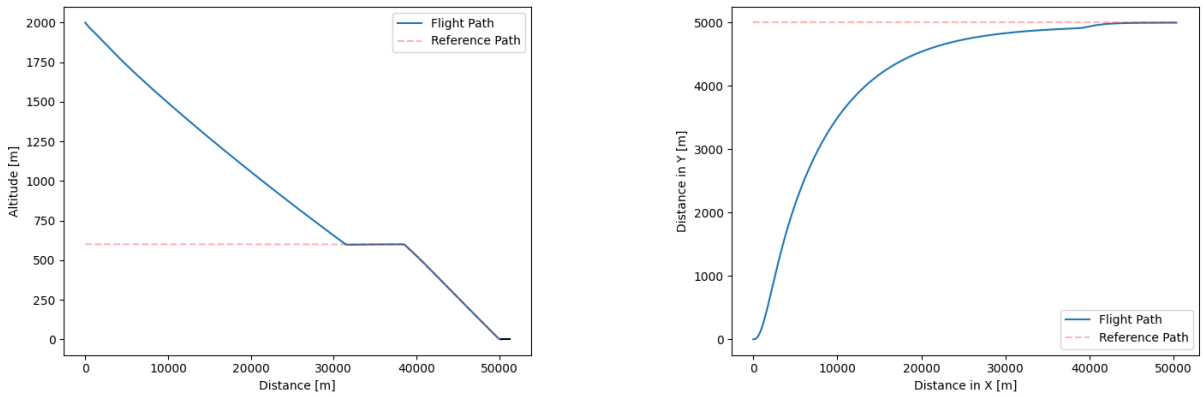
Additionally, the environment also features a model of the ILS, which is used to inform the aircraft about its angular deviation from the runway centre-line and from the glideslope angle. The angle between the aircraft flight path and the runway centre-line is given by λ, in the lateral direction, and the angle between the aircraft fligth path and the desired glideslope angle is given by Γ, in the longitudinal direction. To simulate realistic signals, the ILS is modelled following its required accuracy and characteristics, retrieved from [29]. The GS beam-width is $\pm 0.7°$ with an error no larger than $\pm 0.07°$, similarly, the LOC beam-width is $\pm 2.5°$ with an error no larger than $\pm 0.06°$, and the DME systems accuracy must be within $10m$. The last component of the ILS is the radio altimeter, this is required to inform the aircraft about its altitude and that it can start the flare manoeuvre at the correct height. Since this equipment is considered to be accurate, in this research, radio altimeters were considered to be ideal sensors, and therefore no bias or noise was added to it.

B. Tracking Task & Controller Structure

Here the task designed for this research along with a control structure that is to be used as a base for the different PID and RL controllers is described. The control structure is directly derived from the task and it features a cascaded structure with switching outer loop controllers.

1. Task

The designed landing task is shown in figure 4b and consists of four phases on the longitudinal side and two phases on the lateral side. On the former, the aircraft must descend from its initial altitude ($2000m$) to the ILS intercept altitude ($\approx 600m$), it then briefly holds the altitude such as to intercept the ILS's GS whilst flying horizontally; once it is within reach of the guidance of the ILS it starts to descend with a glideslope angle of $-3°$ until it reaches the flare altitude (see section III.C), at which point it performs the flare and touches down on the runway. On the latter, the aircraft initially follows the guidance of a VOR system, as soon as it intercepts the ILS's signal, it switches to the LOC guidance, where it is more accurately steered to the runway center-line. In the simulation environment, the aircraft starts at position $x_0 = 0m, y_0 = 0m$, and the runway is positioned at $x_r = 50000m, y_r = 5000m$.



(a) Sideview **(b) Top View**

Fig. 4 Landing task

For a successful landing, the aircraft must follow a smooth transition path from the -3° descent angle to the ground level through the flare maneuver. Additionally, the aircraft must reduce its vertical speed for a soft touchdown, however, not too much, such that floating may occur in case wind is present. The pitch and roll angles must be such that there is no tail, nose, or wing strike. The acceptable range of such variables at the touchdown moment is organized in table 3, placed in section IV for practical evaluation of results. On the analysis of the performance of the controllers, this research limits itself to the variables at the touchdown point, and it is assumed the aircraft states at that specific time are sufficient to determine whether the landing was successful or not.

2. Base Controller Structure

The controller structure to be used in this research is directly derived from the tasks in the landing problem at hand. Figure 5 presents the longitudinal control structure which features a cascaded system with an inner-loop *pitch attitude* controller, and outer-loop *altitude*, *glideslope* and *automatic flare* controllers.

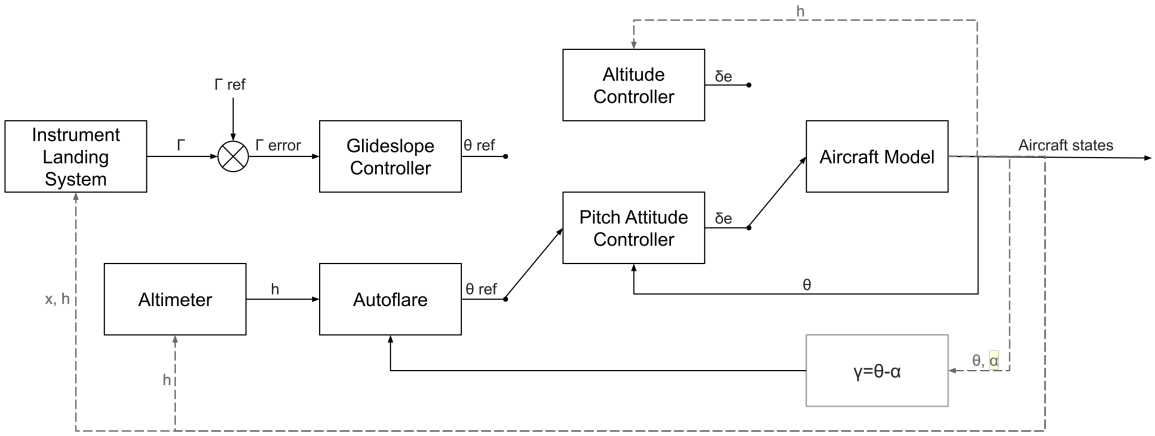


Fig. 5 ALS longitudinal controller structure

For lateral control, the structure features a *VOR hold controller* and a *LOC hold controller*, both of which work in a cascaded fashion with a *roll angle hold controller*. Both of which can be seen in figure 6.

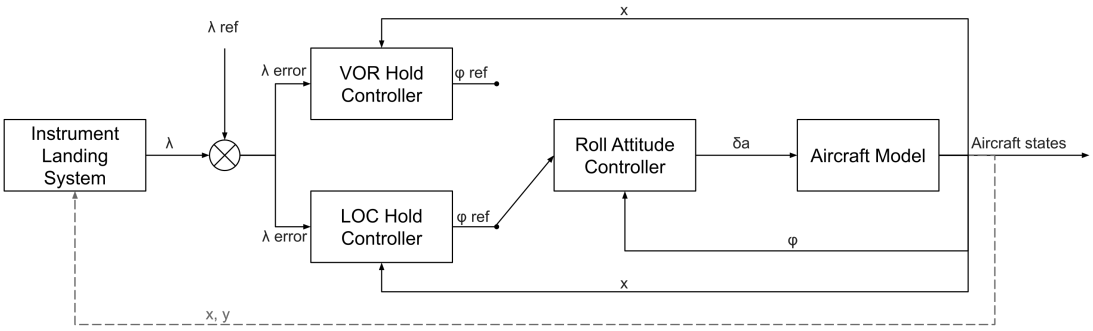


Fig. 6 ALS lateral controller structure

C. PID Controllers

This section describes an automatic landing system designed according to the base controller structure described above. The system only contains PID controllers and it is used to validate the ALS structure, as well as to establish a baseline performance for a landing controller that can be used for comparison. The PID controller design was chosen as a representative of the classical control theory methods that are currently used in ALS, and in aviation in general. Additionally, the outerloop PID controllers are also used in the ALS proposed in this paper, where the aicraft attitude is controlled with reinforcement learning. The PID controllers herein contained were designed loosely following the guidelines and instructions in [30]. Since the implementation and tuning of PID controllers are well-known and this research focuses on the study of reinforcement learning-based controllers these processes are not described in detail. Table 1 is provided containing information about the structure of each of the PID controllers in the ALS structure.

Table 1 PID controllers overview

Controller	Control Surface	Input	Output	Tuning
Pitch Attitude	Elevator	θ_{ref}	δ_e	PI
Altitude Hold	Elevator	h_{ref}	δ_e	PID
Glideslope Hold	-	Γ_{ref}	θ_{ref}	PI
Autoflare	-	h_{ref}	θ_{ref}	PID
Roll Attitude	Aileron	ϕ_{ref}	δ_a	P
VOR Hold	-	Y_{ref}	ϕ_{ref}	PD
LOC Hold	-	Y_{ref}	ϕ_{ref}	PD

To summarise, the Pitch Attitude, Altitude, and Roll Attitude controllers acquire their reference inputs directly from the environment states. The Glideslope acquires its input from the ILS-GS, the VOR-Hold from the VOR, and the LOC hold from the ILS-LOC. The Autoflare controller, however, is more interesting and undoubtedly the most crucial part of the landing procedure. To ensure a smooth landing transition its reference altitude h_{ref} follows an exponential path, given in equation 13. The exponential altitude is a function of the time, containing two design parameters, h_{flare} and τ, both of which are acquired following the method described in [31]. The control law assumes the intended touchdown position to be $1100ft$ $(335, 28m)$ from the GS transmitter, yielding a $\tau = 1.25s$ and $h_{flare} = 5.85m$, a value within the admissible range since most general aviation aircraft start their flare maneuver between $10ft$ $(\sim 3m)$ and $30ft$ $(\sim 10m)$.

$$h_{ref}(t) = h_{flare} * e^{\frac{-t}{\tau}} \tag{13}$$

D. SAC Controller

In order to set up the training for reinforcement learning, the control task is interpreted as an optimal control problem. The evaluation structure follows the structure used in other papers [18] [19] and it employs a continuous reward function that penalises the agent proportionally to the tracking error. The reward is given at each time step as shown in equation 14, where the error e_e is scaled by a scaling factor e_r and subsequently clipped between $[0, 1]$. The rewards at each time step are added up and are used to evaluate the tracking performance of each episode.

$$\tilde{r}(t) = -\frac{1}{3}||clip[e_r \odot e_e(t); 0, 1]|| \tag{14}$$

In order to keep the training more realistic boundaries were set for pitch and row angles if the agent steers the aircraft through one of these boundaries the simulation ends and the agent is penalised proportionately to the amount of time left in the episode. The penalty for ending the simulation early is taken from [28] and is shown in equation 15, where C_p is empirically set to 2 for a $20s$ simulation episode sampled at $100Hz$. The total rewards per episode are then estimated as the sum of the error-proportional reward plus the early termination penalty, as shown in equation 16.

$$\tilde{r}_{t+1} = -\frac{C_p}{\Delta t} * (t_{max} - t_{end}) \tag{15}$$

$$R_{ep} = \sum_{i=1}^{k} \tilde{r}(t) + \tilde{r}_{t+1} \qquad \text{with} \qquad k = \frac{t_{max}}{\Delta t} \tag{16}$$

1. Pitch Attitude Controller

The majority of the dynamic changes are mainly reflected on the control surfaces and being the attitude controllers the closest to them it is expected that these would be the most affected during the failure cases to be analysed. Additionally, the majority of the manoeuvres made during landing occurs in the longitudinal dimension. Therefore, the first controller to be designed using RL is the pitch attitude controller.

The Pitch Attitude Controller tracks a reference signal θ_{ref} and its task is to keep the error as close to zero as possible. To reduce the amount of unnecessary information, the agent does not observe the full set of environment states, instead, the agent only observes the pitch error θ_e and the pitch rate q. The latter is not strictly necessary, however, it helps the agent to differentiate similar states in different conditions, for instance when $\theta_e = 0$ but the aircraft is tending upwards versus downwards. The inclusion of the pitch rate as agent input increases learning speed as well as the smoothness of the agent's actions. Note that there is no tracking, therefore no rewards, are given for the q.

The action coming from the pitch controller is an elevator deviation δ_e command. The reward scaling and error used in equation 14 are defined in equation 17. The action and observation vectors are defined in equation 18.

$$e_r = \frac{6}{\pi} \qquad \text{and} \qquad e_e = \theta - \theta_{ref} \tag{17}$$

$$a := [\delta_e]^T \qquad \text{and} \qquad s_{obs} := [\theta_e, q]^T \tag{18}$$

2. Roll Attitude Controller

Next, an RL-based Roll attitude controller is designed to track the roll angle reference signal ϕ_{ref}. The agent is very similar to that of the pitch attitude controller, the agent observes two states, namely the roll angle error ϕ_e and the roll rate p, as shown in equation 20. The sole control surface of the roll attitude controller is the ailerons, therefore the output of the agent is the aileron deflection δ_a. The reward scaling and error used in equation 14 are defined in equation 19

$$e_r = \frac{6}{\pi} \qquad \text{and} \qquad e_e = \phi - \phi_{ref} \tag{19}$$

$$a := [\delta_a]^T \qquad \text{and} \qquad s_{obs} := [\phi_e, p]^T \tag{20}$$

3. Network Structure & Hyperparameters

The Deep Neural Network structure used in the SAC algorithm consists of a series of neurons that are linearly combined with each other in four layers: one for the inputs, two for the hidden layers, and one for the outputs. The basic structure of the network was retrieved from [18]. Figure 7, illustrates a basic NN topology with the components mentioned earlier. Both hidden layers have 64 nodes and are fed to Normalisation layers to improve sample efficiency in the fashion described in [32], the normalised values are activated by a ReLu function. The output layer, however, is not normalised and is, therefore, a direct result of the linear combination of the weights of the second hidden layer.

The actor-network outputs two parameters, a mean μ and a standard deviation σ, both of which are used to normally sample and output the agent's action. It is important to note that in the evaluation the last step is foregone and the action is a direct product of the actor's mean. There are two critic networks, following the SAC algorithm, that have the same structure, they take a state and action pair as inputs (observations) and output the $Q - value$ of the given pair. The network parameters are represented by k and θ for the Q-function (critic) and the policy (actor), respectively.

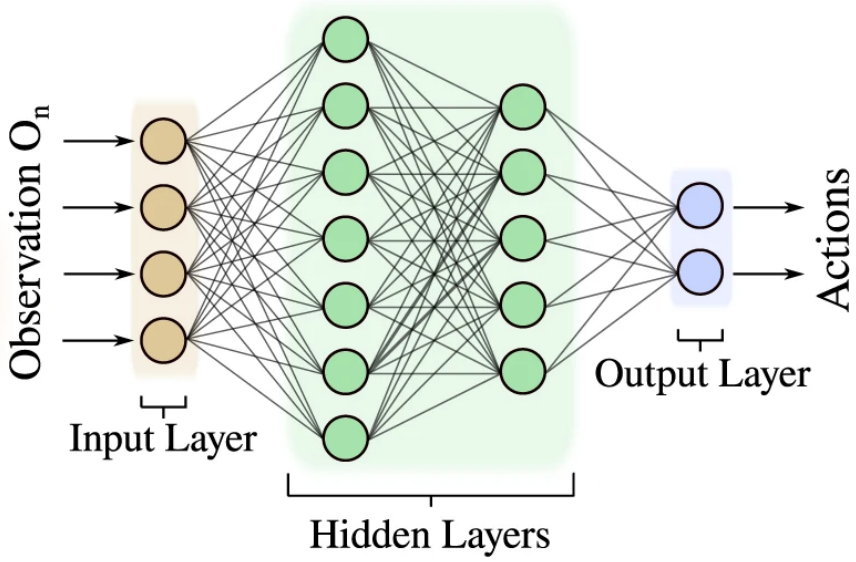


Fig. 7 Neural Network topology, adapted from [33]

The utilisation of correct hyperparameters is known to dramatically improve sample efficiency and learning speed. However, there are far too many parameters to be considered, making tuning time-consuming, difficult and not always resulting in optimal configuration. Fortunately, there have been previous studies using the SAC algorithm applied to the controllers on the PH-LAB [18] [19]. A brief parameter search was performed to evaluate the role of certain hyperparameters in the learning process and the performance of the resulting agents. It was concluded that a learning rate of 4.4×10^{-4} with a linear decrease until the end of the training, proposed in [18], results in fast learning and consistent convergence. The fast convergence means that the algorithm does not train for long with large errors, since these rapidly decrease, therefore, the use of lower learning rates could lead to an algorithm that has spent more time in those areas and therefore is more robust. The consequence of that was unstable training, which is a common issue with DRL algorithms and has already been reported in previous research on SAC-based controllers. Therefore, the learning rate was kept at 4.4×10^{-4} for this research, a value that favours training stability.

Additionally, a brief parameter search was performed on the CAPS scaling coefficients using values of 40, 400, and 40000. The results were coherent with what was reported in the original smoothing policy paper [25], the simulation with lower value showed extremely oscillatory actions, and the simulation with higher values showed much smoother actions, however, very poor training stability and degraded tracking performance. Therefore, the CAPS coefficient value of 400 from [19] was maintained in this research, offering a good balance between smoothness, tracking, and training stability.

The Network initialization followed the Xavier weight initialization method [34], a gain of 0.25 and no bias showed the best results. Higher gain values showed exploding weights and inconsistent training. Additionally, the gradient descent of the algorithm uses Adam optimizer, a method that has proven its superiority in learning stability [35].

4. Agent Training

The SAC algorithm was trained offline for 2×10^5 timesteps, about half of the length usual DRL algorithms are trained for, this is because the agent was capable of learning its task relatively rapidly which lead it to try to improve an already good performance for a long period, resulting in overtraining and loss of generalization. During training, the agent was set to track a sinusoidal signal for a series of around 250 episodes of $20s$, sampled at $100Hz$. A brief search was made between different training signals and a sinusoidal reference signal was chosen instead of a fixed or step reference because it resulted in more stable agents that were better capable of handling the constantly changing reference coming from the outer loop during the landing procedure, as well as presented less sings of command oscillations.

Additionally, to widen the flight envelope that the controller is capable of controlling the aircraft and to ensure that during training it does not get stuck in local optima exploration is enforced in the first stages of training. This is done through enforcing 10% of completely random actions, independent of the policy, and the rate is linearly decreased and maintained at 1% by step 1×10^5 all the way until the end of training.

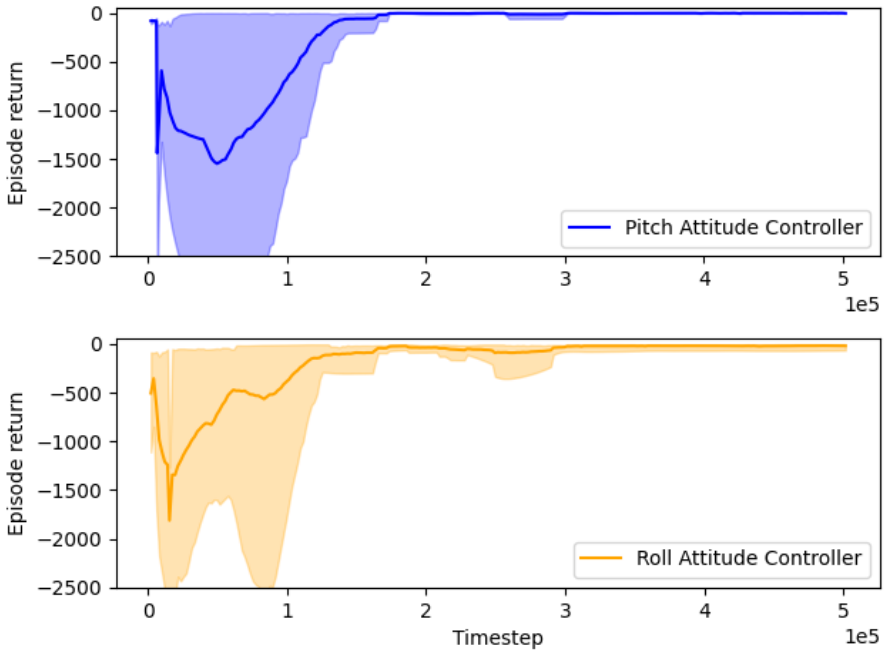


Fig. 8 SAC Learning curve - Line and the shaded area represent the mean and the standard deviation, respectively, across different trainings.

Figure 8 shows the learning curve of four different pitch and roll attitude controllers that were made using random seed initialisation. For both controllers, it can be seen that the rewards per timestep initially decrease as the agents explore the environment, but quickly start to increase as the agent begins differentiating beneficial from detrimental actions. Using the reward system described above the pitch agent and the roll controllers achieve rewards higher than -5 in 2×10^5 timesteps, and there is no significant improvement for either of the controllers in the remaining training period, suggesting that training is complete. The results presented in section IV were achieved using the agent with the highest rewards and less noisy commands from those whose learning curves are shown in figure 8. Table 2, shows an overview of the most important hyperparameters used in this research.

Table 2 SAC algorithm and Network Hyperparameters

Hyperpatameter	Agent
Learning rate λ_r	0.00044
Hidden Units lxl	64
Entropy Target	0.0
Discount Factor γ	0.99
Activation function	ReLu
Memory buffer size	$1x10^6$
Minibatch size	256
Smoothing Factor	0.995

IV. Results & Discussion

This section presents a comparison between the performances of the controllers designed in section III.B on the landing task, described in the same section, employing only PID controllers and employing RL Pitch and Roll controllers. Both iterations of the ALS are evaluated under the nominal condition under ideal system conditions, to establish a baseline for their performance, and subsequently, under various failure cases, to establish robustness and fault tolerance.

There are several types of failures an aircraft might face during flight, however, it is infeasible to test every one of them. Therefore, a few failures where a change in the plant dynamic is unexpectedly introduced to the system were chosen in order to evaluate the robustness and fault-tolerance of the controllers, in both, longitudinal and lateral directions. The simulations are performed considering ideal sensors, the use of real-world sensors is discussed in section IV.D.1. The evaluation follows the performance criteria in table 3, whose parameters have already been discussed in section III.B.

Table 3 Acceptable range of variables at touchdown, adapted from [20]

Metric at touchdown	Limit values (95%)		Unit	Reason for limits
Longitudinal Position X_e	-90	+270	[m]	Touchdown on the runway with adequate braking distance
Lateral Position Y_e	-8	+8	[m]	Touchdown with main gear 1.5[m] from runway edge
Lateral Velocity	-2.5	+2.5	[m/s]	Limit risk of leaving runway after touchdown
Altitude Rate	-1.8	0.0	[m/s]	Limit landing gear/tire damage & passenger comfort
Pitch angle	0.0	+5	[deg]	Limit risk of noise-wheel landing or tail drag
Roll angle	-5	+5	[deg]	Limit risk of damage to wing tips or engine nacelle

A. Nominal System

As part of this research, it must be proven that the proposed controller is able to perform the landing task in the nominal condition and with a system where no failures are present. Here, both PID and RL use the same outer-loop controller tuning, with the exception of Autoflare, which is re-tuned such that the PID and the RL achieve performance within the acceptable range of variables. In an attempt to make the comparison between the two a fair one, the tuning was done such that both controllers achieve similar touchdown performance under non-failed conditions. Noticeably, the RL pitch flare tracking is worse than that of the PID, however, the variables at touchdown are similar, this is because in the re-tuning of the Autoflare for the RL pitch controller, a trade-off between tracking and touchdown conditions had to be made. For this research, the touchdown variables are more important than the tracking itself, in this case, the reference path only provides guidelines for an acceptable landing, however, it is not strictly necessary that it is followed to achieve a landing performance within the acceptable boundaries.

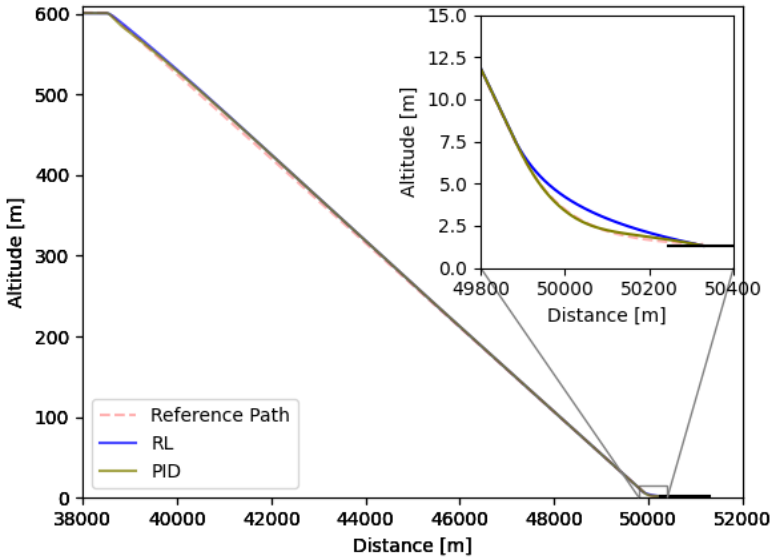


Fig. 9 PID & RL Tracking performance in the landing task - Nominal Case, No Failures

Table 4 shows that the touchdown data for both controllers are very similar under the nominal case, although it can be noticed that the RL controller has a slightly worse touchdown position than the PID. Figure 10 shows the agent's inputs and the trajectory of the aircraft. Both controllers are able to follow the reference signals in a similar fashion, the RL controller, however, does so with much more aggressive command behaviour, without becoming unstable. Figure 11 shows the aircraft's states throughout the landing task. The performance of both controllers is similar, with a few differences in immediate behaviour when there is a switch between controllers, however, both of the eventually converge to the same values.

Table 4 PID & RL-SAC performance in the landing task - Nominal Case

Metric at touchdown	PID	RL	Unit
Longitudinal Position X_e	-1.36	-5.24	[m]
Lateral Position Y_e	-0.3	-0.61	[m]
Lateral Velocity	0.01	0.01	[m/s]
Altitude Rate	-0.42	-0.41	[m/s]
Pitch angle	1.74	1.82	[deg]
Roll angle	0.0	0.0	[deg]

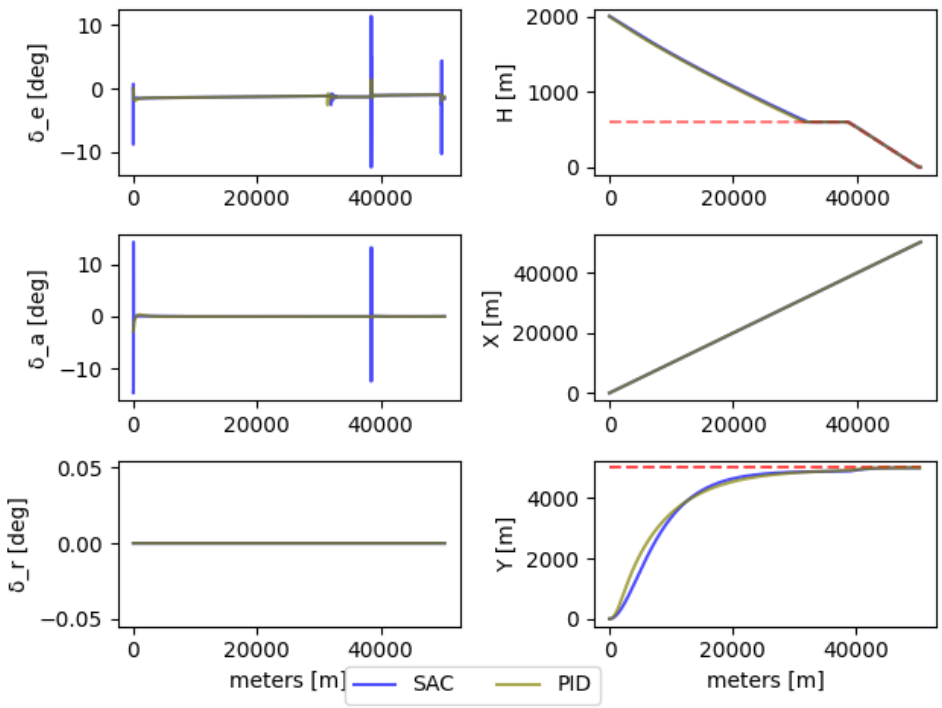


Fig. 10 PID & SAC inputs and trajectory in the nominal case, no failures present - Longitudinal and lateral references represented by the red dashed line

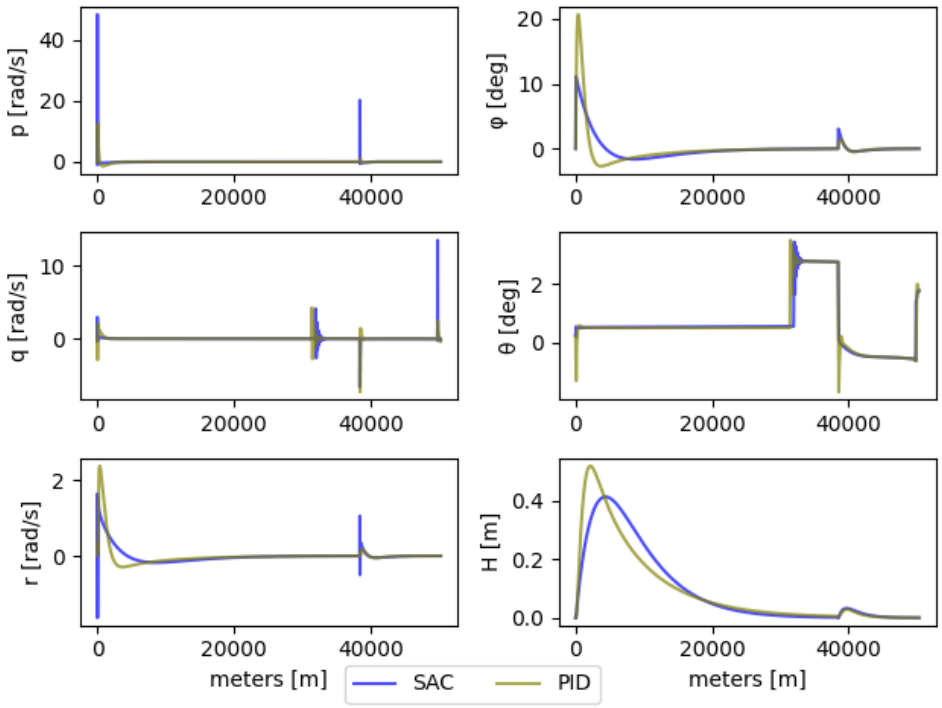


Fig. 11 PID & SAC aircraft states in the nominal case, no failures present

B. Sensor Failure - Noise

Aircraft systems are not ideal and operate under certain tolerances, the ILS acceptable errors are discussed in section III.A. Table 5 shows the data at touchdown when the maximum acceptable positive and negative biases have been added to the ILS LOC and GS reference signals. It shows that both controllers are able to successfully land the aircraft even with biased ILS. The following two failures consider the case when the ILS is defective making the transmitted signals not only biased but also noisy.

Table 5 PID & RL-SAC performance in the landing task - Realistic ILS

Metric at touchdown	PID	RL	PID	RL	Unit
	Positive error		Negative error		
Longitudinal Position X_e	-7.16	-1.56	7.58	24.38	[m]
Lateral Position Y_e	1.94	2.04	-1.93	-1.81	[m]
Lateral Velocity	-0.11	-0.11	0.11	0.11	[m/s]
Altitude Rate	-0.43	-0.47	-0.43	-0.42	[m/s]
Pitch angle	1.73	1.77	1.74	1.8	[deg]
Roll angle	0.0	0.0	0.0	0.0	[deg]

1. GS Noise

Table 6 shows the controller performance when the transmitted ILS GS signal is noisy. To simulate that, Gaussian noise with mean $\mu = 0°$ and standard deviation $\sigma = 0.1°$ is added to the GS reference signal. The table shows the average and standard deviation for each of the parameters calculated from fifty simulation runs. The RL-based controller presented lower average longitudinal error as well as lower standard deviation than the PID-based controller. The latter landed the aircraft before or after the limiting landing longitudinal position 11 out of the 50 simulation runs performed, whilst the RL-based controller was able to land the aircraft 100% of the times. Figure 12, shows the landing position in X and Y directions for all 100 simulation runs; the image clearly reflects the variance data in the table, it shows that the PID controller has a more varied landing position in the X direction than the RL, which has landing positions closer to each other. The other parameters are not affected by the noise, which is expected because the GS mainly affects the longitudinal movement of the aircraft. This test shows that the RL-based pitch attitude controller is capable of correctly performing the flare independently of the aircraft conditions at the starting point, or the end of the glideslope descent. Therefore, here the algorithm shows its robustness by handling the noisy signal provided by the ILS during the final approach.

Table 6 PID & RL-SAC performance in the landing task - Failed ILS: GS Noise

Metric at touchdown	PID		RL		Unit
	μ	σ	μ	σ	
Success rate	78.0	-	100.0	-	%
Longitudinal Position X_e	25.01	155.845	12.35	61.97	[m]
Lateral Position Y_e	-0.48	1.9	1.74	1.81	[m]
Lateral Velocity	-0.01	0.06	0.03	0.08	[m/s]
Altitude Rate	-0.38	0.07	-0.46	0.1	[m/s]
Pitch angle	1.77	0.05	1.78	0.0	[deg]
Roll angle	0.0	0.0	0.0	0.0	[deg]

2. LOC Noise

Table 7 shows the controller performance when the transmitted ILS LOC signal is noisy. The simulation is done in a similar manner as the previous test, Gaussian noise is added to the transmitted signal, this time however, with mean $\mu = 0°$ and standard deviation $\sigma = 0.03°$. In this situation, the RL-based controller presents higher error and deviation in the longitudinal direction, this is due to the aileron effect on the longitudinal direction. Additionally, both controllers present similar deviations in the lateral direction, however, the RL-based controller averages a much higher error, leading it to overshoot and land on the right of the runway 30% of the time. In this case, the RL controller is capable of performing consistently, similar to the PID, however, with a larger error in the lateral direction. The results show that indeed the SAC algorithm can be robust and insensitive to noise, by landing the aircraft in very close proximity between different simulations. However, it also shows higher deviation from the runway centre-line, as can be seen in figure 13.

Table 7 PID & RL-SAC performance in the landing task - Failed ILS: LOC Noise

Metric at touchdown	PID		RL		Unit
	μ	σ	μ	σ	
Success rate	100	-	100	-	%
Longitudinal Position X_e	15.38	5.11	28.64	4.89	[m]
Lateral Position Y_e	-1.83	0.59	-5.93	0.81	[m]
Lateral Velocity	-0.01	0.09	0.18	0.07	[m/s]
Altitude Rate	-0.43	0.0	-0.41	0.01	[m/s]
Pitch angle	1.74	0.0	1.8	0.01	[deg]
Roll angle	0.0	0.0	-0.02	0.01	[deg]

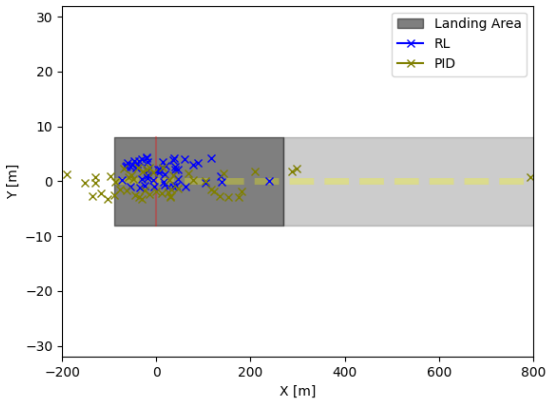


Fig. 12 Landing positions With GS Noise

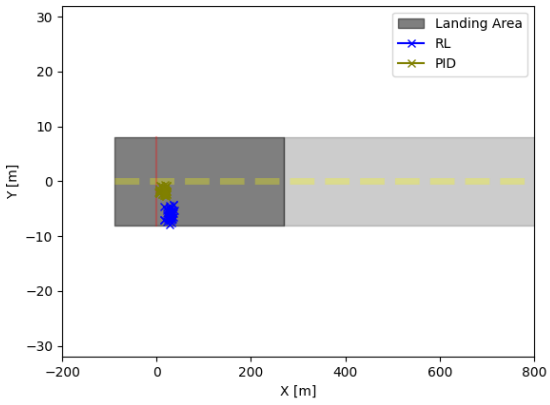


Fig. 13 Landing positions With LOC Noise

The low deviation shows that the RL roll attitude controller can be robust, a fact that is corroborated by the results with the Pitch attitude controller. However, there is still the issue of high average error, this can be due to the controller not being trained with the PID outer loop or without the presence of the pitch attitude controller, both of which are factors that influence the performance of the controller.

C. Actuator Failure - Loss of Efficiency

Table 8 presents the touchdown data in case of an elevator Loss of Efficiency (LOE) during the glideslope descent ($t = 500s$ in the simulation). First controllers are tested with actuator efficiency of 90%. and then gradually reduced until they cannot comply with one of the pre-determined criteria. The performance of the PID controller rapidly decreases, with the controller barely able to land the aircraft on the runway with only a 10% reduction in elevator efficiency. This data clearly shows that the PID requires the use of some extra tool to be able to deal with this failure, for instance, gain

scheduling since the performance can be ameliorated by a simple increase in PID gain.

The RL controller is able to handle much larger reductions in elevator efficiency, this is because the training resulted in a very fast controller that outputs very high deflections, whilst still being stable at the same time. The reduction in elevator efficiency essentially reduces the output of the controller, in the case of the RL controller this is not a problem, because its output is initially already high, the PID, on its own, presents a much calmer behavior which does not handle the LOE of the elevator. For the RL controller a reduction in the effectiveness of 70% is required before it no longer complies with the criteria, when it eventually lands before the runway.

Table 8 PID & RL-SAC performance in the landing task - Actuator failure: Elevator LOE

	PID	RL	PID	RL	RL	Unit
Effectiveness	90%		80%		30%	
Longitudinal Position X_e	-83.2	-10.58	-217.5	-21.5	-91.25	[m]
Lateral Position Y_e	-0.31	-0.61	-0.31	-0.61	-0.62	[m]
Lateral Velocity	0.01	0.01	0.01	0.01	0.01	[m/s]
Altitude Rate	-0.29	-0.43	-0.56	-0.46	-0.66	[m/s]
Pitch angle	1.81	1.8	1.99	1.79	1.68	[deg]
Roll angle	0.0	0.0	0.0	0.0	0.0	[deg]

D. Additional Results

1. Biased-noisy aircraft Sensors

As mentioned previously and tested in the previous sections, in reality, aircraft systems are not ideal and bias and noise are present in all of them. During the validation study [27] of the PH-LAB the characteristics of the aircraft's sensors were measured and can be reproduced using Gaussian noise according to the parameters in table 9.

Table 9 PH-LAB sensor noise characteristics, from [27]

State(s)	Bias	Variance	Unit
p, q, r	$3x10^{-5}$	$4x10^{-7}$	[rad/s]
θ, ϕ	$4x10^{-3}$	$1x10^{-9}$	[rad]
β	$1.8x10^{-3}$	$7.5x10^{-8}$	[rad]
h	$8x10^{-3}$	$4.5x10^{-3}$	[m]

Table 10 PID & RL-SAC performance in the landing task - Realistic aircraft sensors

Metric at touchdown	PID		RL		Unit
	μ	σ	μ	σ	
Success rate	100	-	100	-	%
Longitudinal Position X_e	7.62	13.85	13.99	19.99	[m]
Lateral Position Y_e	-5.72	0.0	-5.6	0.0	[m]
Lateral Velocity	0.0	0.0	0.0	0.0	[m/s]
Altitude Rate	-0.36	0.01	-0.19	0.03	[m/s]
Pitch angle	1.78	0.01	1.96	0.02	[deg]
Roll angle	0.23	0.0	0.23	0.0	[deg]

To evaluate the performance of the controllers in a situation closer to the real world the biased noisy states were implemented in the environment and another fifty simulation runs were made in the Nominal Case, the results can be

seen in table 10. At first, it was noticed that both controllers were landing before the runway, therefore, the Autoflare was re-tuned for both of them and in both cases the integral and the differential gains were increased until acceptable landing positions were achieved. Both controllers were able to land the aircraft in all simulation runs. The RL controller showed a slightly larger variance in the longitudinal direction and both controllers showed significant overshoot in the lateral direction, in about the same dimension. Figure 14 shows that the noisy sensors affect the lateral landing position in a similar manner for both the PID and the RL controllers, where both of them land to the right of the runway centre-line, with very little variation. On the longitudinal direction, however, unlike the test presented in the previous section, the RL controller presents more variance than the PID controller, indicating that aircraft internal sensor noise and bias may have more effect on these controllers than external sensors, such as the ILS.

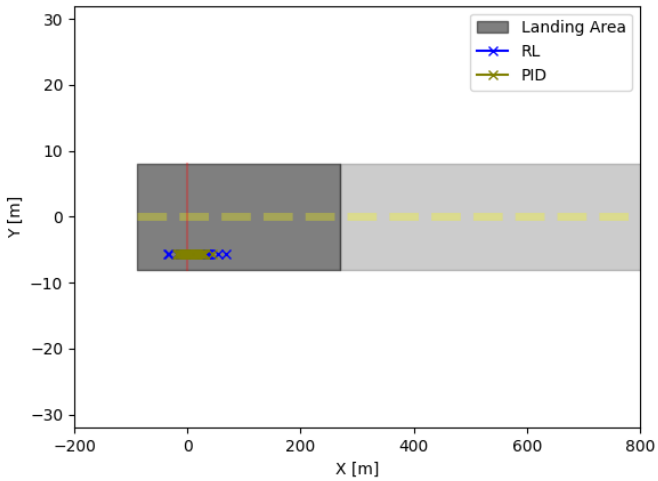


Fig. 14 Landing positions With realistic sensors - Acceptable landing area in dark grey

2. Training Stability & repeatability

The SAC algorithm strongly relies on randomness, from its stochastic policy, to minibatch sampling, to Deep Neural Network initialization parameters. This randomness of the policy ensures that the algorithm provides some level of exploration during training, hence generating a more robust controller that has explored different areas of the flight envelope. However, this randomness also hinders training stability, since every training run is different from the next, hyperparameter tuning is also more difficult due to these differences. Therefore, the training stability issues of the algorithm still remain and it is possible to see dips in rewards even in the latter stages of training, which diminishes the confidence of the algorithm. Nonetheless, as long as the algorithm is trained in an offline manner, multiple agents are trained, and selection and evaluation are performed to check the performance of the resulting controllers, their performance should be in line with the results presented in this research.

The controller herein presented also suffer from overtraining, an issue identified when the agents were trained with fixed reference signals. In these cases, the agent developed aggressive behaviour resulting in extremely oscillatory commands to the control surfaces. This issue was partially solved by using a sinusoidal reference signal for training, however, some of the agents still present such behaviour, especially in training periods longer than 5×10^5 timesteps. From the agents used in this paper, one out of four presented such behaviour, however, a longer study would have to be done to determine the exact percentage of agents that present such behaviour.

V. Conclusion

This research demonstrates that reinforcement learning controllers can be used for aircraft control in landing tasks, through the training of pitch and roll controllers based on the SAC algorithm. A cascaded ALS controller structure where the RL controllers were implemented in the inner loop and PID controllers were implemented in the outer loop. The performance of the ALS controller was compared using RL and PID inner controllers, and it showed that the SAC controller is capable of achieving acceptable performance that is similar to that of the traditional PID in the nominal case, both with ideal and biased and noisy aircraft sensors as well as biased ILS signals. Additionally, sensor and actuator failure were tested in order to determine the robustness and failure tolerance of the controllers. The RL controller showed good performance in the longitudinal direction, where the pitch controller surpassed the performance of the PID when noise is added to the GS signal and in the case of the elevator reduced efficiency. On the lateral direction, however, albeit performing similarly to the PID in the nominal case, the Roll attitude controller was not able to handle noise added to the LOC signal, rapidly overshooting the runway and landing outside of it.

This research shows that the SAC algorithm can be used to create pitch and roll controllers that are trained separately and still be robust enough to work in conjunction with each other in a larger controller structure. Additionally, it also shows that model-free DRL algorithms are indeed capable of creating robust controllers that can be insensitive to noise. In addition to that, the robustness of the SAC algorithm can still be retained even if the reference signal is dictated by a PID controller, technically a more error-prone method in the face of noise and non-linearities.

The current challenges with RL controllers are clear in this research, where training is shown to easily turn inconsistent depending on hyperparameters. Offline training signifies that the controller is only robust to the area that it has been exposed to and may not function at all when situations steer the aircraft outside of that. Overtraining and inconsistent performance have been improved through the use of CAPS parameters, however, are still issues with RL controllers.

This research however, shows that, despite the challenges of the field, RL is capable of generating high-performing controllers that can also surpass the performance of traditional controllers for aircraft attitude control and work in combination with PID controllers to improve the longitudinal performance of ALS. Further research is recommended utilizing full attitude RL controllers to enhance longitudinal and lateral performance when controlled in the same system, and/or the use of RL-based on the outer loop of the ALS controller. A further study on the over-trained agents and possible solutions to solve the issues presented in section IV.D.2 is recommended, premature truncation of training or the addition of penalties for oscillatory behaviour on the reward system are possible solutions. Additionally, the use of iADP in combination with DRL has been shown to improve the performance of attitude and altitude controllers [19], therefore, further research on ALS using this hybrid technique is recommended.

References

[1] Boeing, *Statistical summary of commercial jet airplane accidents*, Aviation Safety, Seattle, Washington, 2021.

[2] IATA, *Safety Report 2019*, 56th ed., International Air Transport Association IATA, 2020.

[3] Charnley, W. J., "Blind Landing," *The Journal of Navigation*, Vol. 12, No. 2, 1959, p. 115–140. https://doi.org/10.1017/S037346330001794X.

[4] Harris, J. J., and Stanford, J. R., "F-35 flight control law design, development and verification," 2018. https://doi.org/10.2514/6.2018-3516.

[5] Canin, D. G., McConnell, J. K., and James, P. W., "F-35 high angle of attack flight control development and flight test results," American Institute of Aeronautics and Astronautics Inc, AIAA, 2019, pp. 1–29. https://doi.org/10.2514/6.2019-3227.

[6] Guan, Z., Ma, Y., Zheng, Z., and Guo, N., "Prescribed performance control for automatic carrier landing with disturbance," *Nonlinear Dynamics*, Vol. 94, 2018. https://doi.org/10.1007/s11071-018-4427-3.

[7] Ismail, S., Pashilkar, A. A., Ayyagari, R., and Sundararajan, N., "Improved neural-aided sliding mode controller for autolanding under actuator failures and severe winds," *Aerospace Science and Technology*, Vol. 33, 2014, pp. 55–64. https://doi.org/10.1016/j.ast.2013.12.016.

[8] Liao, F., Wang, J. L., Poh, E. K., and Li, D., "Fault-tolerant robust automatic landing control design," *Journal of Guidance, Control, and Dynamics*, Vol. 28, 2005, pp. 854–871. https://doi.org/10.2514/1.12611.

[9] Tamkaya, K., Ucun, L., and Ustoglu, I., "H-based model following method in autolanding systems," *Aerospace Science and Technology*, Vol. 94, 2019. https://doi.org/10.1016/j.ast.2019.105379.

[10] Shue, S. P., and Agarwal, R. K., "Design of automatic landing systems using mixed H2/H control," *Journal of Guidance, Control, and Dynamics*, Vol. 22, 1999, pp. 103–114. https://doi.org/10.2514/2.4356.

[11] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A., "Playing Atari with Deep Reinforcement Learning," *CoRR*, Vol. abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

[12] "Mastering the game of Go with deep neural networks and tree search," *Nature*, Vol. 529, 2016, pp. 484–489. https://doi.org/10.1038/nature16961.

[13] Zhou, Y., "Online reinforcement learning control for aerospace systems," Ph.D. thesis, Delft University of Technology, 2018. https://doi.org/10.4233/uuid:5b875915-2518-4ec8-a1a0-07ad057edab4.

[14] Zhou, Y., van Kampen, E., and Chu, Q., "Incremental model based online heuristic dynamic programming for nonlinear adaptive tracking control with partial observability," *Aerospace Science and Technology*, Vol. 105, 2020. https://doi.org/10.1016/j.ast.2020.106013.

[15] Zhou, Y., van Kampen, E. J., and Chu, Q. P., "Incremental model based online dual heuristic programming for nonlinear adaptive control," *Control Engineering Practice*, Vol. 73, 2018, pp. 13–25. https://doi.org/10.1016/j.conengprac.2017.12.011.

[16] Konatala, R., van Kampen, E., and Looye, G., "Reinforcement Learning based Online Adaptive Flight Control for the Cessna Citation II(PH-LAB) Aircraft," *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics Inc. (AIAA), United States, 2021. https://doi.org/10.2514/6.2021-0883, virtual/online event due to COVID-19; AIAA Scitech 2021 Forum ; Conference date: 11-01-2021 Through 21-01-2021.

[17] Heyer, S., Kroezen, D., and van Kampen, E.-j., "Online Adaptive Incremental Reinforcement Learning Flight Control for a CS-25 Class Aircraft," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics Inc. (AIAA), United States, 2020. https://doi.org/10.2514/6.2020-1844, aIAA Scitech 2020 Forum ; Conference date: 06-01-2020 Through 10-01-2020.

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

[19] Teirlinck, C., "Reinforcement Learning for Flight Control Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance," Master's thesis, Delft University of Technology, the Netherlands, 2022. URL https://repository.tudelft.nl/islandora/object/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85?collection=education.

[20] Myron Kayton, W. R. F., *Avionics Navigation Systems*, John Wiley Sons, Inc., 1997. https://doi.org/10.1002/9780470172704.

[21] Borst, C., "Avionics - Landing Systems - Lecture Slides," , February 2022. Delft Univerity of Technology.

[22] Pallett, E. H. J., Coyle, A. S., and Blackwell, M., *Automatic Flight Control Fourth Edition*, 4^{th} ed., Wiley-Blackwell, 1993. URL www.blackwellpublishing.com.

[23] Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M., "Parameter Space Noise for Exploration," *CoRR*, Vol. abs/1706.01905, 2017. URL http://arxiv.org/abs/1706.01905.

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

[25] Mysore, S., Mabsout, B., Mancuso, R., and Saenko, K., "Regularizing Action Policies for Smooth Control with Reinforcement Learning," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1810–1816. https://doi.org/10.1109/ICRA48506.2021.9561138.

[26] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., and Levine, S., "Soft Actor-Critic Algorithms and Applications," *CoRR*, Vol. abs/1812.05905, 2018. URL http://arxiv.org/abs/1812.05905.

[27] van den Hoek, M. A., de Visser, C. C., and Pool, D. M., "Identification of a Cessna Citation II Model Based on Flight Test Data," *Advances in Aerospace Guidance, Navigation and Control*, edited by B. Dołęga, R. Głębocki, D. Kordos, and M. Żugaj, Springer International Publishing, Cham, 2018, pp. 259–277.

[28] Gavra, V., "Evolutionary Reinforcement Learning A Hybrid Approach for Safety-informed Intelligent Fault-tolerant Flight Control," Master's thesis, Delft University of Technology, the Netherlands, 2022. URL https://repository.tudelft.nl/islandora/object/uuid:46989854-bcf0-4d3d-a6bf-f35cbb559d49?collection=education.

[29] United States Department of Transportation, "2001 Federal Radionavigation Systems," Tech. rep., Washington DC, 2001. URL https://rosap.ntl.bts.gov/view/dot/8476.

[30] Nise, N. S., *Control Systems Engineering*, 3rd ed., John Wiley Sons, Inc., USA, 2000.

[31] Blakelock, J. H., *Automatic Control of Aircraft and Missiles*, 2nd ed., John Wiley & Sons, Inc., 1991.

[32] Ba, J., Kiros, J. R., and Hinton, G. E., "Layer Normalization," *ArXiv*, Vol. abs/1607.06450, 2016.

[33] Moro, L., Paris, M. G., Restelli, M., and Prati, E., "Quantum compiling by deep reinforcement learning," *Communications Physics*, Vol. 4, 2021. https://doi.org/10.1038/s42005-021-00684-3.

[34] Glorot, X., and Bengio, Y., "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, Vol. 9, 2010, pp. 249–256.

[35] Kingma, D., and Ba, J., "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations*, 2014.

3

Literature Review

This chapter presents a literature study on the basic topics of this research. It includes a literature review of (Automatic) Landing Systems, its phases, components and requirements, a review of the basics of Reinforcement Learning, its most common frameworks, the state-of-the-art and its challenges. Additionally, a brief review of the related work is also presented for each of the topics. [1]

3.1. Automatic Landing Systems

Automatic Landing or Autoland are systems that have been developed to land an aircraft completely autonomously, allowing landings to happen in weather conditions that would have been dangerous or completely impossible otherwise. This chapter serves as a basis for understanding how such systems work and their characteristics. Section 3.1.1 outlines the Landing procedure, including the flight phases involved in such process, the instrumentation and requirements needed to perform it safely, and how automatic landing is performed. Section 3.1.2 provides an overview of the state-of-the-art methods that are currently used in aviation as well as modern propositions for ALSs. Section 3.1.3 presents the main challenges faced in autonomous landing and how these affect the field. Section 3.1.4 concludes the chapter with a short summary of what has been discussed in the chapter and uses its findings to answer research questions *RQ1.1*, *RQ1.2*, *RQ1.3*.

3.1.1. Aircraft Landing Basics

The first section of this chapter is dedicated to the description of the basic concepts of landing, its phases, the aiding instruments, and the regulations imposed upon the landing process. This part of the research is done in order to introduce the topic of landing, set the basis for understanding automatic landing and identify parameters relevant to the control problem.

Landing Phases

The landing procedure of an aircraft represents a minor part of the entire flight, however, together with the approach phase, it represents the most critical for fatal accidents [8]. Generally, the landing procedure is considered to be composed of four phases: *Glide slope descent* (or final approach), *Flare*, *Touchdown*, *roll-out* [50]. The glide slope descent is the landing phase where the aircraft is aligned to the airfield and begins to descend a predefined straight flight path leading directly to the runway's centre line. The flight path to be followed by the aircraft has a standard angle of -3 degrees for most airports, though this value may vary slightly [55]. Whilst the gliding slope descent path ensures that the aircraft approaches the landing point at the correct position, the glide angle ensures that the aircraft lands at a stable descent rate [67]. Figure 3.1 shows the glide slope path along with the other 3 landing phases.

[1] *Chapter 3 - Literature Review* of the report has already been assessed as part of the *AE4020 Literature Study* course.

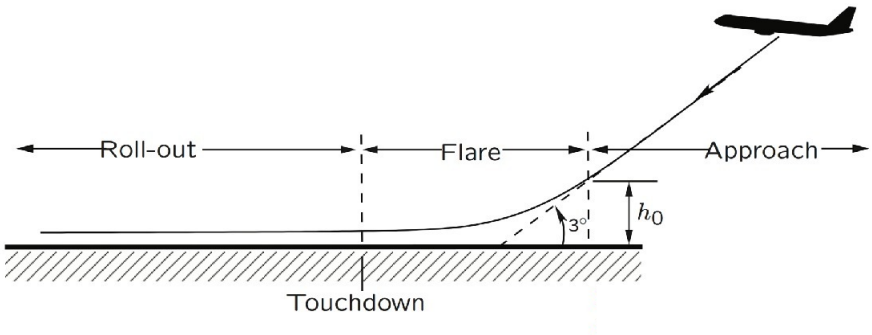


Figure 3.1: Aircraft Landing phases, retrieved from [10]

During the final approach, the aircraft has a sink rate of 6 to 16 ft/s, which is too fast for routine landings in general aviation, therefore, in the final part of the descent the pilot performs a manoeuvre termed flare to reduce the sink rate below 0.9 m/s, or 3 ft/s, [50]. In this operation, the aircraft transitions its descending attitude to an effective landing attitude. During the flare the aircraft follows a curved path by pitching up and reducing the altitude rate, bringing the flight-path angle (γ) to nearly null. This procedure changes the aircraft's attitude and sets it up for a soft touchdown, improving passenger comfort and minimising landing gear impact. The Flare manoeuvre occurs at a height between $30-15ft$ depending on the situation, aircraft type and weight. The flare path is an exponential curve described in equation 3.1, where $h(t)$ is the altitude as a function of time, h_0 is the initial height of the flare, e the exponential function, t the time and τ a time constant.

$$h(t) = h_0 e^{\frac{-t}{\tau}} \tag{3.1}$$

The final two phases of the landing are the touchdown itself and the roll-out. Touchdown is characterised by the landing itself, in other words, the gentle touch of the aircraft's landing gear onto the landing surface. To ensure a smooth and safe landing certain parameters, such as altitude rate, horizontal velocity, pitch angle and the touchdown point, must follow certain thresholds. The analysis of said parameters at touchdown can also be used to evaluate the performance of a specific landing. In this case, it is assumed that, if the parameters at touchdown are within acceptable bounds, it means that the preceding phases of landing must have been performed correctly. The landing, however, in some definitions, is not finished after touchdown, the aircraft is still required to follow the runway and decelerate to taxiing speed, this phase is known as roll-out.

Instrumentation & Requirements

As with many procedures in aerospace, landing requires the use of multiple instruments and aiding systems. According to [52], there are two main ground-based systems that are required to perform landing: 1. the *Instrument Landing System (ILS)*, which is divided in (i) *localiser - LOC*, *glideslope - G/S* and possibly (iii) *beacon markers*, and 2. *Radio Altimeters*. The onboard equipment is limited to suitable receivers and computers that are able to process the information provided by the ground-based equipment.

Fist, the **Instrument Landing System (ILS)** is a navigation-aid system that provides short-range lateral and vertical guidance for aircraft during the landing procedure. The ILS was developed shortly before World War II and it used a complex system f signals and antenna arrays. Nowadays the ILS consists of a localiser antenna (LOC), a glide-slope antenna (G/S) and a beacon marker. Whilst the LOC provides lateral guidance to the aircraft, the G/S provides vertical guidance by means of a descent flight path; The working principle of them is virtually the same. These two systems consist of antennas transmitting two signals that have been modulated at 90 and 150 Hz, such that the receiver can discern them, see figure 3.2. The antennas are positioned in the same place such that the emitting point of the signals is the same. The direction in which the signals are transmitted however is slightly different, this allows the aircraft to receive both of them with varying strengths depending on its position relative to the antenna. The beam directions, however, point in such a way that the aircraft is in the correct position only when the strength of both signals is the same. This is how the pilot is informed whether

the desired path is being followed or not. The LOC indicates if the aircraft is aligned to the centre line of the runway and the G/S indicated if the aircraft is following the correct glide-slope path (of around -3 degrees). The remaining component of the ILS is the beacon markers, which provide the aircraft with information about its distance from the runway, this system is not entirely necessary due to the development of other positioning systems such as the Distance Measuring System (DME) and more recent enhancements to the Global Positioning System (GPS).

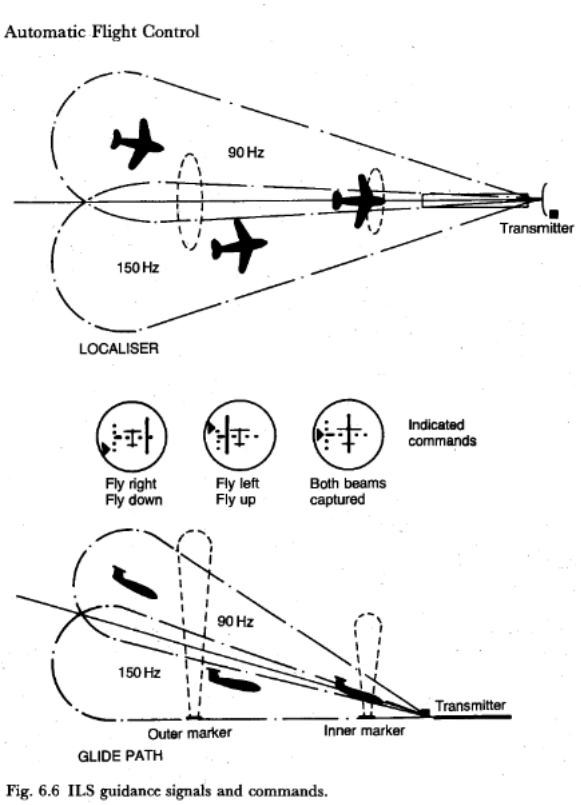


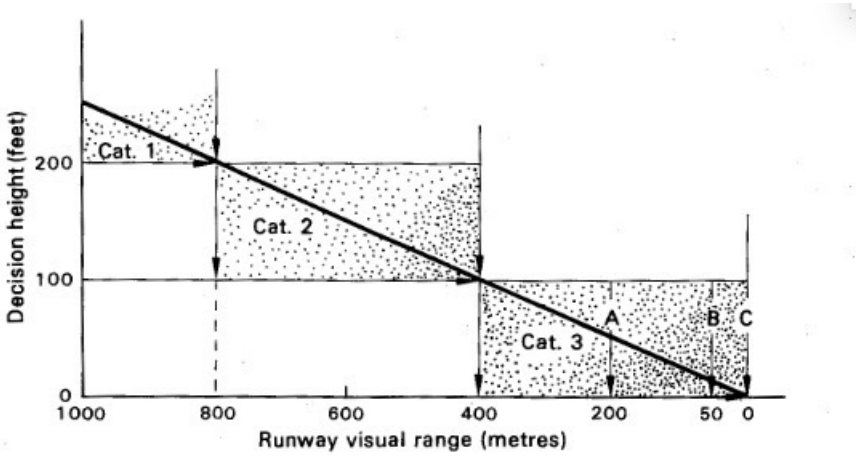
Figure 3.2: ILS Working principle, retrieved from [50].

The ILS feeds safe information to the pilot that can be used in bad weather conditions, then allowing landing to take place in situations that otherwise would not have been possible. Weather conditions and the use of ILS is classified through visibility limitations known as *weather minima*. Low visibility conditions are given in terms of two aspects: *Runway Visual Range (RVR)* and *Decision Height (DH)* [50]. The former refers to the distance over which the pilot can see the centre line of the runway, this value is instrumentally-derived. The latter refers to the decision point in the final approach where the pilot has to decide if there is enough visual reference to continue descending and landing. During landing, the traffic controller informs the pilot if the RVR is above the minimum requirement, allowing the aircraft to descend to decision height, where the pilot must decide if a sufficient segment of the runway can be seen or not. In a positive case, the pilot proceeds to landing, in the negative, the pilot must execute a missed approach, enter a holding pattern and prepare for another landing attempt, or diverge to an alternative airport.

RVR and DH quantities are determined as limiting factors by national licensing authorities - International Civil Aviation Organisation (ICAO), Federal Aviation Administration (FAA) and Joint Aviation Authorities (JAA) (European Union Aviation Safety Agency - EASA). A graphical representation of ICAO's main categories is presented in figure 3.3. Furthermore, each landing category has additional specific requirements on airport equipment, avionics and crew (pilot license and training) [1]. These additional requirements are not further discussed, since it falls outside of the scope of this project, however, they are summarised as:

- CAT I: ILS, beacon markers and can be performed with a single pilot.

- CAT II: Additional to CAT I, it requires a Dual ILS receiver, Radar Altimeter, Autopilot coupler or dual flight director, and can only be performed in the presence of two pilots and additional miss-approach attitude guidance (go-around)
- CAT IIIa: Additional Fail-passive (dual) Autopilot or Head-Up Display (HUD)
- CAT IIIb: Additional Fail-operational (triple) Autopilot and Automatic Roll-out
- CAT IIIc: Has no additional requirements, since it has not been approved anywhere in the world, at the present moment.



Category 1: Operation down to minima of 200 ft decision height and runway visual range of 800 m with a high probability of approach success.

Category 2: Operation down to minima below 200 ft decision height and runway visual range of 800 m, and to as low as 100 ft decision height and runway visual range of 350 m with a high probability of approach success.

Category 3A: Operation down to and along the surface of the runway, with external visual reference during the final phase of the landing down to runway visual range minima of 200 m.

Category 3B: Operation to and along the surface of the runway and taxiways with visibility sufficient only for visual taxiing comparable to runway visual range value in the order of 50 m.

Category 3C: Operation to and along the surface of the runway and taxiways without external visual reference.

Figure 3.3: Graphical representation of ICAO's landing Categories, retrieved from [52]

Furthermore, certain landing conditions - e.g. CAT II & III - require the use of **Radio Altimeters (RA)** to inform the aircraft about its height. Radio Altimeters are mainly used during instrument approach and landing, low level and night flight under 2500ft. During landing it is the RA that provides the primary information about decision height and altitude. This system employs radars to estimate the aircraft's altitude above the terrain and the working principle of the device is based on time difference. The antenna transmits a signal from the aircraft to the ground, and the time it takes for it to travel to the ground, reflect and return to the aircraft is computed. The wave's travel time is used in combination with the wave's speed - the speed of light - to estimate the aircraft's height Above Ground Level (AGL)

Additionally, ICAO has also determined *Aircraft Approach Categories*, in which aircraft are differentiated based on their required landing seed as they approach the runway [1]. The main determinant factor is the aircraft's size and weight, and the categories serve as a basis for determining if landing operations are safe to be performed. The ICAO Doc 8168 also provide guidelines for *maximum and minimum descent rate* and other parameters. Additionally, there are additional parameters that help to determine if the landing was performed properly or needs improvement. Table 3.1, retrieved from [50] presents the constraints that must be followed by aircraft at touchdown to constitute an acceptable landing of a jet transport aircraft.

Table 3.1: Acceptable range of variables at touchdown, adapted from [50]

Metric at touchdown	Limit values (95%)		Unit	Reason for limits
Longitudinal Position X_e	-90	+270	[m]	Touchdown on the runway with adequate braking distance
Lateral Position Y_e	-8	+8	[m]	Touchdown with main gear 1.5[m] from runway edge
Lateral Velocity	-2.5	+2.5	[m/s]	Limit risk of leaving runway after touchdown
Altitude Rate	-1.8	0.0	[m/s]	Limit landing gear/tire damage & passenger comfort
Pitch angle	0.0	+5	[deg]	Limit risk of noise-wheel landing or tail drag
Roll angle	-5	+5	[deg]	Limit risk of damage to wing tips or engine nacelle

3.1.2. Challenges of Automatic Landing Systems

Approach and landing under all visibility and weather conditions are, in the face of the fatal accidents in this flight phase and the nature of it, undoubtedly the most difficult manoeuvre a pilot has to perform. During the landing process, it is required to control the aircraft's attitude in all three axes simultaneously, besides controlling vertical and horizontal velocities. As in most automatic systems, the performance of an autoland system must exceed those of pilots, and provide exceptional guidance and control in order to be acceptably used. The United Kingdom certification authorities require a minimum reliability value of 1 in 10^7 for fatal accidents [52]. These factors make the effective implementation of automatic landing systems a difficult task at best and reinforce the need for more adaptive and robust control methods that are capable of providing accurate control and achieving fault tolerance.

An autoland generally makes use of the ILS that indicates the aircraft of its positioning with respect to the runway. The flight control computer then uses that information to control the aircraft's control surfaces and the throttles, such that it follows the desired paths provided by the ILS and maintains the desired velocity. The flight computer is also responsible for inducing the flare motion of the aircraft at the correct height - requires accurate height above ground - and activating the auto-brake system once touchdown occurs. Therefore, autoland requires the use of a radar altimeter to determine the aircraft's height above the ground very precisely so as to initiate the landing flare at the correct height. Most Autoland systems can operate with a single autopilot in an emergency, but they are only certified when multiple autopilots are available.

Additionally, automatic landing systems make use of various Automatic Flight Control Systems (AFCS) present in the aircraft, such as longitudinal and lateral autopilot modes (Pitch attitude, Airspeed, Roll and heading angle, amongst others), and navigational autopilot systems (Glideslope hold, automatic flare, amongst others). The ALS receives multiple inputs from several systems at the same time, and the failure of a single system could result in catastrophic situations. Therefore, Automatic landings require high degrees of redundancy and to be *failure tolerant (fail-active)*. For example, the failure of one of the automatic systems during flare or roll-out could cause the control surfaces to fully deflect in one direction, known as "hard over", which due to the proximity to the ground during landing could result in fatal accidents. Autolandings are also *fail-passive*, which means that they are able to detect if it cannot perform its task, warn the pilot if this is the case, turn itself off and hand over control of the aircraft; ideally, this would happen before decision height, such that the pilot is able to determine whether to continue with the landing or perform a missed approach. The majority of the automatic landing systems are capable of performing their tasks with a single autopilot, however, they are only certified to do so when multiple are available [52].

From the control point of view of ALS, currently, the majority of automatic landing systems present in general aviation strongly rely on traditional control techniques - e.g. PID controllers. The reason behind this choice is that classical control methods are efficient, relatively simple to implement, and reliable [66]. However, the downsides of PID controllers are their (i) high dependence on mathematical models, (ii) they may require lengthy tuning, (iii) are not suitable for non-linear and complex systems, and (iv) present significant performance decrease when uncertain/unpredicted factors are present. Additionally and related to the use of traditional control methods is the fact that the majority of ALSs proposed in

the literature are modelled based on linearised aircraft models. By doing so the longitudinal and lateral aircraft dynamics may be separated into two independent models, which allows the use of separate controllers for each aircraft mode of motion [55]. That means that the aircraft's equations of motion only represent the condition in which they have been linearised in. Hence, the performance of the controller decreases as the aircraft's dynamics move away from the nominal condition.

The automatic landing system design problem consists of controlling various states of the aircraft, making it a complex system. The variables related to each state must be carefully controlled in order to maintain the aircraft stable and comply with aerodynamic, structural and even regulatory requirements. To summarise, the challenges imposed to both reducing aircraft accidents, discussed in the introduction, and the development of better ALS control systems arise from two sources:

1. Unpredictable landing conditions: (i) *external disturbances* - severe turbulence and severe wind conditions - and (ii) *component failure* - actuator, engines, systems -, and
2. The *aircraft dynamics*: inherent aerodynamic characteristics, such as drag and lift changes due to speed and mass changes, or centre of gravity changes.

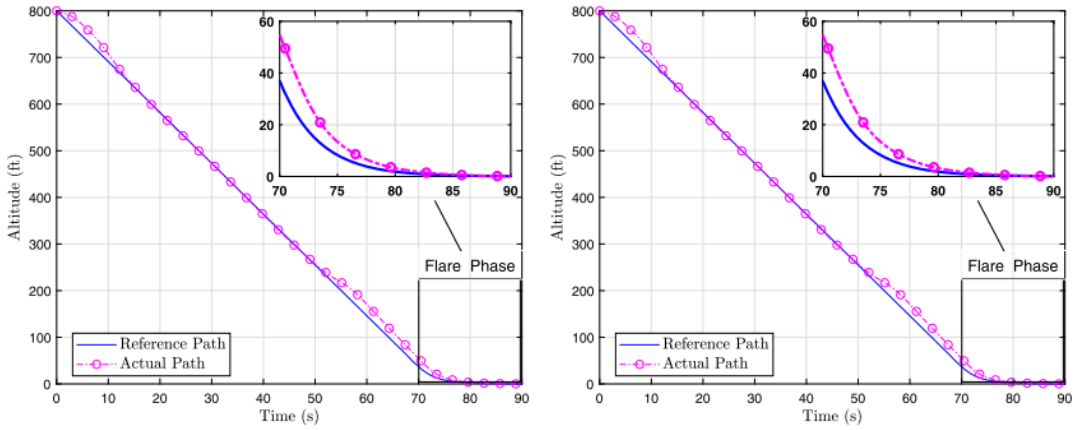
The majority of the development in ALS in recent years has been the improvements of tools and aiding systems, such as Distance Measuring Systems (DMEs), Global Navigation Satellite Systems (GNSSs), ILSs, etc. [43]. The advance in the ALS aiding tools allow for further development of ALSs themselves, through the use of modern control techniques, the use of newly developed systems. An example of this would be the use of a Global Positioning System (GPS), as of now GPS is not accurate enough to be used in high-precision systems such as ALS, however, recent developments in the field lead to believe that it will be sufficient in a close future [10]. According to [44], there has been little progress in the field of landing flight control using Neural Networks, Dynamic inversion, linear dynamic compensator, Platform Controller Hub (PCH) blocks, etc. This is one of the motivators for this thesis, which has as one of its goals to contribute to the development of enhanced control methods for ALSs. Regarding the control portion of automatic landing systems, the challenges to be overcome relate to the uncertainties related to the inherent characteristics of landing an aircraft. That means that it is necessary for future ALSs to be *adaptive* and as well as *robust*, in order to deal with the ALS issues, defined as: *Unpredicted disturbances* and *changing dynamics*, of both the aircraft and its surroundings.

3.1.3. Related Work

The previous section of this chapter defined the two main issues encountered in the development of automatic landing systems, in order to identify the aspects of ASL control that have to be improved in this thesis. Once this has been done it is important to analyse what has already been done and proposed, in practice and in literature, in an attempt to mitigate the identified issues, such that the research herein performed is not redundant and/or does not follows paths that have already been proven to be inefficient.

One of the most common methods found in the literature for ALSs involves the use of optimal control. The optimisation of a predefined cost function over a period of time through determining the control and state trajectories in order to find the control of a dynamical system is subject to a field known as **Optimal Control**. One of the most relevant works utilising this technique in ALSs has been proposed by [60], which uses a combination of H_2 and H_∞ control. In their work, H_2 control is used to determine the optimal trajectory, taking into account the predefined and the actual trajectories of the aircraft. Subsequently, H_∞ is used to minimise the effects of disturbances in the performance output. Their study, however, is limited to the longitudinal mode of the aircraft and is represented by a linearized model. These two facts make their system suffer from the same flaws observed in current ALS methods, that is, they are limited to certain flight envelopes and have decaying performance as the aircraft moves away from trimmed condition. [15] proposed a method that utilises H_∞ and stable inversion to provide robust control and accurate tracking. The results of their testing were positive, however, they only included longitudinal dynamics and the tracking was performed offline based on the desired trajectory. Having those characteristics in mind, [39] developed a landing system flight control based on H_2 that is fault-tolerant and takes into account wind disturbances. Their tests were performed using a high-fidelity fighter aircraft model and showed margin errors within 10 m lateral deviation and 5 m in the longitudinal case. However, the designed controller is still limited to linearised models of the aircraft. Yet another combination of robust H_∞ is proposed by [66], the paper described and achieved good results in the study of severe wind conditions during landing performance. Figure 3.4 shows

the performance of the developed controller under moderate downbursts. Once again, the paper only considers longitudinal dynamics in its tests and the controllers were only designed for the Flare phase of landing.



(a) Moderate downbursts **(b)** Severe downbursts

Figure 3.4: Glideslope descent and Flare paths, from [66]

Sliding Mode Control (SMC) is a nonlinear control technique that involves the use of discontinuous control signals to alter the controlled systems normal dynamics, control is achieved when the system is forced to ”slide” along a cross section of its normal behaviour. There have been some papers that utilise of this technique to be applied in ALSs, such as the works of [55] and [32]. In the first paper, SMC is used to control the aircraft control surfaces and throttle such that the errors of the actual path in relation to the desired path are close to zero. Tests were performed using the HARV (High Alpha Research Vehicle) aircraft model, in which the aircraft is left to perform landing at a position with a large offset from the correct flight path. The results showed that the designed controller presents higher performance if compared to classic PID controllers. However, there has been no tests under adverse conditions. The latter SMC method, on the other hand, focused its study on severe wind conditions and random actuator failure The authors developed a neural-aided SMC control method, which contains three different controllers. The tests presented in this paper showed improved performance compared to the performance of an earlier system designed by the authors.

Dynamic inversion has been successfully implemented as a model-based control law methodology that uses dynamic inversion in the F-35 fighter [27] [12]. It is the first time such a technique is used to enhance the performance qualities of an aircraft in such a wide range of speeds, from zero to supersonic, and activities, such as short take-off and vertical landing. This is the most notable application of dynamic inversion and the fact that it has been successfully applied to flight control corroborates the use of such technique in automatic landing systems.

Artificial Neural Networks or simply Neural Networks (NNs) is a bio-inspired technique which simulates the neurons and connectors existing in the brain of animals. It has been utilised and proved to be a powerful tool to approximate functions and identify patterns. Approximating functions is highly important in the field of control engineering, hence, it is not a surprise that the use of NNs has reached the field of autonomous landing systems. [34] took on a project to prove that NNs could significantly improve the performance of conventional landing systems. In the paper five different controller designs are tested against a conventional controller. The controllers were designed according to the following architectures: conventional back-propagation network (BPN) , Counter-Propagation Network (CPN), Improved BPN (IBPN), Radial Basis Function Network (RBFN), and Multilayer Functional-Link Network (MFLN). The results of the tests show that if properly trained and implemented, NNs can be of high value in ALSs. Figure 3.5 shows the performance of the controller trained faith the conventional BPN structure against the desired landing path. The simulations, however, only utilised a simplified model of a commercial aircraft and only simulated longitudinal and vertical movement. Furthermore, other NN base controllers have been proposed, [45] compares the performance of 4 different controllers, utilising Neural Networks, PID, a combination of both, and a combination of NNs and fuzzy logic. The simulations take wind into account and showed that classic PID has acceptable performance under

nominal conditions, but it does not under windy conditions. The neuro-PID and Neuro-Fuzzy controller, however, presented acceptable to good performance in nominal and windy conditions. The simulations were, once again, only performed in the longitudinal direction.

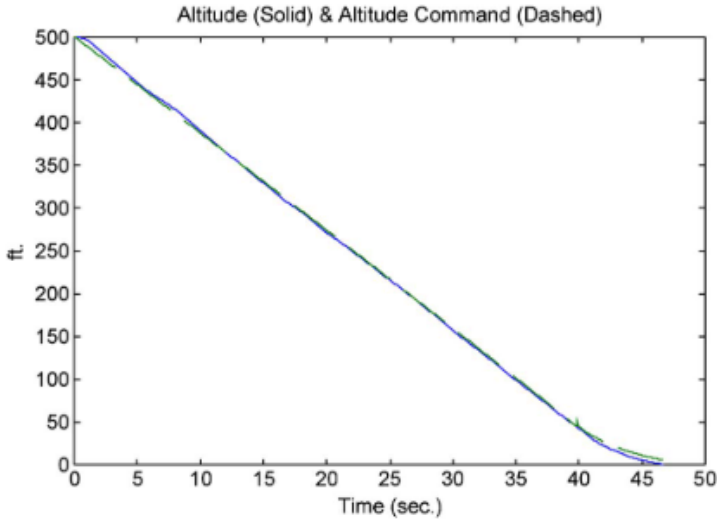


Figure 3.5: Glideslope and Flare paths of the BPN controller, retrieved from [34]

There are many more methods that have been proposed in literature, however, mentioning all of them and describing the results of each one of them would prove not viable to be added to this report, however, interesting. Hence, a general overview of control techniques, its advantages and disadvantages is presented in table 3.2. It is important to note that Gain scheduling is not a control method in itself, but an approach to control non-linear systems using linear controllers, however, it brings with it its own characteristics and it is often used applied to some of the methods presented in the table, hence it has been added to it.

Table 3.2: Overview of Control Methods and their characteristics

Control Method	Advantages	Disadvantages
Classic control (PID)	Widely used, simple implementation, reliable under nominal situation [66]	Lack robustness [66], require tuning, limited performance, do not handle: linearities, complex systems and uncertainties
Gain Scheduling*	Available for broad flight envelopes [67]	Additional extensive design process [67]
Optimal Control	Optimise a property of choice (cost function) [49], Rapid response, multivariable control	Limited to linearised aircraft models [55], inherently complex, limited performance in problems with multiple optima [49]
Robust Control	Allow for use in multi-variable system problems [66], static policy, allows for best achievable closeed-loop performance (within required robustness) [41]	Depend on model accuracy, limited performance in problems with multiple optima [49]
Sliding Mode Control	Explicit control law, does not require aircraft model linearisation [55], insensitive to disturbances and uncertainties [13], adapts to changing parameters [49]	Errors might not converge in finite-time, may suffer from singularities [13] High frequency commands [49]
Dynamic Inversion	Cancel non-linearities, closed loop behaves as a stable linear system, simplicity in control structure [44]	may inherently suffer from lack of robustness [13]
Backstepping	Can be used recursively	Require measuring of all state-variables, sensitive to parameter variation
Adaptive Control	Adapts to unknown environment [49], decrease non-linearities due parameter changes	Prone to instability, computationally involved [49], adaptation loops has to be slower than control loop, requires model of the system (?)
Neural Network	Approximate unknown system [44] (no exact model is required), possibly applied for feedforward control	Performance dependent on training set [15], require adjustment of large amount of parameters, stability is not guaranteed, essential to choose size and structure
Fuzzy Logic Control	Intuitive design, Exact model is not demanded	Uses trial and error for optimisation (might require long time), stability is not guaranteed, require several tuning parameters

* Gain scheduling is not exactly a method on its own, but a technique that is often used in control of non-linear systems, which comes with its own characteristics, hence it has been added to this table.

3.1.4. Conclusions

The Literature survey presented in this chapter has the aim of introducing the topic of Landing Systems and the complex task of performing them automatically. This chapter explained that landing can be divided into four phases, namely: glide-slope descent, flare, touchdown and roll-out, and that an ALS must be capable f performing all of those automatically and safely. Additionally, comprehension of landing systems allows for the understanding of which parameters play a role in the process and how they affect the aircraft, it also provides the basics for the design of a controller framework. Additionally, the content presented in this chapter is used to answer research sub-questions Q1.1/2/3 as well as Q1 itself.

Q1.1 What are the current methods utilised?

This question is partially covered in the introduction and it is part of the motivation for this thesis. The vast majority of automatic landing systems utilise classical control theory, i.e. PID controllers, or a modified version of it. The main reason for that is the fact that classical control methods are efficient, relatively simple to implement, and reliable. However, these methods in general lack robustness, often require lengthy tuning, are designed to be used in linear systems and not complex systems, and cannot handle uncertainties, amongst others. Therefore there have been a wide variety of landing systems that have been proposed. Section 3.1.3 presented some ALSs that use alternative control methods, such as Optimal Control, Sliding Mode Control, Dynamic Inversion, and Neural Networks. Additionally table 3.2 has been made containing control methods and their advantages and disadvantages. It is also observed that the majority of landing systems are only tested in longitudinal motion, leaving lateral motion aside. Many of them also utilise linearised models of aircraft for their tests. Both of these simplifications/assumptions are acceptable if the aircraft's attitude and movements do not deviate from the nominal condition, however, in reality, that might not be the case and exceptional circumstances are not such a rarity.

RQ1.2 Under what circumstances do these systems become faulty?

Section 3.1.2 presents the challenges faced in the current ALSs, and as stated before, the majority of autoland systems rely on classical control theory, which brings inherent issues, mentioned in the answer to Q1.1, to the final system. Additionally, the nature of the problem of automatic landing makes ALS a very complex system, which has to deal with changes in the environment and control of the various states of the aircraft. These factors can be summarised in two main challenges of ALS, that represent the situations where the systems become faulty: 1. Unpredictable landing conditions: (i) *disturbances*, severe turbulence and severe wind conditions, and (ii) component failure, actuator, engines, systems, etc., and 2. The *aircraft dynamics*, inherent aerodynamic characteristics.

RQ1.3 What are the requirements for such systems? (airfield, weather conditions, onboard equipment...)

Automatic landing systems heavily rely on the use of aiding systems that provide information to the aircraft. This is logical since the ALS has to have information about the aircraft's position, course and attitude in order to properly control it. There are three airfield and equipment conditions that are necessary. It is required that the airfield possesses an ILS, to provide lateral and vertical guidance, a Radio Altimeter, to provide accurate height above ground level, and a positioning system, in the past beacon markers were used to perform this function, nowadays most aircraft are equipped with DMEs, and in the future, it is expected that GNSSs will be accurate enough for the function. Additionally, international agencies have divided weather landing conditions into three categories, namely CATI, CATII and CATIII. Each of them is determined by its Runway Visual Range and has specific Decision Heights. These categories are used to determine whether the use of ILS is permitted or not and set several equipment and situation criteria.

3.2. Reinforcement Learning

Reinforcement learning is an assortment of methods that are based on gathering information to achieve a certain goal through interaction with the environment. This concept can be applied to multiple areas and benefits from the contribution of research in a variety of fields, see figure 3.6, most important for this thesis are two of them. The first is that of *learning*, more specifically, the branch of study within Artificial Intelligence (AI) called Machine Learning (ML), that is dedicated to understanding *learning methods*, that is, methods that use data and adapt it to maximum advantage and increase performance in a certain task. This branch of RL is important because it provides the widest set of algorithms [11]. In a simplistic way, ML algorithms develop a function that maps inputs and outputs from a data set. Reinforcement Learning, however, is a special case of ML, where the algorithm is trained using a data set, but from their own experience, that is, they *learn* what works and what does not through a trial and error processes. The second is more specific to the topic in this research, which is the field of *optimal control*, which revolves around the idea of finding solutions and methods to control a (dynamical) system through optimising a *cost function*. Reinforcement Learning is a paradigm that is well-suited for the identification of control methods in highly dynamic systems; As further described in this chapter, this is due to the inherent relations and similarity that it has with controllers.

This chapter focuses on introducing the fundamentals of Reinforcement Learning, in section 3.2.1, outlining the approaches and classification of RL algorithms, in section 3.2.2 which also includes RL methods applied to flight control, in section 3.2.2, and a description of the requirements and the characteristics that are desired in the algorithm to be used further in this thesis project in section **??**. Section 3.2.3 and 3.2.4 introduce two families of state-of-the-art algorithms, it briefly describes each of the most prominent algorithms of each family outlining its characteristics, strengths and weaknesses. Section 3.2.5 delves into the topic of adaptive and robust control with the goal of proposing a framework to be used further in this project. Finally, in conclusion to this chapter, a brief summary of its findings is given in section 3.2.6, and the findings of the chapter are used to answer research questions *RQ2.1* and *RQ2.2*.

3.2.1. Reinforcement Learning Basics

RL differs from other ML paradigms, such as *Supervised learning* and *Unsupervised learning*, in terms of its goals and applications. In the former algorithms learn to predict values and classes based on a labelled data set, whilst in the latter algorithms focus on identifying patterns and clustering an unlabeled data set, *Reinforcement learning* deals with finding (optimal) behaviour in a given environment, based on the environment's feedback. There are multiple uses and applications for Reinforcement Learning, this is due to the broad and general concept of the RL problem. This makes an overlap between multiple fields and the use of RL. In this thesis, however, the main concern is the application of Reinforcement Learning in the engineering field, more specifically in the sub-field of Control engineering. By reason of completeness, figure 3.6 presents a ven-diagram containing the fields in which RL is influential and the term given to its overlapping correlations.

The following subsections describe the elements and sub-elements that play a role in the RL problem formulation, and the fundamental concepts and how they are used to expand the RL paradigm.

Agent-Environment Interaction

To its bare-bones Reinforcement Learning is the process of learning from interactive experiences, inspired by how animals learn through the evaluation of the consequences of their actions to their surroundings. To elucidate and construct a formal definition, the entities in the problem at hand are labelled the **agent** and the **environment**. The former is the *decision-maker*, or the *controller*, and is charged with the task of deciding what actions to take, it is also the agent of the *learner* in this problem. The latter is also called a *plant* and it is simply the environs surrounding the agent. It is characterised as *everything else* other than the agent, including the system in which the agent finds itself in.

In the reinforcement learning problem, it is the agent's task to learn how to *behave* such that the environment is altered and reaches a desired state. The agent-environment interaction occurs by means of **actions** that the agents take to manipulate the environment. In turn, the environment-agent interaction occurs through observations, or **states**, and **rewards**. The agent's *decision* of what actions to take is a conjunction of two factors, the interpretation of the environment's situation (*state*) and the feedback that the environment gives upon each action of the agent (*reward*). It is important to note that rewards can be positive, in case the agent takes a 'good' action, and negative, in case the agent takes

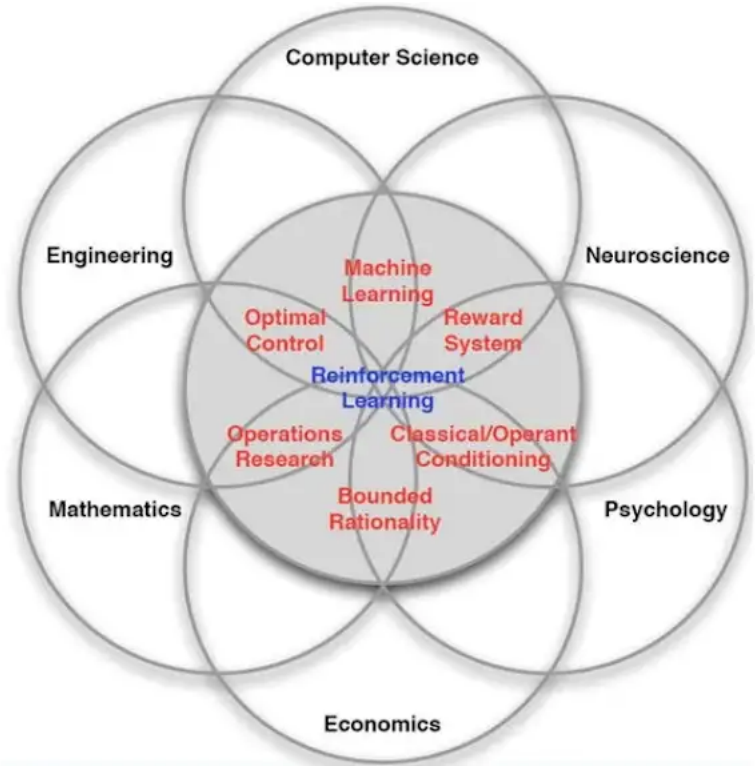


Figure 3.6: Ven-diagram representing the fields of Science where there is RL contribution, retrieved from [26]

a 'bad' action. The concept of 'good' and 'bad' has nuances, it is not necessarily a binary problem and specific reward functions are designed for each problem such to clarify this definition. As discussed in section 3.2.2, it is a difficult task to translate the goals of the agent to a mathematical function, and it is the root of many problems. A diagram exhibiting the reinforcement learning problem and the interaction between its elements can be seen in figure 3.7.

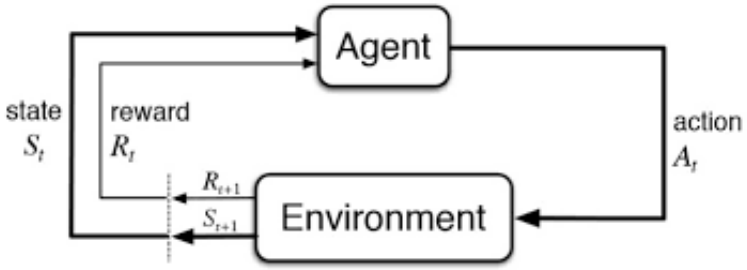


Figure 3.7: Reinforcement Learning Diagram, retrieved from [63]

The Reinforcement Learning problem can be analogous to the problem studied in the field of control theory. Comparing figures 3.7 and 3.8 it can be seen that the terms agent, environment and rewards states can be analogous to the terms *controller*, *controlled system* (or *plant*) and *control signal* (or *sensor signal*), respectively, from the field of engineering. As shown in figure 3.6 reinforcement learning is utilised in a wide variety of fields and the analogy herein made between the RL and the control problem can be made in the other fields as well. Hence, for the sake of consistency amongst the different fields, RL adopts a more general terminology for its elements and processes such that it is meaningful and accessible to a wider audience. Nonetheless, reinforcement learning is a paradigm that is well-suited for control applications in highly interactive and dynamic environments.

Additionally, it is important to note the difference between observations and states. Although it is not the case, in literature these terms are most often used interchangeably. Whilst states need to contain

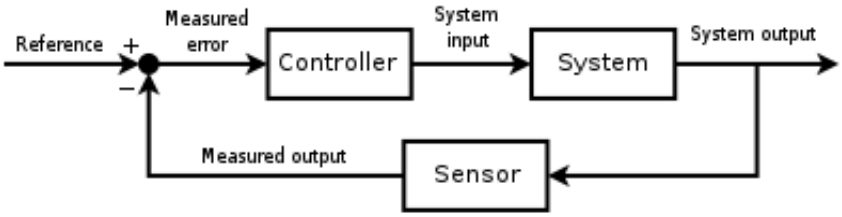


Figure 3.8: Feedback-loop, analogous to the control problem, retrieved from [16]

all available information, observations only contain the sub-section of the information that is available to the agent. To be considered true states, observations must comply with the *Markov Property*, which is discussed further in the document. However, it suffices to understand that oftentimes observations contain enough information, i.e. are considered *almost* Markov, and hence are sufficient to solve the reinforcement learning problem. This distinction is made here because it in certain cases observations do not hold enough information to support a classic RL system, and hence need adaptations.

Markov Decision Process

The concepts presented in the previous paragraphs represent the theoretical idea of reinforcement learning, or one interpretation of the learning process as it happens for animals, including humans, through experience. In order to mathematically solve this problem it is necessary to introduce a formal definition for it. The most common formalisation utilised in the RL world is the **Markov Decision Process (MDP)**. Note that this process is not the only formalisation and it does not fit all RL problems, however, it describes the vast majority of modern reinforcement learning, hence it is the one presented in this document. In short, the reinforcement learning problems can be described as *finding the best sequence of actions that result in the optimal outcome*, that is, that generates the highest cumulative reward. The MDP has its origins in the field of optimal control and it is defined as a discrete-time stochastic control process for sequential decision-making. According to [63], the world of reinforcement learning has much to thank for the concept of Markov decision processes.

The Markov decision process is defined using four elements: a **state**, an **action**, a **reward** and a **probability**, whose mathematical representations are S, A, R, and P, respectively. If the state and action spaces (the set of all possible states, or actions) are finite, the process is referred to as a *finite MDP*, which is the most important case for understanding the theory of reinforcement learning. In an MDP the agent interacts with the environment in sequential discrete time steps $(t_1, t_2, t_3, ...)$. In each time step, the agent receives information about the environment's *state* S_t, the agent ponders the situation and takes an *action* A_t based on the state. The action taken alters the environment, hence in the next step, the environment is in a different state. The new state S_{t+1} is sent to the agent, along with information about the *reward* R_{t+1} as a form of a feedback of action A_t, which shapes the next action A_{t+1}. The process occurs sequentially, generating a series of states, actions and rewards, represented by $S_0, A_0, R_0, S_1, A_1, R_1, ...$; this set of data is denominated the *trajectory*. A new state-action pair is the result of previous state-action pairs, and the *probability* of either of them happening is given in equation 3.2. The probability of the agent moving from a specific state to another is named the *transition probability*.

$$P(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\} \tag{3.2}$$

Additionally, it is assumed that all MDPs follow the *Markov Property*, which states that *"future is independent of the past, given the present"*. The Markov property implies that the present state of the environment holds enough information about its past that the future can be predicted without the need for information about its past, hence it can be described as memoryless. A state S is said to be Markov if and only if

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1, ..., S_t) \tag{3.3}$$

With the exception of most board games and certain physics problems, real-life situations rarely comply with the Markov property. Although, problems are often *nearly* Markov, hence state signals should be chosen or constructed in such a way that they characterise a Markov property as much as

possible, which oftentimes is sufficient. The equations above and the explanations herein presented consider a *finite MDP*, this is done to simplify representations, but can easily be adapted to continuous MDPs.

According to [11] MDPs, and by default reinforcement learning, have two big advantages when applied to control systems:

1. Generality - MDPs are able to deal with nonlinear and stochastic systems, and also can use non-quadratic reward functions.
2. Model-independence - MDPs can be model-free, meaning that a model of the system's dynamics is not always necessary.

Additionally, [11] also mentions that the downside of classic MDPs is the fact that they work in discrete time, which is non-realistic. However, this problem can be easily mitigated when MDPs are paired to function approximators, considering that powerful approximators have already been developed, this problem can be mitigated. The use of approximators is necessary and plays an important role in this project, hence will be discussed in greater depth further in the report.

Return, Value functions and Policy

In the RL problem, the agent's goal is to maximise the amount of reward it can get, which means not only the immediate reward but also the long-term reward. Hence it has to find a *rule* or *strategy* that allows the agent to choose the actions such that the reward is maximised in the long term. In reinforcement learning terms, the cumulative reward is represented by equation 3.4, where G_t is the cumulative reward at time step t, r is the immediate reward, and γ is a discount factor. The discount factor is included in the formulation in order to balance the future and the immediate rewards, and its value varies from 0 to 1 it determines how relevant future rewards are, and by default, how much they influence the current decisions of the agent. This factor allows the designer to have more, or less, control over the agent's behaviour. For example, when γ is null, the agent will only concern itself with immediate rewards and will only take *greedy* actions.

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = r_{t+1} + G_{t+1} \tag{3.4}$$

The law that rules the agent's decisions is referred to as the **policy**. It is mathematically represented by π and it is the strategy that the agent employs when taking decisions and it is essentially mapping from state to action. In an MDP different policies in the same situations will have different transition probabilities. It is important to note that policies can be *deterministic* or *stochastic*, deterministic policies directly map states to actions, whilst stochastic policies define the probability of each possible action for each state. Hence a deterministic policy can be seen as a stochastic policy that assigns a probability of 1 to one only possible action in each state.

Furthermore, almost all reinforcement learning algorithms make use of **value functions**. These are functions that determine how advantageous it is for an agent to be in a particular state - in case of **state-value functions**, or how advantageous it is for an agent to take a certain action in a certain state - in case of **(state-) action-value functions**. The notion of benefit, in this case, is determined by the expected future rewards, or returns, and varies depending on the policy being followed. Hence, value functions are mathematically defined as functions of the policy, and are formalised in equation 3.5 for an MDP. In this case, a state-value function defined as v_π, and it is the expected return when starting in state s following policy π, and the cumulative reward G_t is defined in equation 3.4.

$$v_\pi = \mathbb{E}\{G_t | S_t = s\} \tag{3.5}$$

Similarly the state-action function $q_\pi(s, a)$ under policy π is defined in equation 3.6, and it has similar working as the state-value function, only with the additional influence of the action on the return value.

$$q_\pi(s, a) = \mathbb{E}\{G_t | S_t = s, A_t = a\} \tag{3.6}$$

Additionally, some reinforcement algorithms make use of **advantage functions** which is defined as the difference between the state-value function and the state-action function. This function is presented in equation 3.19, it is used in an important array of algorithms.

$$A_\pi(s, a) = q_\pi(s, a) - v_\pi \tag{3.7}$$

The Bellman Equation & Optimality

The goal of an MDP is to take the best actions to maximise the long-term reward, e.g. the *value function*. Assuming that all the rewards for each state and actions are already known, reaching the maximum value is only a matter of taking the actions that result in the highest rewards, this would characterise the *policy*. However, in reality, the rewards for each state are usually unknown; This is when the **Bellman equation** becomes important because it defines a necessary condition for optimality. The Bellman equation reformulates value functions into a recursive form, by doing so the reward of a state is defined as the possible rewards of future states. The recursive nature of the Bellman equation allows the rewards to be initially unknown, such that through recursive trial and error, it converges to the real values. The Bellman equation is described as:

$$Q(s,a) = r(s,a) + \gamma max_a Q(s',a) \tag{3.8}$$

Where Q is the (Quality) Q-Value function, which represents the cumulative reward form being in state S and taking action A. The equation is formulated as the *immediate* reward for taking an action, or r, added to the highest possible reward of being in state S' after taking action A, or $max_a Q$. The latter part of the equation is multiplied by the *discount factor* γ which is a fixed variable that determines the relevance of long-term rewards; this value might change from problem to problem.

Summarising, the Bellman equation is a tool that can be used to solve problems formalised by the Markov Decision Process, such as many of the reinforcement learning problems. This process is done through finding **optimal policies** and **value functions**, using the Bellman equation, in order to *maximise of long-term rewards*.

Exploration & Exploitation

There is a paradigm in the context presented above related to how actions are chosen. The goal of MDPs is to maximise the cumulative rewards, therefore the agent is inclined to only choose actions that it knows result in good/positive and high rewards. However, if placed in an unknown environment, the agent does not know *all the possible* rewards, and sometimes it must take ”bad” actions in order to make ”batter” states available, and only then being able to maximise the rewards. Therefore, there must be a *trade-off* between **exploration** and **exploitation**, such that the agent can accumulate rewards by taking actions that it knows to be effective, but it also must select actions that it has not taken before in order to discover which are the most effective actions, and states.

The agent is never able to fully exploit the environment if it does not explore it. However, it should progressively switch its focus from high exploration in the initial stages of training to high exploitation towards the end of the training process.

Design Choices

Reinforcement Learning is a wide field which embraces a large variety of methods with their own characteristics. However, there are high-level concepts that can be used to group algorithms together as well as aid algorithm selection through design choices. These choices relate to model dependency, the manner in which the algorithm updates its policy, whether learning happens in real-time or not and what functions it estimates.

Model-based vs Model-free

At the highest level, RL algorithms can be separated in *model-based* and *model-free*, this distinction is made based on whether an algorithm requires a model of the environment or not to function. Contrary to intuition a model-based algorithm does not necessarily require that the model of the environment is known in advance. This leads to another distinction among model-based algorithms, which is between those that *learn the model* and those that require a *given model*.

The main advantage of model-based methods is that, once the model has been given/learned the agent is able to plan ahead by predicting the future and analysing the possible results of different actions [63]. Model-based methods are also extremely sample efficient allowing for *online* implementation. The downside of model-based algorithms is that often times their agent is only as good as the model it uses, that is because models of the environment are merely models and they contain flaws to varying degrees, this means that frequently model-based algorithms might perform well with respect to its model, but poorly in the real world. Additionally, model learning can be extremely difficult and might require intense and lengthy efforts without any certainty of pay-off.

Model-free methods are those that do not require any sort of model or estimation of the environment it is in. These methods do not even try to predict how the environment will respond to their actions [63]. The main advantage of such methods is the fact that it does not require a model, this can be very useful in situations where the environment is not known or rapidly changes (highly dynamic). Model-free methods tend to be easier to implement and tune, however, forego the potential sample efficiency that its model-based cousin presents, hence are mainly applied in *Offline* manner.

On-Policy vs Off-policy

Another distinction that can be made is regarding how the agent in a reinforcement learning problem uses its policies. Essentially *On-policy* methods utilise the responses from the environment to evaluate or improve the policy that was followed when generating those responses, in other words, the policy that is used to make decisions is updated. Hence, the same policy is used for exploration and exploitation. *Off-policy* methods are those that utilise the responses from the environment to evaluate or improve the policy gathered from other (or previous) policies; in other words, the policy updated is different from the ones used to collect the data. Hence, different policies are used to generate the episodes. Off-policy methods are more flexible since there is a decoupling between the learning process and the agent's behaviour (policy). However, the possible wide number of collected data and its dissimilarities might result in high variance and slow convergence. Additionally, according to [51] off-policy methods can use deterministic policies (due to the use of different policies), but are unstable when paired with function approximators; Whilst on-policy methods are stable when using function approximators, they require stochastic policies (due the dual use of a single policy). On-policy methods are usually simpler and present less variance compared to off-policy, which on the other hand present higher levels of exploration.

Online vs Offline

Online learning happens in real-time, which means that the learning process happens as data becomes available, hence learning happens incrementally as more data arrives. This type of learning is well suited for situations where data is fed to the agents in a continuous flow and the agent must adapt to rapidly changing conditions, such as those encountered in highly dynamic environments. Algorithms that can learn in an online fashion are often times said to belong to the class of *adaptive control*. Online learning is usually reserved for algorithms that have high sample efficiency. On the other hand, in *offline learning*, the agent is trained with a predetermined data set/simulation, that is in batches of samples. Once it has been placed in the real environment its behaviour does not change, independently of its experiences and the data it collects. Therefore, offline learning is, in a sense, the direct opposite of online learning. Algorithms that can learn in an offline fashion are often times said to belong to the class of *robust control*.

Value-based vs Policy-based

Another worthy distinction that can be made is between *value-based* and *policy-based* methods. The former operates by estimating and optimising the quality of states that the agent finds itself and/or actions the agent (might) take. The policy to be employed then is derived from a given state/action. In this case, exploration is an additional step that is not embedded in the method itself, since it is naturally greedy and tends to make actions known to be good. Value-based techniques are said to be *indirect* because they derive the optimal policy from choosing the optimal state/action value. *Policy-based* methods, on the other hand, learn the stochastic policy function that maps the state to action. In simple terms, it samples the policy from the inputs and outputs of the environment. In this technique, exploration is included in the process and therefore does not require an add-on. Policy-based methods are said to be *direct* since they directly look for the optimal policy [7].

Taxonomy

Finally, after discussing the aspects of reinforcement learning in this section it is possible to create a basic taxonomy of reinforcement learning algorithms, organised in the diagram in figure 3.9. Note that this taxonomy only contains basic information and high-level characteristics. Due to RL being an extremely large field in itself that encloses a number of other fields, as seen in figure 3.6, too many distinctions can be made and it falls outside of the scope of this project to analyse all and each one of them, therefore a basic taxonomy is given in this section and it is expanded in later sections as more specific branches of RL are explored. This characteristic distinction is done to clarify RL algorithms and to understand where they fit in relation to each other, this allows us to (i) highlight fundamental design choices, (ii) expose trade-offs, and (iii) place algorithms into context. The distinction concept of gradient optimisation that branches off from Policy-based methods has not been discussed in this

section, but in the section 3.2.2.

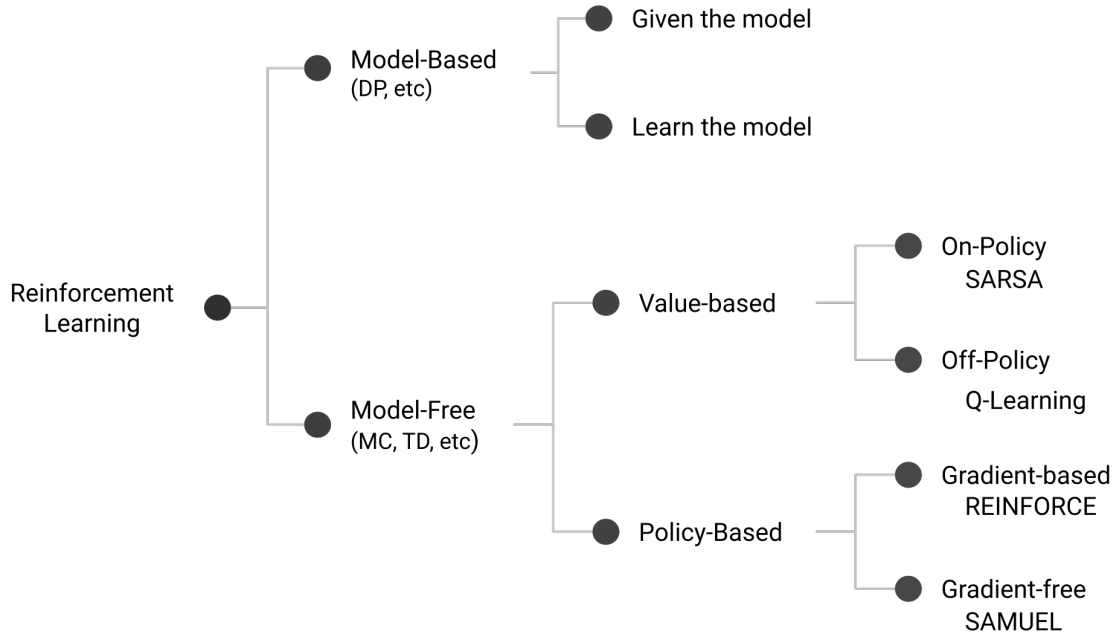


Figure 3.9: Basic Reinforcement Learning taxonomy

3.2.2. Reinforcement Learning Approaches

Reinforcement Learning encompasses a large variety of algorithms, this is due to the complex nature of the problem and the fact that there are no clear boundaries between one method and the next. That is to say that they somewhat blend into each other and can be thought of as a spectrum (see figure 3.10), where there are areas delegated to certain methods, however, a specific algorithm in a given category might flirt with characteristics and techniques present in other methods. Section 3.2.2 delves into some of the most common solution approaches, these are at the basis of reinforcement learning. Section 3.2.2 discusses aspects and measures that must be taken into account when applying reinforcement learning to continuous spaces, this is important because common RL approaches, presented in the previous subsection, are based on discrete time only. Section 3.2.2 describes the challenges in the field of reinforcement learning and how some of them are dealt with. Section 3.2.2 briefly describes some of the papers that use state-of-the-art reinforcement learning algorithms in flight control applications. Section 3.2.2 discourses over the topics that have been touched upon previously in the chapter in order to narrow down the families of algorithms that may be suitable for this project.

Common Solution methods

A common method of dividing RL algorithms is based on their update rule method. [63] identifies three general classes of approaches to the solution methods of the reinforcement learning problem, namely: *Dynamic Programming (DP), Monte Carlo (MC) methods* and *Temporal Difference (TD)*. It should be noted that these are not the only approaches available and existent, however, these are considered fundamental frameworks. Additionally, the classical version of these methods considers discrete state and action spaces, which is not the case for the majority of real-world problems. However, they are still mentioned because they contain the theoretical foundation for understanding and developing continuous space methods as adaptations of them circumvent these issues and are widely employed.

Dynamic Programming

The Bellman equation presented previously is the central result of R. Bellman's *dynamic programming* created in the 1940s and further developed in the 1950s. DP methods are particularly useful in solving discrete control problems, however, in their classical form assume a perfect model as an MDP and can be computationally expensive. These characteristics limit their usefulness in RL problems,

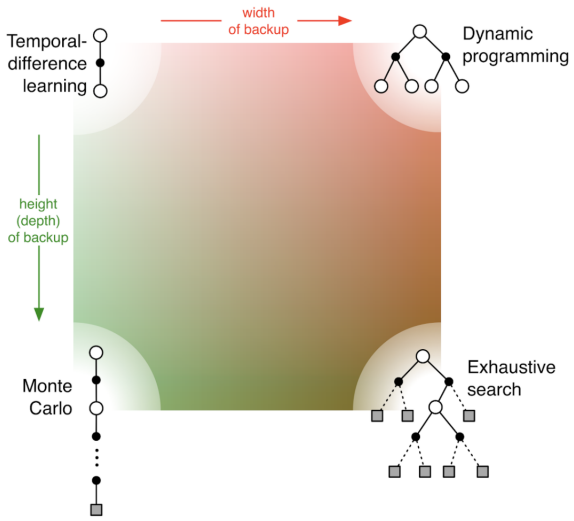


Figure 3.10: Reinforcement Learning spectrum, retrieved from [63]

nonetheless, they can be adapted to continuous state-action spaces, and serve as the foundation for understanding more complex methods.

As mentioned previously, optimal policies can be easily computed once the optimal value functions have been found, hence the usefulness of DP methods. The goal of Dynamic Programming in reinforcement learning is to optimise the value function, either the state or the action value functions, v_π pr q_π, respectively. Dynamic programming can be used to solve problems that satisfy two criteria:

1. Substructure: a problem has to be divided into sub-problems.
2. Overlapping sub-problems: in DP the solution for a specific sub-problem can be used to solve another sub-problem.

In MDPs both criteria are satisfied by the use of the Bellman equation, where the value of a state action takes into account the value of the next state. Hence, the value at a state is a combination of the previous states and the value of the next state is partially calculated in the current state. Two of the most common methods of DP in RL are *Policy Iteration* and *Value Iteration*. These methods are model-based, require the knowledge of the dynamics, and work offline, require the knowledge of the reward function.

The **Policy Iteration (PI)** method consists of two main steps that occur in every iteration, namely: *policy evaluation* and *policy improvement*. The interaction between the two is called Generalised Policy Iteration (GPI), and fundamentally refers to the idea of simultaneously maintaining an approximate policy and an approximate value function. Firstly, in each iteration, the value function, defined with the Bellman equation 3.9, is successively evaluated such that it represents more accurately the current policy. This is the *evaluation* or *prediction* step.

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{3.9}$$

Secondly, following the evaluation, a new *greedy policy* is found which determines the best action to be taken is the one that returns the most rewards in the shirt term, according to the value function of the previous policy. Hence, this is called the *improvement* step, and it characterises a one-step ahead approach, and it is expressed through equation 3.10. By assuming that the evaluation step is perfect, the algorithm works on the presupposition that each new policy is an improvement regarding the previous, unless it is already optimal, which then indicates convergence.

$$\pi(s) = argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{3.10}$$

The **Value Iteration (VI)** is, from a certain point of view, an improvement from Policy Iteration. This is because in PI the evaluation process repeats itself until convergence, however, exact convergence

in intermediate steps are not always necessary and may result in greater computational effort and time. Several algorithms have been designed to truncate the evaluation step such that convergence is still guaranteed, value iteration does this by stopping policy evaluation after a single update (one backup of each state). Equation 3.11 combines the truncated policy evaluation and policy improvement in a single update step.

$$v_{k+1}(s) = max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \tag{3.11}$$

Monte Carlo Methods

Monte Carlo methods refer to a broad class of algorithms that strongly rely on randomness and probability distributions to obtain numerical results. In Reinforcement Learning the term Monte Carlo is used to distinguish solution methods that rely on finding *optimal policies*. Essentially, MC methods revolve around the idea of running a series of several trials starting at a specific state in order to observe how much reward is possibly collected across all possible actions to be taken thenceforth. The value of that state is then calculated as the average of the total rewards obtained across all trials performed. This represents the *evaluation* or *prediction* step of the process. Similarly to DP methods, MC methods also utilise the concept of GPI to simulate the interaction between the evaluation and improvement steps. The *improvement* step is then done by updating the policy with the use of a *greedy* value function. The difference between policy improvement in DP and in MC is that in the latter the evaluation step occurs on an episode-to-episode basis.

At the end of every episode, the MC agent evaluates how well it performed through the total rewards accumulated. The agent's actions however depend on the *expected future rewards*, which is estimated by adding increments to previous estimations of the future rewards; Equation 3.12 shows how these increments are computed using the total rewards accumulated G_t, expected future rewards $V(s)$, and a discount factor α to control learning rate.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{3.12}$$

Additionally, the estimation in Monte Carlo methods might occur on a *first-visit* basis or on a *every-visit* basis. In this case, each occurrence of state S within an episode is called a *visit*. **first-visit** methods are those whose estimation of value function is done as an average of the returned rewards of every first visit to each state S. On the other hand, in **every-visit** methods the average is calculated following *all* visits to each state. Due to the nature of the solution, every-visit MC naturally extends towards function approximation and eligibility traces.

The use of Monte Carlo methods introduces the *learning* element of Reinforcement Learning, as here the agent *learns* value functions from returns from experience interacting with the environment. This is different from DP methods, where the agent *computes* the value function from the knowledge it already has of the MDP. This means that Monte Carlo methods do not require knowledge of the environment, meaning that they are model-free methods. Additionally, unlike DP and TD, MC does not make use of bootstrapping, i.e. the use of estimates in the update of values, rather than an exact value. A downside is that *exploration* is complicated in MC methods, this is because choosing greedy actions happens too often, as the policy is updated on less frequent bases.

Temporal Difference

Temporal Difference is a class of learning methods that are centred around the idea of comparing temporally successive predictions, in other words, the comparison of states at different points in time. And it is considered the single most innovative idea in Reinforcement Learning by [63]. Just as the previous two methods, TD uses the concept of GPI to solve its control problem, i.e. finding the optimal policy through the estimation of the optimal value function for a given policy. As it can be inferred from the initial phase of this paragraph, Temporal Difference is an amalgamation of Dynamic Programming and Monte Carlo methods with the aim of combining the best of both methods.

From MC methods, TD inherits the *policy evaluation* step, where it *learns* the value functions through the interaction with the environment, hence does not require knowledge of the environment's dynamics. Albeit, unlike MC methods, TD methods do not require that the episode reaches an end to update its state values, instead it does so after each time step. From DP, TD inherits the bootstrapping characteristic. Here updates are computed using existing estimated returns known as *targets*, instead of relying solely on complete returns with actual rewards. These targets or estimated returns have the general

form of $R_{t+1} + \gamma V(S_{t+1})$. This is best seen in equation 3.13, which is used in the **TD(0)** method, the simplest temporal difference method. To summarise the relation between TD, DP and MC it can be said that TD is naturally implemented as an *online* and it is *fully incremental*. Equation 3.13 shows how TD methods eliminate the need for the episode to end before the accumulated rewards are computed by introducing the TD target $R_{t+1} + \gamma V(s_{t+1})$ at the end of every step. Where R_{t+1} is the immediate reward at step t_1 and $\gamma V(s_{t+1})$ the discounted immediate value.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{3.13}$$

Referring back to the *on-policy* and *off-policy* concepts from the previous section, a complication regarding maintaining sufficient exploration arises in TD methods. Hence, TD methods can be separated regarding how they treat their policies. The difference between the two has been explained in section 3.2.2, to avoid redundancy and for the sake of completeness, this section only presents the value estimation equation of two distinct methods. Equation 3.14 shows the value-estimation method for the algorithm known as **SARSA**, it can be seen that the TD target utilises the state and action of the following step, but still uses the current value function $Q(S_{t+1}, A_{t+1})$. Hence, characterising an on-policy algorithm.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})) - Q(S_t, A_t)] \tag{3.14}$$

On the other hand, equation 3.15 shows the value-estimation method utilised in the **Q-learning** algorithm, an off-policy solution. It can be seen that the TD target utilises the state of the following step, but picks the best action that will result in the maximum state-action value $maxQ(S_{t+1}, A^*)$. It is important to note that the chosen action is the best at that moment, in the next step, that same action might not result in maximal values.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, A_{t+1})) - Q(S_t, A_t)] \tag{3.15}$$

RL in Continuous Spaces

The methods described in 3.2.2 have been proven to be very useful and span a wide range of applications within Reinforcement Learning. These traditional methods were elaborated based on *finite Markov Decision Processes*, hence only suitable to discrete-time problems that contain a limited amount of state action spaces. In these cases, the values in the state and/or action functions can be stored and updated in tabular form, such that the agent has an overview of all gathered information. These methods are said to be *tabular* and represent exact value functions and exact policies, an example of such an algorithm is the previously mentioned *Q-learning*.

In real life, however, this is not always the case; aerospace systems have to deal with continuous variables such as attitude angles, rotational rates, and positions. Therefore the classical reinforcement learning solutions presented in the previous section are not sufficient to create a suitable control system. A solution would be to transform such variables from continuous time to the discrete-time, but then two problems arise: (i) information is lost in the discretisation process, and (ii) sufficiently accurate representation of this information in exact form would require an enormous amount of data, increasing computational and memory requirements, as well as efficiency. A better solution is to employ *function approximators* to generalise the value functions.

Function Approximation

Function approximators are strong tools utilised in *machine learning* to create functions that approximate certain data or a more complex function. In the reinforcement learning problem, such functions can be used to estimate value functions to eliminate the need for exact values. The search for optimal value functions can be represented by equation 3.16, where the left-hand side is the approximate state-value function in terms of the state, and the right-hand side is the real (optimal) state-value function plus an approximation error. By the nature of this action, function approximators treat the *prediction problem* or RL since it estimates the (optimal) value function.

$$V(s_t) = V*(s_t) + r(s_t) \tag{3.16}$$

Since function approximators play such a crucial role in reinforcement learning, it is applicable to briefly explain some of its design choices. There are three levels in which approximators can be differentiated: (i) parametrisation, (ii) linearity, and (iii) optimisation. It is important to note that these distinctions are not mutually exclusive and that different combinations are possible.

Firstly, there are **parametric** and **non-parametric** approximators, the former utilises a finite set of parameters which are manipulated in such a way that the function approximator matches the observed data as much as possible. The downside of these methods is their need of specifying certain parameters before they can be applied in function estimation. The most common parametric method in recent times are Neural Networks (NNs) but also include methods such as linear basis functions [37]. The latter on the other hand does not require such an additional step. Despite this obvious disadvantage of parametric approximators, these are still the most common methods utilised in RL, this is because they are designed for independently and identically distributed data and usually RL generated data do not fit either of those two requirements. Additionally, non-parametric methods are memory-based, hindering computational efficiency and requiring large amounts of stored data [71]. The most notable examples of non-parametric methods are Gaussian and Kernel regression, and regression tree algorithms.

Secondly, there is a distinction between **linear** and **non-linear** approximators. The former is well understood by researchers and has been widely applied to a variety of problems for a long time. The advantages of such methods are the fact that they are simple implementation, fast computation and have a strong guarantee of convergence. On the other hand, their disadvantages are their limited accuracy and the fact that they require knowledge of the domain, as useful features must be identified previous to implementation. Common linear methods use polynomials, Fourier-series and tile coding. Non-linear approximators are less understood and as a result, have fewer guarantees of convergence. However, they may approximate an unknown function with more accuracy than linear approximators given the same initial information, in some cases not even requiring pre-defined features. Additionally, promising empirical results have been obtained in experiments combining reinforcement learning techniques and Neural Networks (NN) [71].

Finally, there is also a distinction between **gradient-based** and **gradient-free** optimisation. The former uses derivatives to find the optimum values of a function, which are then utilised to approximate this function through reconstruction. Gradient-based methods generally follow three steps, (i) *search direction*, where the derivative, or gradient, is calculated and its direction estimated, then (ii) a *step* is taken in the direction of the gradient, lastly (iii) it is *evaluated* whether a maximum or a minimum has been reached. These methods are widely used, fast, easy to scale, and have been proven to guarantee conversion, however, are susceptible to finding local optima and require the approximated function to be differentiable. Gradient-based methods have been successfully used in a number of applications in combination with Neural Networks, which are inherently differentiable. Gradient-free, on the other hand, represents a wider group of methods that do not utilise derivatives, hence these are useful when the function to be approximated cannot be differentiated. These are very flexible methods and have a wider range of applications, however, are less utilised because they are much slower and do not guarantee optimal solutions.

Agent-Structures

The distinction between the different types of approximators made up until this point only concerns the approximators themselves, leaving aside the functions to be approximated themselves. In general, the goal of reinforcement learning in control systems is to find an optimal policy (*control problem*), which is obtained through the estimation of the optimal value-functions (*prediction problem*), which in turn depends on the environment and its model. Hence all these three elements overlap with each other in the reinforcement learning problem. Additionally, *policy, value* and *model* can be approximated as well, resulting in general methodologies and solving approaches.

Model approximation methods are those that approximate the RL problem as an estimation of its MDP and compute the optimal policy from it. In general, terms, approximating the model of the environment is not an easy task, however, often times it is simpler than learning the value of a policy or directly optimising the policy. These functions are only dependent on current and local data, this is because it is assumed that the problem is completely Markovian, satisfying the Markov property. Although approximating the model of the environment eases the optimisation process, it is no guarantee that the resulting policy is better than the one that can be found simply from samples, this is because models are oftentimes not accurate enough.

Value approximation methods are those that aim to solve the reinforcement learning problem through the estimation of the value functions, which is used to give an approximation of the current or optimal policy. In this category, the distinction between *on-policy* and *off-policy* plays an important role, as well as the distinction between *behaviour policy* and *update policy*, since function approximation can be used to estimate the former, the latter, or both. Furthermore, these can be applied to both *online*,

offline or a combination of both of them. Due to its ease of manipulation, such methods are highly present in reinforcement learning. However, it has limited use in continuous spaces, this is because whilst estimating a value function in continuous spaces is not a complicated task, finding the optimal, *greedy*, action requires a search through the entire action space, which is impractical in continuous space problems. Regardless of this drawback, modern RL and AI have been successfully applying such methods, an important example is the use of Deep Q-Networks in games such as Atari, where the algorithms have surpassed human ability. Additionally, agent structures that are designed to learn the value functions are called **critic**.

Policy Approximation Whilst in value approximation the policy is inferred from the estimated value function, policy approximation algorithms directly define a policy and try to improve it through updates until it reaches a desired policy or the optimal policy. Often times such algorithms are referred to as *direct-policy search*. The direct search for the optimal policy handles continuous state and action spaces due to the direct correlation between the agent's policies and actions (and states, by default). Policy approximation is often paired with *gradient* optimisation methods, where optimal points of a function are found through a series of incremental time-steps in a direction guided by the slope, or gradient, of the function; These can be gradient ascent or gradient descent. A disadvantage of such methods is their tendency of falling to local optima. Finally, in contrast with *critic*-based agents, agents that directly learn a parameterised policy function are referred to as **actor**.

Actor-critic agents, were firstly introduced by [5] and combine value approximation and policy approximation. This is an attempt to eliminate the problems encountered in value approximation methods through the use of policy approximation. In this case, the action selection does not fall under the tasks of the critic (i.e. value-function), instead, this task is delegated to the *actor*. The *critic* is in charge of analysing the performance of the actor and deciding when the policy should be updated. This way the policy search algorithms maintain their local convergence features, whilst the value update variance is reduced [23] along with computational expense - due to the large amount of possible action in continuous spaces. There are nuances to these methods and the influence of the actor (or the critic) might be higher or lower, depending on the problem at hand [37]. This intuitive division of tasks makes actor-critic structures the preferred choice of developers to implement reinforcement learning algorithms, hence the most common in the state-of-the-art.

Challenges of Reinforcement Learning

To finalise the theory of reinforcement learning this section of the report briefly mentions some of the challenges currently faced in the world of reinforcement learning. Many of these issues are encountered across all the branches of RL and the fields that it incorporates.

Curse of dimensionality

This term was first introduced by Bellman during his explorations of optimal control problems with high-dimensional discrete spaces, according to himself he faced an *"explosion of states and actions"* [37]. However, the same problem can quickly be identified in reinforcement learning, especially when dealing with continuous spaces utilising classical methods such as DP, MC and TD. Evaluating entire state and action spaces rapidly becomes impossible. This problem has been indirectly introduced in the previous section, where function approximators were discussed. Although this is the main solution found on RL for the curse of dimensionality these are not the only ways of dealing with it, adaptive discretisation, macro-actions and options are also utilised solutions [37].

Reality Gap

In the field of aerospace, it is impractical not to use models when using reinforcement learning. aerospace systems are physical, hence required hardware. Physical parts oftentimes are expensive and require maintenance, hence are not expendable at will. This raises the topic of *safe exploration*. where frameworks are designed to allow exploration whilst retaining the safety of systems and surroundings. Additionally, problems in which flying is involved, such as *aircraft landing*, cannot be paused, sped up or started from an arbitrary state, most of the time only episodes can be repeated. Here, *learning speed* becomes paramount and researchers try to enhance it as much as possible. Hence reinforcement learning applications in the real world are expensive in terms of time, labour and finances.

Due to the factors mentioned in the previous paragraph, even model-free methods often require models for training and testing. Additionally, model-free algorithms are currently extremely *sample inefficient*, hence most RL achievements have been made within the simulation or virtual tasks, such as games. Simulation is widely used in the development of systems to offset the costs of real-world

testing. This raises another problem, the differences between the real world and the models used in simulations. This is called the *reality gap* and it is a problem because simulation requires accurate models, otherwise results in poor transfer-ability of learned policies and unpredictable agent behaviour. Additionally, accurate models require high amounts of data samples. There are two ways of overcoming that. (i) Improving transfer-ability of learned features through algorithm enhancing, and (ii) improving simulation fidelity without an increase in costs.

Curse of goal specification

The agent's behaviour is indirectly dictated by the specified *reward function*. In theory, it is easy to conceptualise what the goals of the agent are, however, in practice designing sufficiently adequate reward functions in mathematical terms is incredibly challenging. Additionally, reinforcement learning algorithms are notorious for finding solutions that have not been anticipated by the designer, often exploiting the reward function in an undesired manner, for example through finding local optima [37]. Solutions to the designing of reward functions problem have been found through *reward shaping*, *inverse optimal control* or *inverse reinforcement learning*, and though building complex policies on top of simple optimal control problems.

Stability

Stability is of extreme importance, especially in the field of control theory, where it becomes the central concern. Adaptive Dynamic Programming (ADP) attempts to define control laws for problems when models are unknown or partially known, in an online fashion. ADPs are further discussed in section 4.2, and even though they closely treat the matter of stability, it is still an open question in reinforcement learning; Additionally, to the guarantees of standard control approaches, RL approaches require additional stability, feasibility and robustness guarantees[11].

Other RL methods exhibit dips in performance during training, which is a natural process as the agents explore the environment and the action space that is available to them. However, that might scale to undesirable levels, which further corroborates the instability presented by RL algorithms. An example of an algorithm that strongly presents such behaviour is the DDPG, in an effort to solve this issue algorithms such as TD3, TRPO and PPO have been developed employing different techniques.

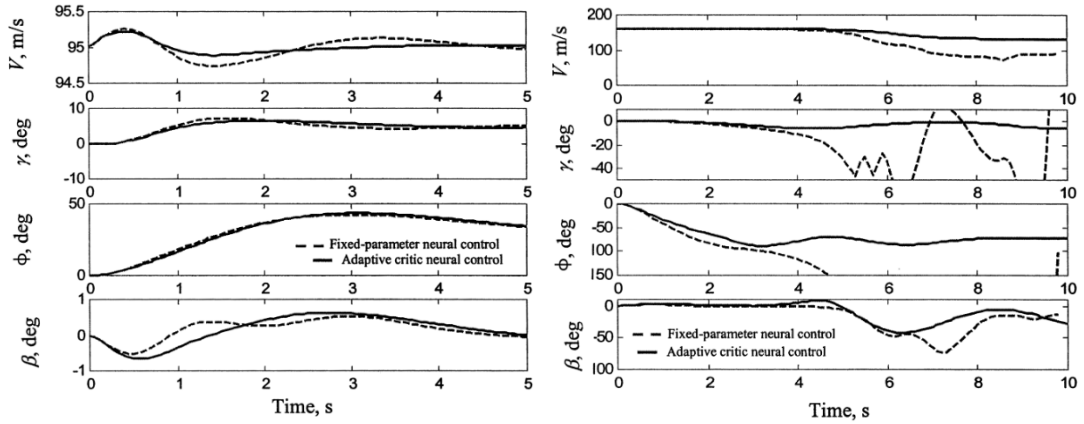
Reinforcement Learning for Flight Control

Reinforcement Learning, as previously stated, can be exploited in the field of control engineering as powerful a tool in the search for optimal controllers, especially in systems with non-linear and stochastic dynamics, that are unknown or highly uncertain. The reason for that becomes clear when RL is compared to a feedback system, from control engineering. The behaviour of the RL policy can be seen as the operations of a *controller*, the *error* as the reward, the *manipulated variables* as the actions and the environment would be everything else other than the inputs and outputs of the agent (i.e. *controller system/plant*, *feedback* and the *reference signal*).

The use of reinforcement learning techniques in Guidance, Navigation and Control (GNC) is relatively new in the field of aerospace systems. The main challenges to the expansion of RL in aerospace relate to safety-critical situations where trial-and-error-based techniques are often not practical, as well as expensive. Nonetheless, they have brought multiple benefits to the field, providing powerful tools to process raw sensory data and perform complex tasks. Over the years, there have been various proposed methods mainly utilising Approximate/Adaptive Dynamic Programming (ADP) techniques for adaptive control, and Deep Reinforcement Learning (DRL) overall, generally for robust control. This section gives a general overview of scientific papers that present the state-of-the-art RL algorithms applied to flight controllers and aviation-related systems.

In [3] the authors apply an actor-critic solution to solve approximate dynamic programming equations to solve aircraft control problems, the technique herein implemented is called Dual Heuristic Programming (DHP) and is further discussed in section 4.2. The study performed on the developed technique is brief and limited, however, the authors were able to conclude the usefulness of the algorithms and state that they can find near-optimal control policies for aircraft control problems. Additionally, it is stated that this approach has built-in fault tolerance, which is achieved through the use of the critic. Another implementation of DHP is presented in [19]. The authors prove the efficiency of the proposed design through the implementation of DHP to control a full six degrees of freedom of a business jet aircraft, in simulation. In this paper, training is split into two parts, one offline using established knowledge of control theory for initialisation, followed by an online phase, where the parameters identified in the first phase are updated to better represent the dynamics of the controlled system. Figure 3.11

compares the performance of the DHP and the PID controllers in two different scenarios, in figure 3.11a a low-bank angle turn with reference signals $\phi = 30°$ and $\gamma = 5°$ is shown, and figure 3.11b shows a high-bank angle turn with reference signals $\phi = -70°$ and $\gamma = 0°$.



(a) Low-bank angle turn **(b)** High-bank angle turn

Figure 3.11: Comparison between DHP and PID, represented by solid and dashed line respectively, retrieved from [19]

Yet another modification of the HDP method is proposed in [62], called IGDHP. Here the author proposes to use an online incremental model identification technique instead of the most common neural network training. Additionally, the incremental model is applied to GDHP, which stands for Dual Global Dual Heuristic Programming, hence the acronym IGDHP. Furthermore, the model identification allows for the elimination of the offline training phase of the agent that is required in previous methods, because it tracks non-linearities in a rapid and precise manner. The proposed method is tested on a second-order non-linear model of a generic surface-to-air missile in order to compare its performance against a generic GDHP method, where the controller has to learn the optimal policy in an online fashion to control the angle of attack of the missile. IGDHP presents more efficient learning of the model leading to a success ratio of 99.4 at the task, against only 37.2 from GDHP. Furthermore, in the search for more adaptive flight controls that are suitable for uncertain systems, [28] studied an RL-based control utilising Incremental Dual Heuristic Programming (IDHP). The controller utilises Neural Networks and is tested on a high-fidelity model of the Cessna 550 Citation II PH-LAB research aircraft, a CS-25 class fixed-wing aircraft. The results show that the framework is indeed capable of learning near-optimal policies in an online fashion without the need for prior offline training, due to the use of an incremental model. Additionally, it was observed that the adaptive controller is able to operate the aircraft in flight regimes that had not been encountered before, as well as adjust its behaviour for unforeseen changes to the aircraft's dynamics.

In [17] the author argues that it is unrealistic to develop flight controls for every possible unexpected failure in an aircraft, hence it is necessary to develop more robust controllers that are able to more strongly generalise control laws. In the paper, the author proposed a controller that is trained offline, cascaded and based on the Soft Actor-Critic (SAC) DRL algorithm. The proposed flight controller is tested in a coordinated $40°$ bank angle climbing turn and a variety of failure cases, including a jammed rudder and a rudder with reduced efficiency, and it proved to be successful at the tested tasks. The author credits the success of the system to the high generalisation power of neural networks and high exploration rates of the SAC's stochastic policy. The latter, however, makes the development of SAC controllers more difficult due to the low reliability and consistency of training-induced by the stochasticity of SAC's policies. Figure 3.12, shows the response of the aircraft controlled by the trained SAC algorithm on an altitude tracking task where the rudder gets stuck at $\delta_r = -15°$ at $s = 10s$. In the conclusion of the paper it is recommended to analyse the performance of Twin Delayed Deep Deterministic Policy Gradient (TD3), a similar algorithm that utilises deterministic policies, to improve training stability, and/or Proximal Policy Optimisation (PPO), an on-policy algorithm.

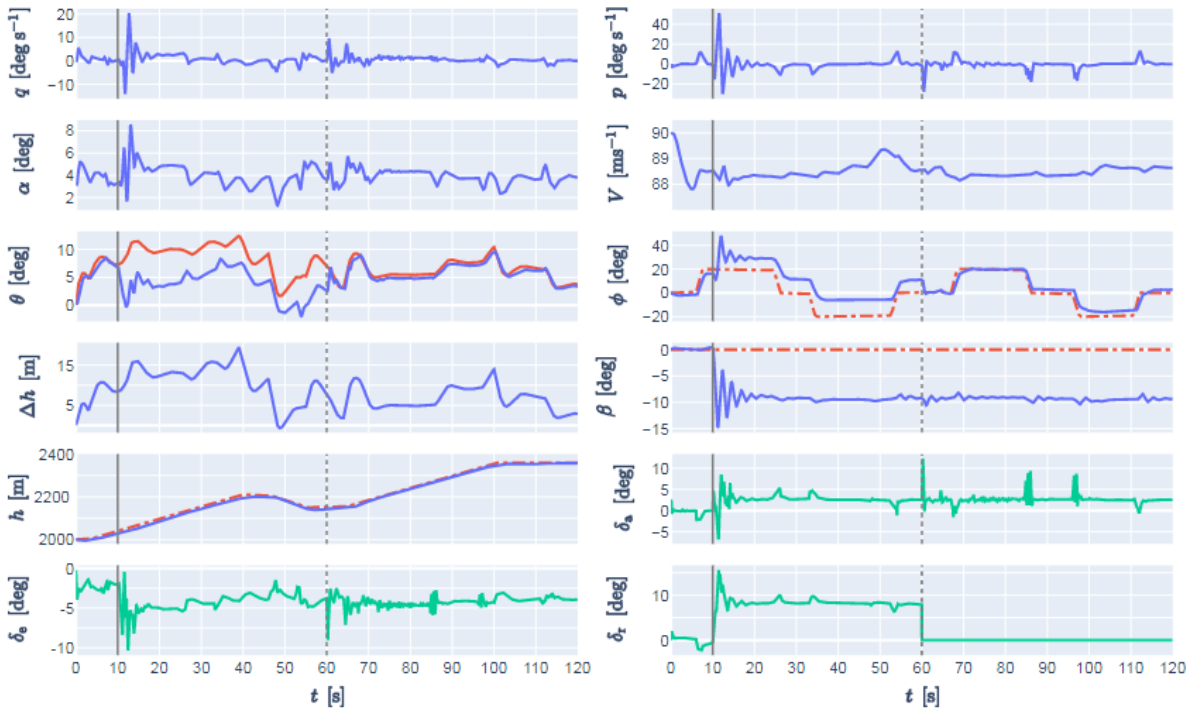


Figure 3.12: Altitude tracking response with rudder stuck at $\delta_r = -15°$ at $s = 10s$. Initially, only robust control is utilised, after $t = 60s$ adaptive control is utilised. Image retrieved from [17]

Next, [18] implements SAC and TD3 base controllers to a flying wind model to analyse the high angle of attack roll oscillations. Training of the agents is performed offline, due to the low sample efficiency of the algorithms. It was observed that both algorithms were able to learn near-optimal policies in a completely model-free fashion in simulation. The learning curves of the algorithms highlight the fact that TD3 has more stable learning than SAC. In the end, however, it is shown that the learning variance of SAC does not affect the final results, since the optimal version of the agent is used regardless of the previous processes. The authors also test the controllers using real-world wind tunnel data, where the controllers have to suppress roll oscillations to the craft. The controllers, however, were incapable of learning good enough policies to perform well in real-life situations.

More closely related to the intentions of the research herein performed is the work presented in [67], where a proof-of-concept automatic landing system is proposed utilising a Deep Deterministic Policy Gradient (DDPG) based controller. The author proposes three implementations: an outer loop controller and two direct controllers. The outer-loop controller was trained with winds ranging from $10ft/s$ to $75ft/s$ and it exert control over the pitch angle θ. The direct controllers are used to directly control the aircraft's elevator to follow the glideslope and the flare paths and were separately trained in two intensities of a random wind model, at $20ft/s$ and $75ft/s$. The three controllers underwent 15 validation flights where the aircraft's vertical speed, pitch angle and touchdown position, were evaluated at touchdown, following previous papers that have done the same. The controllers were tested in 15 different flight conditions, and the touchdown parameters were compared to those of seven previous works. It was shown that the DDPG agents were able to constantly track the gliding and flare paths throughout the landing as well as consistently perform soft landings, by holding the vertical speed under $1.9ft/s$. It was, however, recognised that certain results could have been improved and that certain DDPG hyperparameters were still held sub-optimal. The authors conclude by proposing the study of a system consisting of a DDPG controller outer loop in combination with a Non-Linear Dynamic (NDI) inner loop or even direct controller.

To conclude, a last paper is mentioned, where incremental ADP (iADP) and Deep RL (DRL) are combined in an attempt to extract the benefits of both methods. Inspired by recent IDHP and SAC methods,[68] proposed a hybrid SAC-IDHP framework to combine the adaptive nature of online learning with the robustness of offline learning to control a fully coupled system. The framework is adapted and implemented to the inner loop of a cascaded altitude controller of a six-degree-of-freedom high-fidelity

model of the Cessna 550 Citation II PH-LAB research aircraft. Tracking performance was tested for a simple altitude task with the aircraft in nominal condition, as well as under reduced efficiency of the elevator and the aileron. The results of the hybrid controller were compared to those of a SAC-only controller, and show that the hybrid controller is capable of holding lower tracking errors under nominal conditions and all failure tests. Figure 3.13 shows the performance of both the SAC-IDHP and SAC-only controllers to an altitude tracking task, here the former achieves a normalised mean absolute error $nMAE = 2.46\%$ and the latter $nMAE = 3.28\%$. Albeit performing better than the SAC-only controller, the hybrid controller still presents undesired oscillations in the longitudinal states, as well as inconsistent training, characteristic of SAC.

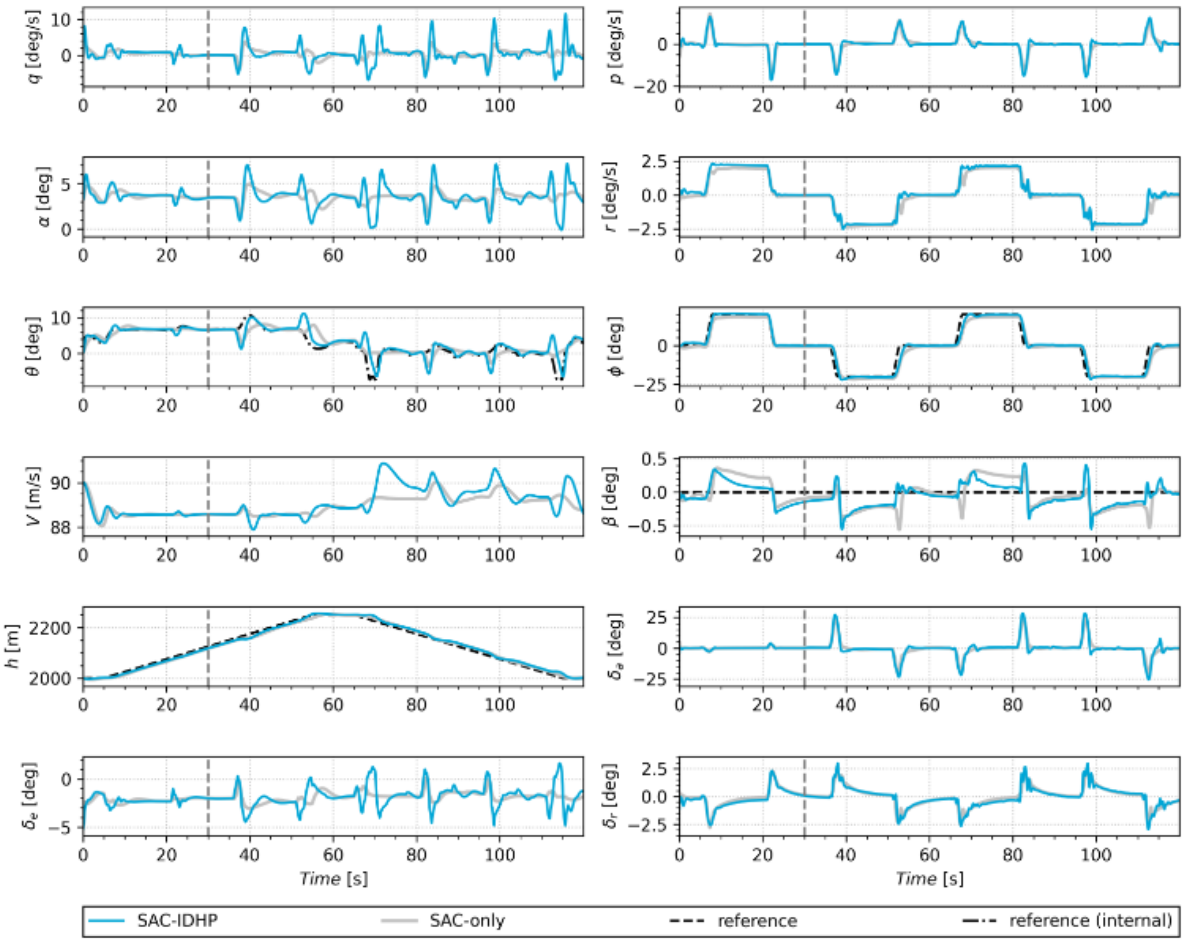


Figure 3.13: Altitude tracking response with aileron efficiency reduced by 90% at $t = 30s$.Image retrieved from [68]

Partial Conclusions

Section 3.2.2 highlighted the importance of applying certain techniques to the classical reinforcement learning methods such that they are adapted from discrete state and action spaces to continuous spaces. Automatic landing, along with most real-world problems, belongs to the second category. Hence, to identify algorithms of families of algorithms that suit the purposes of this project it is first necessary to identify the function approximator's parameters most suitable to the problem at hand, as well as the agent structure.

First, based on previous studies on RL applications for flight control [17] [28] [68] and analysis of the possible aspects of function approximators it is observed that **non-linear parametric approximation** is the most suitable structure for the flight control problems. This is simply due to the fact that non-parametric approximators require data that is not compatible with that generated by the reinforcement learning problem [71], hence not suited for this project. Regarding linearity, approximators that present non-linear structures have been chosen for this project because they provide advantages over their linear counterparts. Linear approximators present limited accuracy and require knowledge of the environment. Furthermore non-linear approximators have show promising empirical results, mainly through experimentation with ANNs [71].

The state-of-the-art reinforcement learning algorithms use artificial neural networks for function approximation, i.e. estimating policy and value functions. ANNs are computing systems that simulate the networks of neurons in the nervous system of a biological cell [76]. These networks are trained by estimating the difference between the output of the network and the output of the system (target output), which is called the *error*. In reinforcement learning that would be mapping state to values, or state-action pairs to $Q(s, a)$ functions. Once the difference has been calculated, the network updates the weighted parameters to reduce the error. The key difference between ANNs to other approximators is that the processing of information occurs through interconnected layers of neuron-like structures referred to as *perceptrons*. These artificial neurons are constituted of inputs, weights, a Transfer Function and an activated function [76], as shown in figure 3.14. ANNs are made of an input layer, hidden layer(s) and an output layer, as shown in figure 3.15. Further discussion of ANN structures is left to be treated in the main phase of the thesis project. It suffices to state that the only ANN assumption is that the function to be approximated is smooth, and that given enough parameters, a strong generalised approximation can be made.

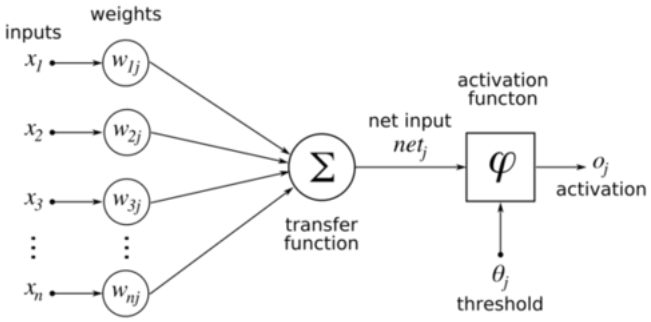


Figure 3.14: Artificial Neuron (perceptron) structure, retrieved from [76]

Furthermore, **gradient-based** methods have been chosen, because they have shown faster and more concrete optimisation results than their gradient-free counterparts. The only requirement for gradient-based optimisation is that the optimised functions must be differentiable, which is an inherent characteristic of ANNs, therefore can be used in combination with gradient-based optimisation.

Second, **actor-critic** is the current state-of-the-art agent structure and has been used multiple times in literature for flight control applications [62] [78] [3] [67]. This favouritism for this specific agent structure is due to its intuitive division of tasks, where *learning* is solely reserved for the critic and *controlling*, that is action selection, is solely reserved for the actor. Here the critic must learn and evaluate, "critique", the policy that is currently being used by the actor for action selection, as shown in figure 3.16 this evaluation takes place through a TD error, which should be reduced by the actor. This task division has been proposed as a means of reducing the total complexity of the general problem RL attempts to solve [35]. When actors are combined with critics, the disadvantages of singular methods are eliminated whilst retaining the advantages of both methods, that is to say, that the resulting agent is highly capable of

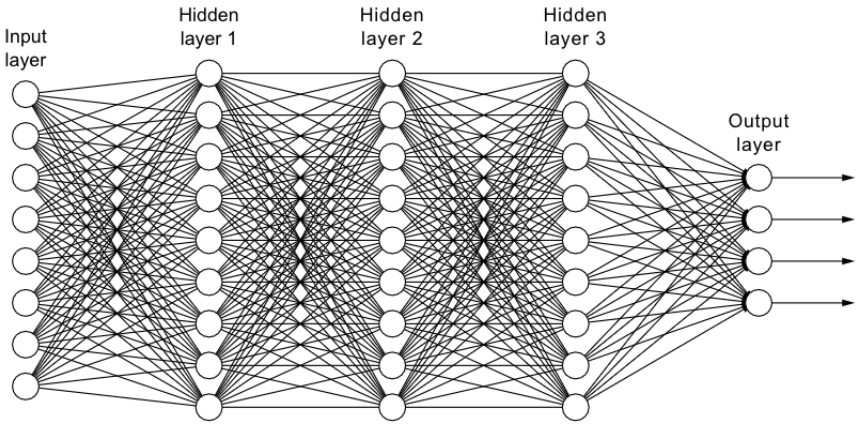


Figure 3.15: Fully connected Artificial Neural Network structure with three hidden layers, retrieved from [54]

handling high dimensional action and state spaces whilst reducing variance in value estimation.

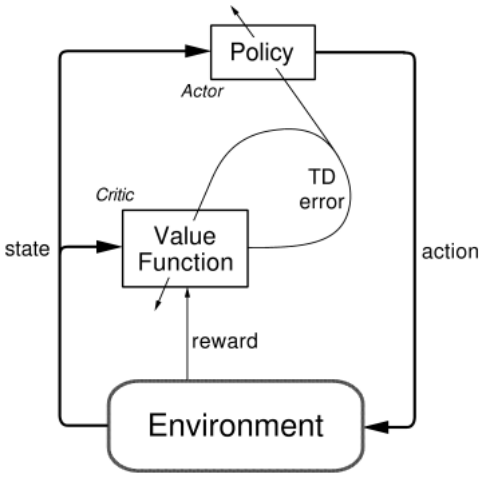


Figure 3.16: Diagram of the actor-critic architecture, retrieved from [63]

Additionally, [65] states that actor-critic agents are likely to remain of interest due to low computational requirements for action selection and their ability to learn explicitly stochastic policies. In general terms, actor-critic agents are well suited for continuous state and action spaces problems, due to policy approximation (actor), and present better policy gradient variance, due to value-approximation (critic) [65].

Third, model dependency is not a desired trait in the ALS algorithm; as explained before, the major problems with automatic landing occur when there are changes to the dynamics of the internal and external systems. When a controller is model-based its efficiency is not only dependent on its structure and how it has been designed, but also on how accurately the model utilised represents the controlled system. Under uncertain circumstances, a controller model often fails to accurately represent the real world. Additionally, model-based algorithms that *learn* a model tend to be much more computationally expensive than their model-free counterparts. For these reasons, it is desired that the automatic landing controller be designed to present **low model-dependency**, such as those found in model-free algorithms.

Defining those aspects of the algorithm to be chosen narrows down the suitable algorithms to two groups of methods. namely *Approximate Dynamic Programming (ADP)* and *Deep Reinforcement Learning (DRL)*. These are not completely separate from each other and there is significant overlapping; however, they are often referred to as two distinct groups in literature. These two methods are described in the next sections along with a brief introduction of the main algorithms that are categorised under those groups. Figure 3.17 presents an expanded version of the previous reinforcement learning

diagram containing the fields of ADP and DRL. In this project, the focus is on RL and ADP algorithms that make use of ANNs and present an Actor-Critic agent structure.

Furthermore, the automatic landing problem requires fault tolerance, this can be achieved through adaptive control and robust control. In reinforcement learning these terms are closely related to online and offline learning, respectively. In hindsight, online learning seems to be more suited to the problem at hand, due to its capability of dealing with highly dynamic environments. However, offline learning can provide certain fault tolerance, through robustness, making it less error-prone and the designer has control over the data that is fed to the agent. Hence, at this stage, it is not possible to eliminate one of the methods. A third option would be to implement online-offline hybrid learning, such as the method proposed by [68].

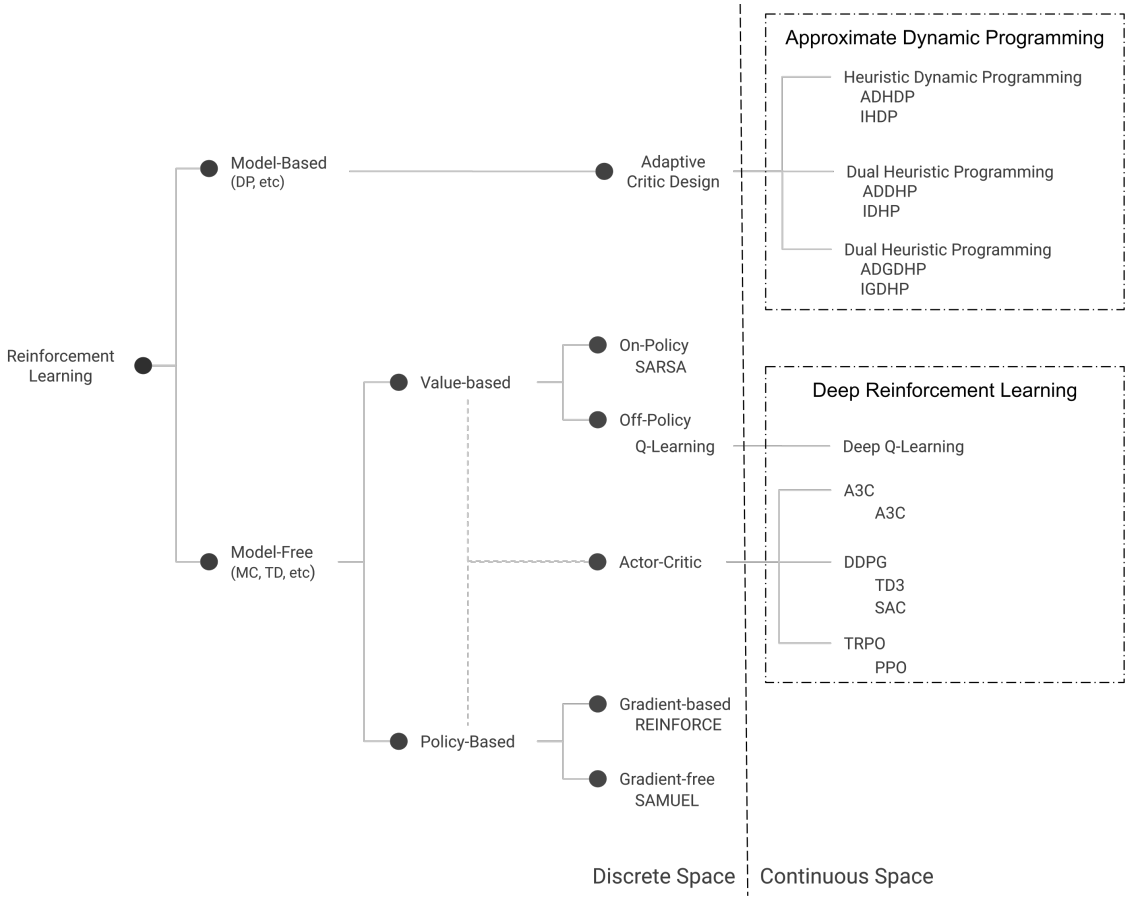


Figure 3.17: Taxonomy of Approaches considered in this project

3.2.3. Approximate Dynamic Programming - ADP

Approximate or Adaptive Dynamic Programming includes control-theoretical approaches and techniques utilised to approximately solve optimal control problems. In literature, it is not rare to find ADP and RL being utilised interchangeably, and that may be true in some fields, however, generally, it is not [11]. ADP does not necessarily have a direct connection to RL, but can be used in RL problems, especially those methods within ADP that involve learning, utilise ANNs and have a low model dependency. As the name states, ADP utilises approximations to the dynamic programming approach, in order to deal with the curse of dimensionality and adapt DP methodology to continuous state and action spaces. The application of ADP in highly dynamic and large systems mainly utilises actor-critic agent structures, whose advantages are outlined in the previous section and outweigh their actor or critic-only counterparts. Hence the methods herein presented are limited to the class of ADP that makes use of such structures, named Adaptive Critic Design (ACD).

Originally, ACDs were introduced utilising ANNs for function approximation, a combination that is used in this project, however, they are not limited to these methods and other approximation techniques

may be used. ACDs can be mainly divided into three design families (i) *Heuristic*, (ii) *Dual Heuristic* and (iii) *Global Dual Heuristic*. Whilst the actor's structure stays largely the same, i.e. mapping states to actions, the structure of the critic varies resulting in the different strategies of ACDs. The main algorithms relating to the three families are named *Heuristic Dynamic Programming (HDP)*, *Dual Heuristic Programming (DHP)* and *Global Dual Heuristic Programming (GDHP)*. Moreover, these basic algorithms are solely dependent on states, and actions do not play a role in the critic, hence they compute state-value functions. The adaptation of such algorithms to be action-dependent include the prefix AD to the original titles, i.e. ADHP, ADGHP and ADDHP, where the critics estimate the state-action value functions. A taxonomy of such methods can be seen in figure 3.18

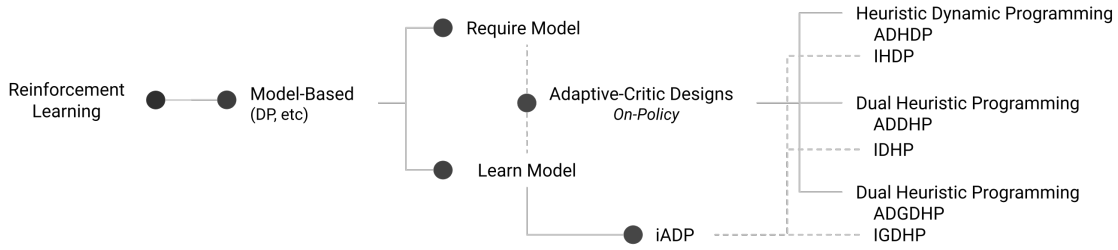


Figure 3.18: Taxonomy of Adaptive Critic Designs of ADP

Furthermore, DP is model-based, which has multiple downsides when applied to highly dynamic environments. Since ADPs are derived from DPs, these are also mode-based. However, attempts have been made to reduce model dependency of classic DP methods. An example of that is the previously mentioned AD variations of the algorithms, which assume a direct connection between the actor and the critic networks [53]. Most interesting for this research is a specific branch of ADPs that uses incremental model identification, this branch of ADP is referred to as iADP. Model identification is done in the RL framework, through model approximation by a linearised, time-varying incremental model. The incremental fashion of the approximation makes it suitable for online adaptive control, where increments occur each time step using the previous step's condition of the model. The decrease in model-dependency that iADP providers overrule any improvements the AD variations make, hence "iAD-ADP" methods would frivolous, and thus do not exist.

Heuristic Dynamic Programming - HDP

HDP is the simplest form of adaptive critic design. In this method, the critic simply estimates the state-value function $V(s_t)$ in the Bellman equation of dynamic programming based on the state of the environment s_t it receives. The task of the actor-network is to map states to actions to derive a policy. In this case, it uses the output of the critic, i.e. $V(s_t)$, to derive the state value with respect to the action. Its output is $\frac{\delta V(s_t)}{\delta a_t}$, which is calculated by $\frac{\delta V(s_t)}{\delta s_t}\frac{\delta s_t}{\delta a_t}$, hence it requires back-propagation and knowledge of the state transitions. Due to the latter, the actor in HDP is model-based.

ADHDP on the other hand eliminates the model dependency of the actor through the critic. In this modification, the critic estimates the action state value function $Q(s_t, a_t)$, and subsequently, the actor derives its policy from $\frac{\delta Q(s_t,a_t)}{\delta a_t}$. The last operations only require back-propagation, and the need for a transition state model is eliminated [53]. An in-depth comparison between HDP and its action-dependant variant ADHDP has been performed in [35], the study shows that HDP has nearly twice as high converging accuracy as ADHDP does in the initial offline training phase; However, ADHDP can be more adaptive, since it showed to perform better than HDP when noise is introduced. The final conclusion of the study is that HDP can operate in a wider range of flight conditions than its counterpart.

Another attempt at reducing model dependency is the utilisation of incremental techniques to model identification. This adaptation completely eliminates the necessity of prior knowledge of the dynamical system of the environment, requiring only an online identified incremental model. Additionally, an interesting advantage of such variation is that it also eliminates the need for an offline training phase, that is because model identification happens fast and with enough accuracy to allow for only an online phase.

Dual Heuristic Programming - DHP

Unlike HDP, the critic network of DHP is the one responsible for estimating the derivative of the state-value function with respect to the states of the environment. Hence, the critic directly outputs the derivative $\frac{\delta V(s_t)}{\delta s_t}$; therefore, the critic is the one that requires a model of the environment, instead of the actor. On the same note, the actor is also model-depended, this is because it still needs knowledge of the state transitions to estimate the policy; which is done through differentiating the critic's input with respect to the actions. However, with this technique, the need for back-propagation is completely eliminated, which results in smoother and more accurate results [72], if compared to HDP.

The action-dependent variation of the algorithm eliminates the actor's model dependency. However, ADDHP assumes that there is a direct connection between the actions and the critic network and unlike ADHDP, the pathways of back-propagation are still maintained through a model network. Hence, the critic is still model-dependent.

Once again, the incremental version of DHP reduces model dependency through model identification. IDHP has shown to outperform its counterparts with regards to efficiency, accuracy and robustness, in fully online learning [77].

Global Dual Heuristic Programming - GDHP

Finally, the last general group of ACDs combines the characteristics of both previous ones, that is, in GDHP the critic network is responsible for estimating both the state-value function $V(s_t)$ - like HDP - and its derivative with respect to the state $\frac{\delta V(s_t)}{\delta s_t}$ - like DHP. This is done through two critic networks, where the first is model-independent, and the second is mode-dependent because it requires the state transition model. In this algorithm, the actor is model dependent, as HDP and DHP's actors are. In theory, GDHP outperforms DHP, such as DHP outperforms HDP. However, GDHP is much more complex than its prior counterparts, being the computation of a second order derivative, i.e. $\frac{\delta^2 V(s_t)}{\delta s_t \delta a_t}$, the major reason for that. This additional complexity hinders the implementation and algorithm computation, which oftentimes makes DHP a more efficient framework.

GDHP also has its action-dependent and incremental versions, which work in a similar fashion as has been explained before. The actor's model dependency is eliminated in ADGHDP, and the critic remains mode-dependent. The increase in complexity from IGHDP to IDHP, in general, does not justify the increase in performance.

Synopsis

The information provided in the previous sub-sections is provided in table 3.3 to be used for quick reference.

Table 3.3: Summary of ADP ACD methods and its characteristics

Algorithm	Critic	Actor	Model knowledge
Heuristic			
HDP	$V(s_t)$	$\frac{\delta V(s_t)}{\delta a_t}$	Actor
ADHDP	$Q(s_t, a_t)$	$\frac{\delta Q(s_t,a_t)}{\delta a_t}$	-
IHDP	$V(s_t)$	$\frac{\delta V(s_t)}{\delta a_t}$	-
Dual Heuristic			
DHP	$\frac{\delta V(s_t)}{\delta s_t}$	$\frac{\delta s_t}{\delta a_t}$	Actor & Critic
ADDHP	$\frac{\delta Q(s_t,a_t)}{\delta s_t}$, $\frac{\delta Q(s_t,a_t)}{\delta a_t}$	-	Critic
IDHP	$\frac{\delta V(s_t)}{\delta s_t}$	$\frac{\delta s_t}{\delta a_t}$	-
Global Dual Heuristic			
GDHP	$V(s_t)$, $\frac{\delta V(s_t)}{\delta s_t}$	$\frac{\delta s_t}{\delta a_t}$	Actor & Critic
ADGDHP	$Q(s_t, a_t)$, $\frac{\delta Q(s_t,a_t)}{\delta s_t}$, $\frac{\delta Q(s_t,a_t)}{\delta a_t}$	-	Critic
IGDHP	$V(s_t)$, $\frac{\delta V(s_t)}{\delta s_t}$	$\frac{\delta s_t}{\delta a_t}$	-

ADP methods are considered in this research because they are mainly implemented in an online adaptive fashion, which is well suited for highly dynamic environments. A characteristic that is strongly in line with the current needs of ALS for more robust and *adaptive controllers*, due to the issues that

appear when uncertainties are introduced in the environment. Furthermore, ADPs are derived from DP methods, hence are model-based, which is a characteristic that is not desired for the system to be designed in this project; That is due to the fact that, in general, a model-based controller is only as good as the model provided/learnt, and it is more computationally expensive than its model-free counterpart. On the other hand, iADP's concept of using an incremental model to deal with the non-linearities of unknown systems and uncertainties present in the environment reduce significantly the model dependence present in other ADP methods.

In general terms, ADPs are divided into HDP, DHP and GDHP. As mentioned previously, the increase in complexity from GDHP does not justify the increase in performance that it presents from its DHP counterpart, hence GDHP shall not be further considered. Moreover, a study on the performance of ACDs on the challenging auto-lander problem showed the DHP has better performance than HDP, with the expense of longer training [53]. Although the authors report that longer training for HDP did not present further performance improvements, hence it is inferred that HDP has a lower performance ceiling than DHP does.

For the reasons above, ADPs are still under consideration to be used in this project, however, it remains to be seen how the ADP method would play a role in the upcoming phases. However, among the possible methods, IDHP would be a better fit for the automatic landing problem due to higher performance promises than its counterparts and reduced model dependency.

3.2.4. Deep Reinforcement Learning - DRL

A promising field that has seen a surge of research in recent years is what is called Deep Reinforcement Learning. DeepMind's atari playing agent by [48] was the first to combine Deep Learning (DL) and RL, in 2013, hence giving the name Deep Reinforcement Learning. Since then, due to its flexibility, DRL has been successfully applied to a number of different contexts and fields. The DNN's structure contains multiple hidden layers (figure 3.15), where the approximation is achieved by computing nonlinear transformations of the outputs between the layers. This characteristic has been proven very useful, particularly for approximating functions of natural data such as images, sounds and languages ([11]). Furthermore, the combination of RL and DNNs also results in very general algorithms, this strong generalisation power makes for especially *robust* algorithms. Additionally, to their strong generalisation power, DRL features the ability to deal with large continuous state spaces, which makes it suitable for control algorithms.

Technically speaking, any RL algorithm that makes use of ANNs can be considered to be Deep RL. The informal definition of the term, however, slightly differs and often times it is less intuitive. In control engineering, DRL is more often than not referred to when *offline robust* control is the focus of discussion, as opposite to ADP, which is referred to when *online adaptive* control is the focus. Generally, DRL is a broader term that is theme of discussion of many fields in the academic world, therefore, it is imperative to clarify the exact meaning of the terminology employed. In this report, the term DRL is reserved to model-free RL algorithms that make use of DNNs in their agent.

Furthermore, the search for algorithms that follows is focused on DRL algorithms based on *actor-critic* architectures (hence, for sake of practicality this group of algorithms may be referred to as simply DRL). The state-of-the-art DRL actor-critic algorithms may be based on: (i) Policy Gradient, (ii) Advantage function, and (iii) Policy Optimisation, such as can be seen in figure 3.19. Whose main algorithms are: (i) DDPG, TD3, SAC, (ii) A3C and A2C, and (iii) TRPO and PPO. It is important to note that there are many more algorithms and modifications to them in the world of RL, however, it falls under the scope of this project as well as is impractical to examine every single one of them, hence the research herein presented limits itself to the general algorithms of the most common methods.

Furthermore, because DRL capable of mapping state to actions in high-dimensional problems the approximation results in high correlation between states and actions. This correlation might result in divergence or oscillation of the parameter weights of the ANNs, which affects the learning process of the agent. This problem is tackled with (i) experience replay (or replay buffer), and (ii) delay target (or target network). In the former past state, actions and rewards (S, A, R, S') are stored in a buffer, which are recalled in a later moment, and a randomised order. This breaks the temporal correlation of the training samples, and since off-policy algorithms can find the optimal policy from other policies samples, it means that the stored experiences can be used for further learning by the agent [11]. In the latter solution, the target values are calculated using an older version of the value function, instead of computing with the current version, as it is typically done. This breaks the correlation between the

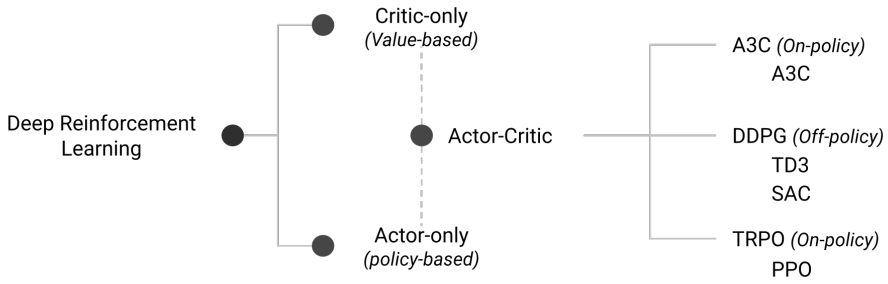


Figure 3.19: Taxonomy of DRL algorithms based on the actor-critic agent structure

calculated Target values and the network prediction. Additionally, DRL algorithms are known to be sample inefficient

Policy Gradient

As stated before, the goal of reinforcement learning is to teach an agent how to learn a suitable behaviour for the situations it finds itself in, and that can be done in multiple manners. Policy gradients are perhaps the most common and useful algorithms to work in continuous spaces [61]. These methods are amongst those that target learning of optimal policies directly; Note that this can be done with or without the presence of a value function, i.e. in an actor-critic or a critic-only fashion, being however, actor-critic methods more numerous [6]. Policies are usually parameterised functions of states and actions that aim to map the former to the latter, which often take the form: $\theta\pi_\theta(a|s)$.

The underlining principle of policy gradient methods is that the parameters θ of the policy function are adjusted in the direction of the performance gradient, that is following the optimal slope of the cost function, in RL terms, the cumulative reward. To summarise, the *Policy gradient theorem* states that the derivative of the expected reward is the expectation of the product of the reward and the gradient of the policy, which is expressed in equation 3.17 [64]. REINFORCE is an off-policy policy gradient algorithm that is worthy of mentioning, even though it is not further discussed in this report due to not utilising ANNs and being actor-only.

$$\nabla J(\theta) = \mathbb{E}[\nabla_{(\theta)} log\pi_\theta(s,a) Q^{\pi_\theta}(s,a)] \tag{3.17}$$

DDPG

Deterministic Policy Gradient (DPG) is an off-policy actor-critic variant of the policy gradient category introduced by [61]. DPG reduces long computation by utilising a deterministic policy instead of a stochastic policy, where more samples are required since the data is integrated over the entire (often large) state and action spaces. According to the authors of the paper that introduces DeepMind's DPG, this characteristic renders it significantly better performance than other on and off-policy stochastic gradient methods [61]. On the other hand, the deterministic nature of the policy makes exploration difficult. Additionally, DPG presents limited accuracy in function approximation and also requires knowledge of the environment, this is because it makes use of tile coding, which is a type of linear approximation.

Deep Deterministic Policy Gradient (DDPG) is then introduced by [40] as a model-free version of DPG where both the actor and the critic are represented by DNNs, hence no longer relying on linear approximators. It also combines DQN techniques, where it learns Q-functions by experiencing replay and delayed targets. The adaptation of learning Q-functions from discrete to continuous spaces is done utilising the actor-critic framework whilst learning a deterministic policy (DPG). That means that the delayed target network has to be implemented not only for the critic, as in DQN, but also for the actor. Additionally, DDPG makes use of *conservative policy iteration*, where the parameter updates, for both the actor and the critic, are restricted to slow changes through an Exponentially (weighted) Moving Average (EMA) [40]. This is done to increase overall stability, however, might lead to longer training periods for the agent.

Similar to DPG, DDPG treats the exploration problems by adopting off-policy learning, where the *target policy* is reserved for learning the optimal policy, whilst the *behaviour policy* may be inclined

more (or less) towards exploration. The exploratory character of the behaviour policy π' is introduced through the use of noise η, such as shown in equation 3.18.

$$\pi'(s) = \pi_\theta(s) + \eta \tag{3.18}$$

Figure 3.20 shows a block diagram where the working structure of the DDPG algorithm is outlined. Starting from the actor (Policy Network), the algorithm chooses an action based on a randomly initialised actor policy and exploration noise. Note that, unlike the rest of this report, the block diagram represents the policy by μ instead of π. The environment then feeds the critic the rewards and the transition state and actions (s_t, a_t, r_t, s_{t+1}) are stored in an experience replay buffer. After a number of such cycles, a batch of transitions is randomly sampled and fed to the critic (Q Network) who optimises the loss function using the current and target networks, the algorithm then updates the actor's policy based on the critique it received and updates both (actor and critic) target networks. Once again, actions are taken following the new policy with random noise added for exploration and the cycle repeats itself. Additional to the diagram herein presented a pseudo-code of the algorithm, is provided in figure A.1 in appendix A, retrieved from the original DDPG paper [40].

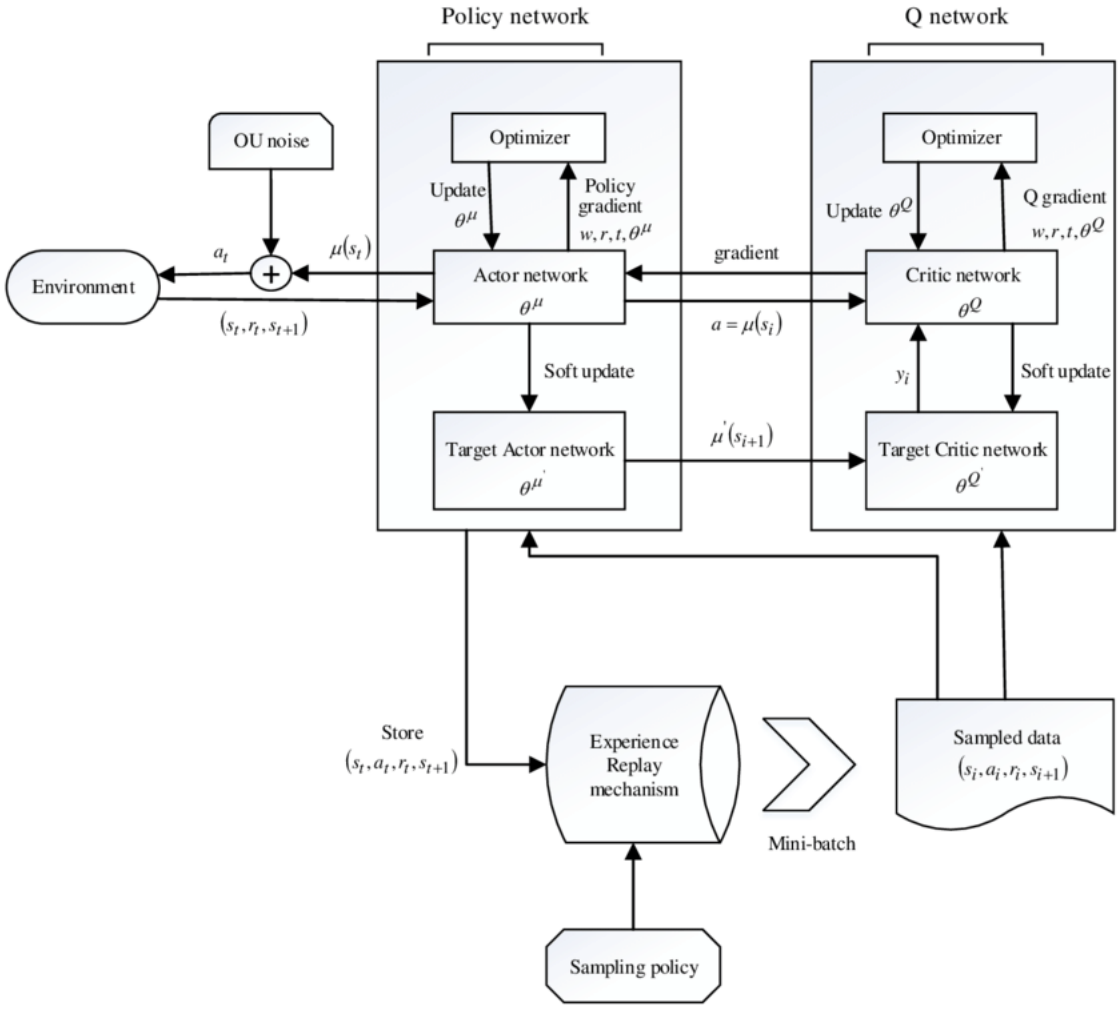


Figure 3.20: DDPG algorithm structured in a block diagram, retrieved from [24]

Over the years there have been multiple variations of the DDPG algorithm, such as Distributed Distributional DDPG (D4PG) [4] that applies distributional techniques in an attempt to improve the original algorithm, and Multi-Agent DDPG (MADDPG) [42], which attempts to augment performance in highly dynamic environments by introducing the idea of utilising multiple agents to complete tasks only using local information. However, there are two algorithms that have been the main focus of studies

due to their elevated performance compared to others, these are called Soft Actor-Critic (SAC) and Twin Delayed DDPG (TD3) and are described further in this report.

The key components of DDPG are:

1. **Off-policy** formulation for increased exploration (if compared to general On-policy methods)
2. **Deterministic policy** to reduce long computations and increase sample efficiency

TD3

A notable adaptation of the DDPG algorithm is the Twin Delayed DDPG (TD3) [21] which addresses the function approximation issues of DDPG. It is not uncommon for the value function to be overestimated in DDPG, this is an issue inherited from the DQN algorithm that has persisted and can be carried on through the training iterations (negatively) affecting the learned policy. The overestimation issue has also been a motivator for the development of other algorithms such as Double Q-learning and Double DQN. This issue is addressed by introducing a twin Q-value approximation network and less frequent updates.

The twin (or double) Q-value function utilised in the double DQN (value-based) algorithm features two different, uncoupled, networks responsible for action selection and Q-value estimation, respectively. In actor-critic algorithms the action selection is already delegated to the actor (policy approximation), hence TD3 adapts the DDPG algorithm to employ two uncoupled networks for the critic and a single actor. Additionally, TD3 updates the policy at a slower rate than that of the Q-value, more specifically, the authors suggest that to be done at half the rate. It was found that by doing so the error of the Q-function is lowered before the policy is updated, hence the policy update is more accurate and the variance in the presence of the target networks is further reduced.

The modifications to DDPG that result in the TD3 algorithm render a block diagram similar to that of in figure 3.20. The major change to that is the fact that there are two critic networks that make estimates independently, the estimated Q functions are then compared and the smaller of the two is used in the Bellman error loss function. As mentioned previously, the policy is updated less frequently than the Q functions, instead of concomitantly as DDPG does. Finally, noise is added to the next state actions in order to smooth it, this is only possible due to its deterministic policy. Additionally, a pseudo-code of the algorithm is provided in figure A.2 in appendix A, retrieved from the original TD3 paper [21].

According to the authors, the TD3 algorithm surpasses the DDPG algorithm regarding both learning speed and performance in a number of continuous control benchmark tests, such as MuJoCo's HalfCheetah, Hopper, Walker2d, and Ant, to name a few [21]. Additionally, to be superior to DDPG in those tasks, the authors claim that the TD3 also matched or outperformed other algorithms such as the Trust Region Policy Optimisation (TRPO), Actor-Critic using Kronecker-Factored Trust Region (ACKTR) and SAC, in those same tasks. Note however, that SAC and TD3 were initially published in the same conference paper, the algorithms have since then been modified, hence initial comparisons, such as this, between the two are not necessarily up to date with their current versions.

The key components of TD3 are:

1. **Off-policy** formulation for increased exploration (if compared to general On-policy methods)
2. **Deterministic policy** to reduce long computations and increase sample efficiency
3. **Double Q-value** approximation to reduce value overestimation

SAC

The Soft Actor-Critic (SAC) algorithm bridges the stochastic policy optimisation to the DDPG-driven approach. It does so by combining the Soft Q-learning algorithm's maximum entropy reinforcement learning framework to an off-policy actor-critic model. The SAC makes use of the *stochastic policy optimisation*'s concept of incorporating the entropy measure of the policy in the reward function to enhance the exploration rate, it is then expected to learn a policy that acts in a random fashion whilst still succeeding at the task at hand. It alternates between soft policy evaluation and soft policy improvement which, probably, leads to convergence to the maximum entropy policy.

The SAC algorithm also makes use of two Q-value functions, just like TD3 does, however, it is not a direct successor, since both algorithms were published roughly at the same time. Both functions are trained to optimise the cost function, the minimum of both functions is then used to estimate the value and policy gradients. The algorithm works by using experience replay to update the function approximator's parameters through stochastic batches of data that have been sampled using the current

policy. It has been found that using two Q-functions speeds up training in harder tasks significantly, even though using a single Q-function is sufficient to perform challenging tasks, such as learning to control a 21-dimensional Humanoid [25]. Additionally, the two Q-functions mitigate positive bias in policy improvement, a well-known issue of value-based methods.

Like TD3, SAC has a block diagram similar to that of DDPG, in figure 3.20, where there are two Q networks instead of a single one. Unlike TD3 however, the Q-function target update that happens after a random batch is sampled from the experience replay memory contains a term that comes from SAC's use of entropy regularisation. Furthermore, SAC the state actions used in the target network follow the current policy, unlike TD3 which follows the target policy. SAC also foregoes the necessity of adding noise to the next state actions for smoothing purposes, this can only be done because it employs a stochastic policy. Additionally, a pseudo-code of the algorithm is provided in figure A.3 in appendix A, retrieved from the original TD3 paper [25]

The key components of SAC are:

1. **Off-policy** formulation for increased exploration (if compared to general On-policy methods)
2. **Stochastic policy optimisation** through entropy maximisation to increase stabilisation
3. **Double Q-value** approximation to reduce value overestimation

Advantage Function

A well-known issue with vanilla policy gradient methods is their high gradient estimate variance. A standard approach to tackle this problem is to subtract the expected reward with a baseline value *b*. There can be multiple choices for *b*, a common and logical choice is the value function $V(s)$, when this is the case the equation is named the *advantage function*. Such computation is an estimate of how beneficial a particular action is compared to the average action in a specific state; Hence, intuitively increasing the chances of actions that perform better than the average happening. The advantage function is mathematically defined simply as the difference between the state-action value function $Q(s, a)$ and the state value function $V(s)$, shown in equation 3.19. The stability of the algorithm is increased because the advantage function reduces the variance in the gradient estimate between the new and the old policies.

$$A(s, a) = Q(s, a) - V(s) \tag{3.19}$$

The operation of the advantage function is based on the signs of its computation. If an action taken produces a *positive* value of the advantage function, this action was good and hence should have a higher probability of happening. The opposite is true if an action produces a *negative* value, and its probability decreases. Logically, if an action results in a null value for the advantage function, it means that the action taken was nor good, nor bad, and that only happens when the optimal policy has been found.

A3C

A3C is an on-policy actor-critic algorithm that makes use of advantage functions in its operation. Introduced by [47] it stands for *Asynchronous Advantage Actor-Critic*, it can be described s a classical policy gradient method that emphasises parallel training. What sets this method apart from others is the fact that it utilises multiple agents that are trained simultaneously. Each agent gets its policy and value from a *Global Network*, with parameters θ and θ_v, and explores the environment on its own, until a t_{max} actions or a terminal state is reached. Each agent then uses their own experiences to update their policies and values, then represented by local networks with parameters θ' and θ'_v. These updated *Local Networks* are then combined in a batch and used to update the parameters of the *Global Network*. Figure 3.21 shows how these parallel actors can be trained simultaneously, act in different versions of (models of) environments, and combined in a global agent. Furthermore, the use of multiple agents eliminates the need for experience replay, which makes the algorithm faster and possibly with increased performance.

Additionally, A3C is also an *asynchronous* method, which means that the different agents are running in asynchronous threads and update the global and local networks at different points in time. This implies that each algorithm explores different parts of the environment with a slightly different policy. For some this is considered a disadvantage because some agents are operating under an outdated of the Global Network, hence will always operate under sub-optimal conditions [73]. For others this is considered an advantage, arguing that this may increase performance because it decreases correlation

amongst the samples [30]. Regardless, A3C ameliorates the correlation problem of DRL, due DNNs assume input samples are independent of each other, which in most cases is not true, by combining data acquired by multiple agents.

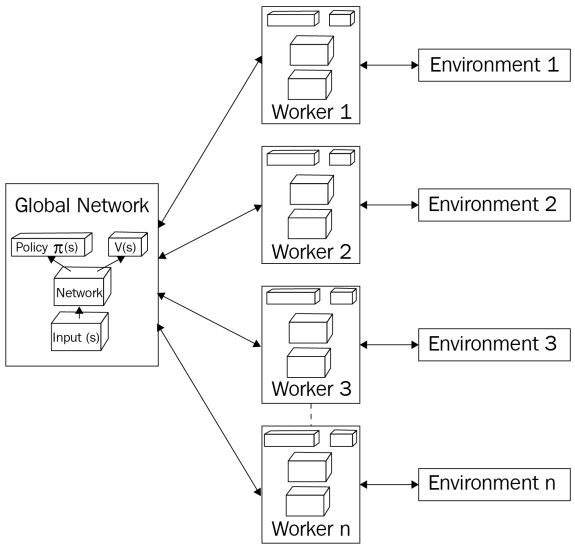


Figure 3.21: How parallel actors are trained together and combined into a single global network, retrieved from [56]

The key components of A3C are:

1. **On-policy** formulation for reduced variance and faster training times (if compared to general Off-policy methods)
2. **Advantage function** to reduce variance
3. Introduction of **parallel training** of multiple agents, for both policy and value function estimation. Updating scheme based o fixed lengths of time (or amount of gained experience)
4. **Asynchronous** updates

A2C

A2C deals with the concern of some researchers that the A3C agents will always operate under sub-optimal conditions. A2C stands for *Advantage Actor-Critic* which eliminates the asynchronous characteristic of A3C. The inconsistency of A3C is dealt with by adding a coordinator to the multiple agents of the system, who awaits until all the agents have finished their respective tasks to update the *Global Network* and resets the agents such that all of them operate under the same (updated) policy. Allegedly, the synchronisation of the agents maintains the gradient updates cohesive by defining a focus on the training process and potentially shortening convergence time [75]. Some studies have not identified any evidence of benefit to the performance of the algorithm due to the asynchrony of updates; On the contrary, it has been observed that the synchronous implementation is more cost-effective when using a single-GPU computer, and performs better (than A3C) when large batch sizes are used [75].

Figure 3.22 shows how the synchronous architecture of A2C operates compared to A3C. The key components of A3C are:

1. **On-policy** formulation for reduced variance and faster training times (if compared to general Off-policy methods)
2. **Advantage function** to reduce variance
3. Introduction of **parallel training** of multiple agents, for both policy and value function estimation. Updating scheme based o fixed lengths of time (or amount of gained experience)
4. **Synchronous** updates for increased computational effectiveness and performance

Policy Optimisation

The methods herein described are referred to as *policy optimisation* (PO) because they tackle policy estimation directly. They are related to DDPG because they also use policy gradient for optimisation

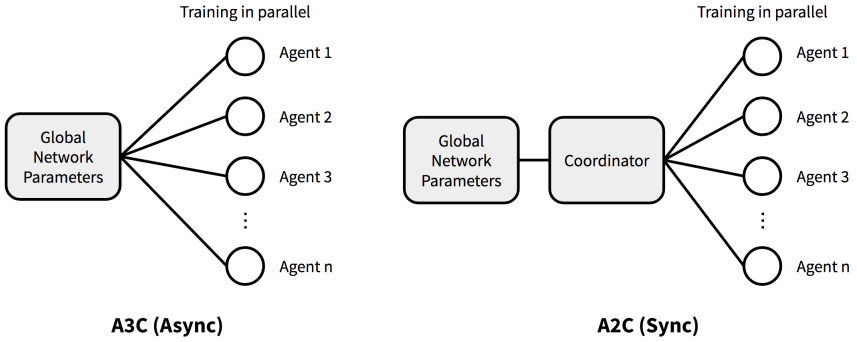


Figure 3.22: How parallel actors are combined into a single global network in A3C and A2C, retrieved from [73]

as described in equation 3.17. Although, unlike those derived from DDPG, PO methods fall under the *on-policy* category, that is because their behaviour policy is the same as their estimation policy. This section of this chapter discusses two of these methods, namely, Trust Region Policy Optimisation (TRPO) and its improved version Proximal Policy Optimisation (PPO).

TRPO

TRPO addresses the issue relating to gradient ascent (or descent) that policy gradient methods present. Gradient ascent is a line optimisation method and generally uses a first-order optimiser, which generally works well, except in curved areas of a graph, where accuracy significantly decreases. This might negatively affect the training process and make it longer and more complex, however, there are other optimisation methods available. TRPO stands for Trust Region Policy Optimisation [58], and it utilises a gradient-based optimisation method called Trust Region optimisation. TRPO involves three concepts: Trust Region Optimisation, KL divergence, and importance sampling.

In simple terms, line search optimisation first finds the best direction of a function to explore and then moves in that direction with a step size a. In *trust region optimisation* first a step size a is defined from points p, this is named the *trust region*, and then a search for a local minima/maxima is done in all points within the radius a from p_0. Once found, the local optima becomes point p_1 and the process repeats itself. In the traditional TRO step size, a might vary from point to point depending on how accurate the approximator is. In RL, the size of the trust region might vary depending on how much the new policy diverges from the old policy, resulting in better-reduced variance and more cohesive training. Ideally, the largest step size possible would be taken, to reduce the chances of the algorithm getting stuck in local optima, as PG methods do, but this might result in large changes in the policies and too many points to compute. Hence, TRPO utilises KL divergence to define constraints to the size of the step taken. KL divergence estimates the difference between an old and estimated probability distribution, implementing a constraint that ensures small changes to the policy.

Equation 3.20 shows the objective function TRPO uses using the policy ratio that is bound by the equation in 3.21, where δ is a constraining value.

$$J_{TRPO}(\theta) = \mathbb{E}[\frac{\pi_\theta(a,s)}{\pi_{\theta_{old}}(a,s)} A_{\pi_{old}}(s,a)] \tag{3.20}$$

$$\mathbb{E}[D_{KL}(\pi_{old}(.|a)||\pi_{old}(.|s))] \leq \delta \tag{3.21}$$

Additionally, TRPO deals with the sample inefficiency of PG methods by implementing a technique called *importance sampling*. A clear explanation of how this is done falls outside of the scope of this project. However, it suffices to state that instead of discarding samples from old policies, as PG methods do, importance sampling still uses them. This only works because TRPO only allows small changes to the policy at a time (KL constraints), through *trust regions*, meaning that samples from different policies are still somewhat related and valid for future training; However, eventually the policies change to a degree that the samples are no longer similar, these should then be discarded.

The key components of TRPO are:

1. **On-policy** formulation for reduced variance and faster training times (if compared to general Off-policy methods)

2. **Trust Region** to ensure small changes to the policy
3. **Importance sampling** to increase sample efficiency of PG methods (and DRL in general)

PPO

Despite good performance, TRPO has the downside of using complex ideas and algorithms in its operation. PPO counters that by using a clipped surrogate objective to simplify the implementation whilst retaining similar performance to TRPO. Proximal Policy Optimisation (PPO) was introduced by the same author as the TRPO algorithm and it is a first-order optimisation that stipulates a probability ratio $r(\theta)$ between the old and new policies [57]. The use of simple probability ratios instead of complicated KL constraints renders PPO much simpler implementation, hence, allowing a wider range of applications and researchers to use the algorithm.

Equation 3.22 shows the probability ratio and equation 3.23 shows how $r(\theta)$ is implemented in the objective function of PPO. It is important to note that $r(\theta)$ is constrained in PPO by a clipped surrogate from -1 to $+1$, instead of the complex KL divergence in TRPO.

$$r(\theta) = \frac{\pi_\theta(a,s)}{\pi_{\theta_{old}}(a,s)} \tag{3.22}$$

$$J_{TRPO}(\theta) = \mathbb{E}[r(\theta)A_{\pi_{old}}(s,a) - c_1(V_\theta(s) - V(s)) + c_2 H(s,\pi_\theta(.)))] \tag{3.23}$$

Additionally, PPO subtracts the error value of the value function with the target value function, parameterised by the hyperparameter c_1, from the objective function, and adds an entropy term to it, parameterised by the hyperparameter c_2. Compared to TRPO, PPO has a simpler implementation, more generality and has proven empirically to have better sample complexity. In [38] a study of different RL algorithms for learning to control UAVs is made, in its results it shows that PPO surpasses PID performance in almost every metric that has been analysed.

The key components of PPO are:

1. **On-policy** formulation for reduced variance and faster training times (if compared to general Off-policy methods)
2. **Clipped surrogate** to ensure only small changes to the policy occur, in a simple fashion

Synopsis

The information given in the previous subsections is provided in an organised and simple manner to be used for reference in table 3.4.

Table 3.4: (Deep) Reinforcement Learning Actor-Critic algorithms

Algorithm	Policy	Policy Type	Action Space	State Space	Additional characteristic
DDPG	Off-Policy	Deterministic	C	D & C	Deterministic policy
TD3	Off-Policy	Deterministic	C	D & C	Double Q-value & Delayed updates
SAC	Off-Policy	Stochastic	C	D & C	Stochastic policy
A3C/A2C	On-Policy	Stochastic	D & C	D & C	Parallel training
TRPO	On-Policy	Stochastic	D & C	D & C	Trust Region Optimisation
PPO	On-Policy	Stochastic	D & C	D & C	Clipped surrogate

In columns four and five C stands for*continuous* and D for *deterministic*

Actor-critic DRL methods are considered in this literature study because they are mainly model-free, meaning that they do not require any prior knowledge of the environment. This is a characteristic that is desired in an automatic landing system, this is because current ALSs fail to perform as expected when the environment changes, i.e. dynamic, which is inevitable in aviation. It is understood that if the controller's operations do not depend on its idea of the environment, i.e. model, it is less likely that its performance is considerably affected by changes in the environment. A downside of the algorithms herein presented is the fact that they are in general sample inefficient, if compared to other methods. Hence not suitable for *online* implementation, which as mentioned previously, is a desired characteristic. They have, on the other hand, been very successful through *offline* implementation due to their high generalisation power. For these reasons, DRL methods are under consideration to be used in this project.

3.2.5. Proposed Framework

In the previous sections of this chapter two groups of algorithms contain many of the characteristics that are desired for the algorithm to be sued in the auto-land system to be designed. Additionally, both groups have already been successfully implemented in flight control applications with state-of-the-art performance. The main difference between the two groups is that iADP methods are implemented in an online fashion as adaptive controllers, on the other hand, model-free DRL methods are mainly applied in an offline fashion as robust controllers. Hence, the question becomes a matter of deciding between adaptive and robust control, more than between the methods themselves.

In [2] a comparison between robust and adaptive control is made with a variety of examples and in different structures. The results of this specific paper in combination with previous studies conclude that, in general terms, robust control tends to present faster responses to variations in parameters, in the case of this study, that would be dynamic changes to the environment. However, these responses are satisfactory only if these variations fall within the design specifications, in the case of RL that would mean within the trained situations. On the other hand, adaptive control, usually provides slower responses to variations in a system, however, is capable of performing well in a wider variety of circumstances, as well as adapting to unknown situations. The paper proposes these two different approaches to be seen as complementary ways of dealing with system uncertainties, rather than opposite. Robust control is designed to be insensitive to parameter variations, and adaptive control is designed to change its control law to better suit the new parameters. Therefore, it is interesting to combine these two characteristics in a single controller, such to yield a control method that inherits both characteristics.

The field of iADP presents algorithms that are capable of learning in a fully online manner, without the need for an offline training phase, due to its high sample efficiency, high convergence rate and the utilisation of a model of the system. Additionally, the use of a fast and accurate incremental model enables model independence of the agent, which is a highly desired characteristic to deal with uncertain environments such as that of aircraft landing. Additionally, the adaptive nature of iADP renders it generally higher fault tolerance if compared to DRL methods, which rely on generalisation power. However, as pointed out by [68], adaptive methods have two drawbacks related to their online implementation, first continual learning may lead to a monotonic increase in the magnitude of network parameters, which leads to instability of the controller [28], and second, present high sensitivity to real-world effects, like sensor noise, leading to degraded performance of the controller. On the other hand DRL methods, albeit having performance limited by their offline training, present insensitivity to changes and noisy data, for example, sensor noise, and generally faster responses. Additionally, robustness may be increased through training the agents in the presence of failure/disturbance, such that it experiences the environment changes and learns how to work with them.

Finally, in view of the characteristics of iADP and DRL methods, the current needs of ALS and the intermediary results of previous research, discussed in the literature study, it is concluded that **robust control** is the most interesting choice for an automatic landing system at the moment and the one that promises the most stable base for future iterations. Therefore, it is proposed to study the implementation of a framework that makes use of DRLs method as a means of making ALSs more robust and increasing their control accuracy. The exact structure of the controller to be used is discussed to be decided after the preliminary analysis, 4, in chapter 5.

3.2.6. Conclusions

The literature study presented in this chapter has the aim of introducing the topic of reinforcement learning, such as to create a basis for the understanding of how it can be applied to control systems and how that would benefit automatic landing systems. This chapter presents basic RL concepts such as its elements, Markov decision processes, the Bellman equation, and design choices, such as exploration and exploitation, model-based and model-free methods, on-policy and off-policy, online and offline etc. Furthermore, three basic solution approaches, namely Dynamic Programming, Monte Carlo and Temporal Difference methods, were discussed such to set the basis for understanding more specific methods. The discussion then shifts towards the implementation of RL in the real-life world through adaptations for continuous spaces and the challenges faced in the field of RL.

State-of-the-art algorithms belonging to two families, namely ADP and model-free actor-critic DRL, have been presented in terms of their workings, their characteristics, strengths and weaknesses. It has also been decided to study the development and implementation of a controller that combines

both robust and adaptive control techniques. From the literature study, it is clear that IDHP is the most prominent algorithm regarding the implementation of flight control techniques and model-independent controllers to continuous problems in general, on the side of adaptive control. On the side of robust control, it was not clear from the literature which of the algorithms provides the most benefits. For example, [17] utilises SAC for its flight controller, but recommends analysing the performance of TD3 and PPO, additionally, [67] was successful in implementing DDPG to an automatic landing system. Hence, it remains to be decided which of the DRL methods is the most suitable for the application of this thesis.

With the content presented in this chapter, it is possible to answer research questions Q2.1 and Q2.2.

RQ2.1 What are the current state-of-the-art RL algorithms?

As stated by [64] current state-of-the-art algorithms utilise actor-critic agent structures and are likely to remain of interest due to their ability to learn explicitly stochastic policies and requiring minimal computational effort in action selection. Furthermore, this question is answered in sections 3.2.2 and 3.2.3, where ADP Adaptive-Critic Designs and model-free actor-critic DRL methods are described along with their main characteristics, strengths and weaknesses. Furthermore, it has been pointed out that some of these methods have already been successfully applied to Flight Controllers and similar applications. A more clear answer is given in section 3.2.2 where eight papers have been briefly discussed, namely, HDP, DHP, IDHP, IGDHP from the field of iADP, and SAC, TD3, DDPG from the field of DRL, and even a mix of both fields, through a SAC-IDHP hybrid, were implemented for flight control applications.

RQ2.2 Which RL framework is suitable for resolving ALSs issues?

As stated in the answer to RQ2.1 there are two groups of algorithms that are interesting for the goals of this project. However, the comparison between the two is not a straightforward one, this is because iADP produces an *adaptive controller* and DRL produces a *robust controller*. According to [2], robust controllers provide fault-tolerance due to their high generalisation power and are expected to work well in a dynamic and noisy environment, as long as the changes are not too extreme. Adaptive controllers provide adaptive behaviour, where the parameters that govern its policies are changing with the environment, therefore, these controllers are expected to stand out in environments where there are exceptionally large. For this research, the robust approach was chosen due to the fact that it provides a more solid controller, such as to make a stable and robust base for an ALS that is insensitive to sensor noise and generates faster responses. It is also not expected that the changes in the environment are too large such that the DRL controllers cannot handle them, however, this can only be affirmed with certainty at the end of this research.

It is acknowledged herein, however, that possibly the best controller design is a combination of both robust *and* adaptive controllers [2], where relatively small changes to the environment are handled by the former and larger changes by the latter. However, it might not be necessary to implement an adaptive control portion to the ALS controller. Therefore, the research herein presented chooses to focus on robust control, and the study on the addition of an adaptive control portion to the ALS is left for future research in case it is shown necessary.

Finally, it is not clear from the literature study which model-free deep reinforcement learning algorithm is the most suitable for the purposes of this research. Previous research was successful in using the SAC algorithm in fault-tolerant flight control [17] [68], however, both of them recommended an investigation on the use of TD3 and PPO, which also appeared in this research as part of the state-of-the-art in RL algorithms. Therefore, a preliminary analysis of such methods is performed in section 4 to determine the most suitable DRL algorithm for the ALS developed in this research.

4

Preliminary Analysis

Chapter 3 introduced the topics of (automatic) landing systems and reinforcement learning, from the conclusions of these chapters research questions *RQ1.1-3* and *RQ2.1-2* have been answered. To attend to the needs of current ALSs, at the end of the last chapter a control framework utilising *robust* control through *offline learning* has been proposed to be studied in this research. However, it was not possible to determine which model-free actor-critic DRL algorithm is the most suitable for the ALS purposes. Therefore, a preliminary analysis is performed to identify which algorithm performs the best under a simplified flight control task and answer $RQ2.3$. Additionally, this preliminary study allows for familiarisation with the implementation of DRL algorithms and how they work.

This chapter is outlined as follows, section 4.1 defines the reasoning on why the preliminary study herein presented is performed, along with the objective and the structure of two tests, namely *robustness test* and *adaptability test*. Section 4.2 presents the results and the discussion of the findings of the tests performed. Section 4.3 presents the structure and the goal of an additional test performed to verify the training consistency of certain algorithms, along with an analysis of the results and findings. Finally, section 4.4 briefly summarises the findings of the chapter and combines information acquired previously, in chapter 3, to fully answer research question *RQ2.3*. [1]

4.1. Testing objective & Structure

In the previous subsections, seven different model-free DRL Actor-Critic algorithms have been discussed, including their main characteristics, the motivation as to why they have been developed, as well as their working principles. As has been discussed in section 3.2.2 (Design-choices & High-level concepts), choosing an algorithm is no easy task, as each design comes within a spectrum and each method has strengths and shortcomings. The preliminary study herein presented is meant to aid the algorithm selection process and has the goal of determining which of the model-free DRL algorithms is the most suitable to be used in the development of an automatic landing system, and to provide concrete arguments to back-up this choice. The preliminary study is performed using five algorithms discussed in chapter 3.2. DDPG, TD3 and SAC are tested in order to verify the promised improvements the latter two algorithms have over the former, A2C since according to [75] A2C has clear advantages to A3C, and finally PPO, since according to the author of both [57] PPO is clearly better than TRPO. Plus an additional PID controller to verify the argument of classical control theory being unsuited for highly dynamic environments.

There is a wide variety of methods for bench-marking algorithm performance, perhaps the most widely known bench-marking toolkit for continuous control and physics-based simulation in reinforcement learning is OpenAI *Gym*. The library contains environments for developing, testing and training RL agents, ranging from a long list of Atari games (Asteroids, Defender, Enduro, etc.) to classic control problems (Acrobot, Cart-pole, Pendulum, etc.). For this preliminary analysis perhaps the most fitting is the classic rocket trajectory optimization problem *"Lunar Lander"* [36] is chosen, a simple flight control problem where the goal is to *land* a spacecraft.

[1] *Chapter 4 - Preliminary Analysis* of the report has already been assessed as part of the *AE4020 Literature Study* course.

4.1.1. Problem introduction

The Lunar Lander problem consists of a spacecraft landing on the rugged surface of the moon and the goal is to land in a landing pad located at $(0, 0)$, as shown in figure 4.1. The Reinforcement Learning structure is set with an *Action space*, *Observation Space*, *Rewards*, and *State Space* as follows:

- The action space consists of four discrete actions: (i) do nothing, (ii) fire the main engine, (iii) fire the left engine, and (iv) fire the right engine, of which (iii) and (iv) cannot be performed at the same time. In the discrete form of the problem the engines can only be *on* or *off*, this analysis however uses the continuous version of the environment, where the engines may be activated with intensity varying from 50% to 100% power, at the agent's will.
- The observation space is an 8 dimension vector containing: The craft's position (x and y direction), its linear velocity (x and y direction), its angle, its angular velocity, and two booleans informing whether the craft's legs are in contact with the ground or not.
- The positive rewards are given when the craft reaches the pad (100 to 140) when the craft comes to a rest (100), when each leg touches the ground ($2x10$), and negative rewards, punishment, is given when the craft moves away from the pad (*not specified*), when the craft crashes (-100), when it fires the engines (-0.3 per frame for the main engine and -0.03 for side engines), the problem is considered solved if a total of 200 reward points is accumulated at the end of the episode.
- The state space is continuous and the problem is initialised with the craft in the top centre of the environment and a force of random magnitude and direction is applied to its centre of gravity.

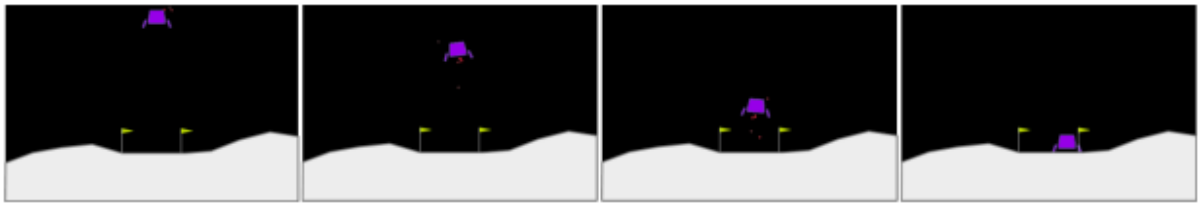


Figure 4.1: Snapshots of the spacecraft landing in the lunar lander problem, retrieved from [22]

The Gym library also provides a *wind function*, with which simulated wind gusts and turbulence can be applied to the craft during the landing. This feature plays an essential role in this study, this is because it adds a simulated atmosphere to the Lunar Lander problem, allowing for disturbances such as those found in the landing environment to be simulated and for more accurate testing of the robustness of the algorithms. Therefore, the landing task herein analysed can be considered a simple flight control problem. The wind gusts provided by the library are modelled through the trigonometrical function in equation 4.3 where the wind magnitude W_{mag} is computed through the summation of two sinusoidal functions, limited by a hyperbolic tangent multiplied by a wind power W_{power}. The two sinusoidal functions are given in equations 4.1 and 4.2, where k is set to 0.01, t is the time, and C is randomly chosen between -9999 and 9999, rendering, according to [36], a non-periodic trigonometrical function.

$$\theta_1 = 2k(t + C) \tag{4.1}$$

$$\theta_1 = \pi k(t + C) \tag{4.2}$$

$$W_{mag} = W_{power} * tanh(sin(\theta_1) + sin(\theta_2)) \tag{4.3}$$

4.1.2. Tests set-up

Having in mind the nature of the proposed framework two series of tests have been designed with the goal of determining which of the DRL algorithms is the most suitable choice for the ALS to be designed. The tests have the ultimate goal of measuring the robustness of each algorithm, which is to be done through an offline implementation of algorithms and analysis of their performance in the simple flight control task described in the previous section.

The first test has the goal of analysing how the agents are capable of dealing with unknown disturbances, that is, disturbances that it has never been experienced before. Therefore the test is structured

as follows: the agents are trained in the environment until it is considered that they are capable of ”solving it”, i.e. until they are able to accumulate 200 reward points, according to gym [36]. In order to avoid prematurely truncated training due to outliers and exceptional performances, a script is written such that the agent is trained until it can achieve a reward average equal to or higher than 200 in 100 episodes. The agents are then evaluated in 1000 episodes in order to establish their baseline performance in the uninfluenced environment, i.e. identical to the one they have been trained in. Subsequently, the agents are subject to an additional evaluation of 1000 episodes, where external disturbances in the form of wind gusts, such as explained in 4.1.1, are introduced in the environment to simulate uncertainties. The performance difference between the two sets of evaluations is then used to determine the robustness, consistency and computational effort of the algorithms. This first test is designed to show the robustness of the algorithms when an unknown disturbance is introduced in a stable environment, hence it tests the **robustness to the unknown** of the algorithms.

In reality, it is known that the vast majority of external disturbances come in the form of *constant winds*, *wind gusts* and *turbulence*, all of which can be modelled. The environmental uncertainties, in this case, are related to the shape and intensity that these disturbances have. Additionally, there are multiple failure cases that are known and can be simulated, the uncertainties appear in the form of the intensity of the failure and the moment in time they occur. Hence, agents can, and should, also be trained with certain models of disturbances and their performance analysed under yet another modelled disturbance.

The second round of tests is then designed with the goal of analysing how capable the agent is of dealing with known but uncertain circumstances, that is, unknown disturbances that are similar to those the agent has seen before, although not entirely the same. In this case, the agents are trained in the environment where wind gusts with $w_{power} = 10$ are present, with the same method in the first test, and evaluations are made for 1000 episodes in environments with $w_{power} = 10$ - to determine baseline performance -, $w_{power} = 15$, and $w_{power} = 20$. It is expected that the agents have worse baseline performance than in the previous test, but also that they learn how to deal with uncertainties, hence perform better in the subsequent rounds of evaluation, even if the disturbances are different than those they have been trained in. This test also evaluates the robustness of the algorithms, in this case however, it measures the **robustness to uncertainties**, in contrast to a more pure robustness test performed in the first test.

Regarding training, parameters of interest are the *training time* and the *time steps* required to reach the desired training level, such as to gather information about the computational necessities of each algorithm. Additionally, a learning curve provides information about the stability of training, which is considered an issue for some DRL algorithms. Regarding the tests themselves, parameters of interest are *average rewards* and *standard deviation* of the results. The former parameter measures the overall performance of the agents, hence the higher the better, it is an important parameter because it ultimately indicates the algorithm that performs the best. The latter measures the consistency of the agents between episodes, hence lower deviations are desired, additionally, it also gives information about the reliability of the algorithms, which is of extreme importance in fault-tolerant systems which involve high-risk, such as aviation systems. Other parameters have also been computed reflecting the computational needs of the algorithms, however, these are less important due to the fact that the agents are trained in an offline manner and that the computation time does not largely differentiate from one method to another, as can bee seen in the results table. Nonetheless, the following parameters can be found in tables B.1 and B.2, in appendix B: steps per episode to evaluate the quality of actions, and computation time per episode and per step to evaluate computational expense have also been computed.

4.2. Results & Discussion

The main algorithms previously described, namely, DDPG, TD3, SAC, A2C and PPO, have been implemented using an RL library called *Stable-Baselines*, which provides standard versions of such algorithms. It is important to note that in order to maintain consistency across the algorithms' implementation, no hyper-parameters have been modified and with the exception of DDPG and TD3 utilising noisy actions to increase exploration (as advised in the library documents), the algorithms have been implemented in their most basic form. Additionally, the PID controller has been designed using a *hill climbing optimisation* technique, where the algorithms start with no control, i.e. PID parameters equal to

0, and small random increments are used to alter the parameters every step, which are only effectively updated in case the performance is indeed enhanced from one step to the next.

4.2.1. Test 1 - Robustness to the unknown

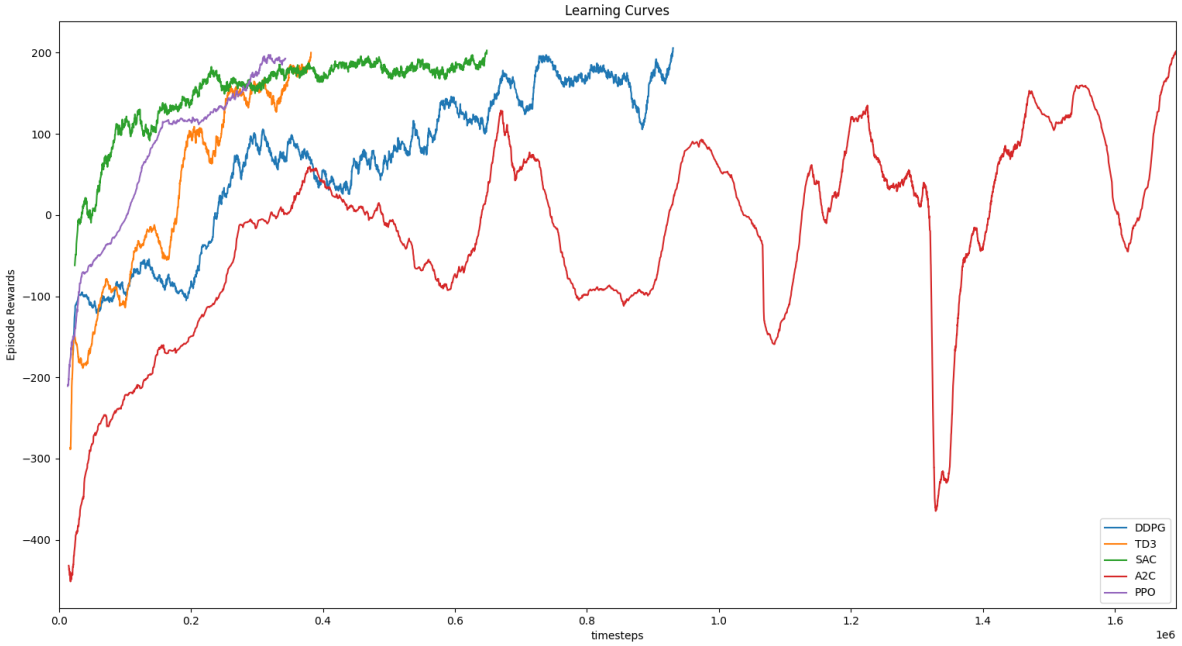


Figure 4.2: DDPG, TD3, SAC, A2C, PPO learning curves

Table 4.1: Actor-Critic DRL algorithms training performance in the Lunar Lander environment

Algorithm	Training time [min]	Time steps
PID	-	-
DDPG	275.2	738000
TD3	70.7	238000
SAC	184.7	774000
A2C	92.6	2084000
PPO	15.5	209000

The results for the training are shown in table 4.1. It is possible to see that DDPG is the slowest algorithm to converge, and as expected both modifications of it, i.e. TD3 and SAC, have faster convergence time. Additionally, PPO is by far the fastest of them all to converge, also requiring fewer steps to do so. A2C requires by far the most amount of steps but sits at the median of the training times, meaning that it has the fastest computation time for each step. Additionally, TD3 is 2.7 times faster than SAC and requires 3.2 times fewer steps to converge, inferring that not only TD3 requires fewer steps, but its steps are also faster to compute. Figure 4.2 shows the learning curve graph of the algorithms, where it is possible to see that SAC learns the fastest in the initial steps, however, becomes more conservative training progresses and better parameters have been found, ultimately requiring more steps than TD3 and PPO. Finally, from the graph it is possible to see that A2C has the highest training variance, seemingly not converging to a stable final agent, but only staying within the desired reward range just long enough to trigger the training to stop. The tuning time for the PID controller is not computed, this is because in the hill climbing methodology, the initial parameters are initially randomly sampled, therefore training may take up to 1 hour as well as a few seconds.

Table 4.2: Actor-Critic DRL algorithms performance in 1000 episodes of the Lunar Lander problem trained with no wind and evaluated in nominal condition and with wind, $w_{power} = 15$

	Nominal Condition		Added Wind	
Algorithm	Mean reward	Standard deviation	Mean reward	Standard deviation
PID	252.6	62.9	57.8	175.3
DDPG	278.4	31.4	265.1	53.5
TD3	280.0	18.1	265.4	38.1
SAC	279.5	16.0	270.1	31.3
A2C	221.3	39.5	182.3	110.5
PPO	220.2	82.1	151.2	135.3

Table 4.2 shows the averaged results of the algorithm's performance over 1000 episodes in the environment, first under nominal conditions, i.e. without wind, and then with the wind. The PID controller achieves rewards well above 200 points in the nominal conditions, whilst A2C and PPO average around 30 points less than the classic control method. DDPG and its two variants, on the other hand, surpass the performance of the PID averaging around 30 points more than the PID. Additionally, TD3 and SAC are significantly more consistent than the other algorithms presenting two to four times better standard deviation among the evaluations. When wind is introduced in the environment, however, the performance difference between the DDPG, TD3 and SAC, and the other algorithms becomes even clearer. The PID scores the least points, followed by A2C and PPO, all three of which are not able to reach an average of 200 reward points. The performance of the policy gradient methods on the other hand only decreases about 10 to 15 reward points from the nominal conditions, therefore showing the robustness of these algorithms. Additionally, once again TD3 and SAC show significantly less divergence than DDPG does, being SAC's divergence still smaller than that of TD3.

4.2.2. Test 2 - Robustness to uncertainties

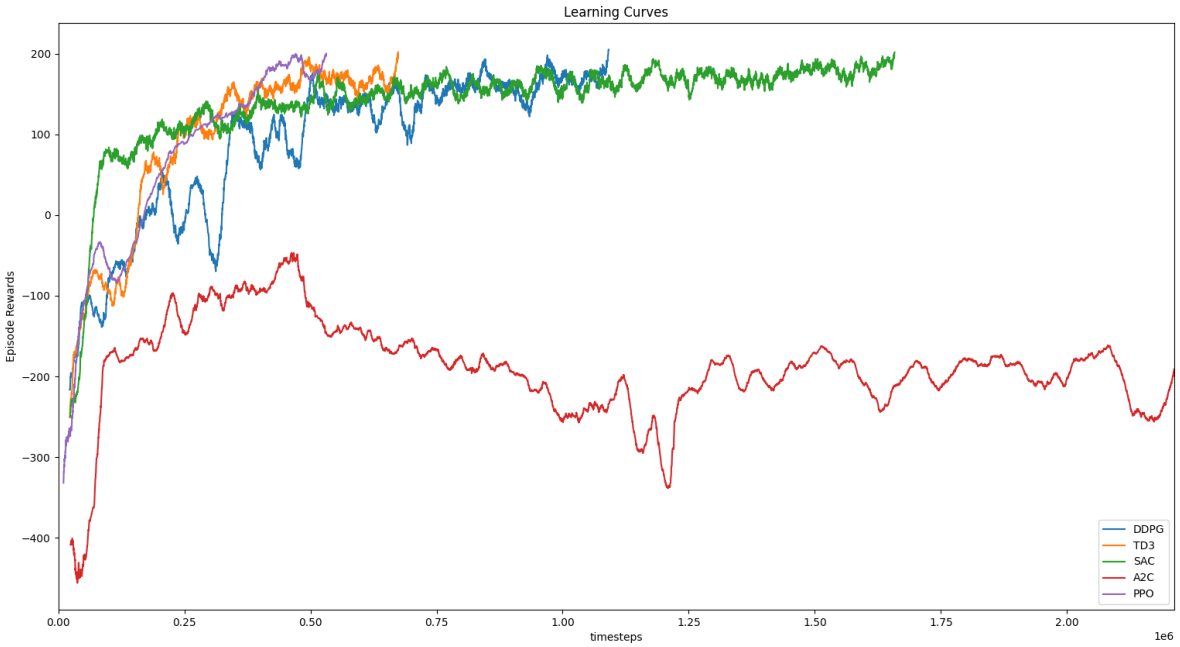


Figure 4.3: TD3, SAC, PPO learning curves

Training under uncertain circumstances exhibits similar results to that in a stable environment when the performance of the algorithms is compared. PPO is still the fastest to converge, followed by TD3 just as before; Surprisingly however, is that SAC has a training time twice as long as DDPG, requiring almost three times more steps to converge. The same phenomenon as in the previous training is observed, SAC is the fastest to learn at the start and it slows down after reaching higher rewards. DDPG is still

Table 4.3: Actor-Critic DRL algorithms training performance in the windy Lunar Lander environment ($w_{power} = 10$)

Algorithm	Training time [min]	Time steps
PID	-	-
DDPG	330.5	896000
TD3	295.8	874000
SAC	655.1	2239000
A2C	-	-
PPO	25.8	302000

slower than TD3, and unlike the other algorithms, it presents a similar training time as it did when there was no wind in the environment. A2C presented extremely high training instability, not even being able to reach positive rewards, hence it does not have complete training data in table 4.3; This might be explained by the multiple parallel environments the algorithms use not being sufficiently correlated due to high levels of uncertainty, and hence hindering the learning ability of the agents. Overall, the algorithms required much longer training times to reach 200 reward points consistently, up to three or four times what has been previously observed and require about three times more steps. The training data is shown in table 4.3 and the learning curve of the algorithms in figure 4.3.

Table 4.4: Actor-Critic DRL algorithms performance in 1000 episodes of the Lunar Lander problem trained with $w_{power} = 10$ and evaluated with wind, $w_{power} = 10$, $w_{power} = 15$, and $w_{power} = 20$

	Wind Power 10		Wind Power 15		Wind Power 20	
Algorithm	Mean reward	Standard deviation	Mean reward	Standard deviation	Mean reward	Standard deviation
PID	159.7	127.8	127.0	129.8	54.83	90.5
DDPG	271.8	43.5	270.7	41.7	265.5	48.6
TD3	276.9	20.3	274.5	24.8	270.3	30.0
SAC	283.6	19.1	279.7	20.1	276.5	26.0
A2C	-	-	-	-	-	-
PPO	195.1	108.4	156.1	131.6	150.2	135.1

Table 4.4 shows the averaged results of the performance of wind-trained agents over 1000 episodes in the environment with wind, first with $w_{power} = 10$, i.e the environment the agents have been trained in, second with $w_{power} = 15$, and third with $w_{power} = 20$. In this test the PID was tuned using the same hill climbing technique as before, but with the presence of the correspondent wind conditions. It is observed that the PID performance considerably increased when the wind is taken into consideration during the tuning process. Albeit a clear improvement is noticed, unfortunately, the PID is still not able to achieve sufficient results under windy conditions, which corroborates the statement that classical control theory is not suitable for uncertain systems. Additionally, PPO retains performance similar to those encountered in the first test, however, it also shows that the algorithm is not robust enough to handle uncertain windy conditions. Note that the algorithm was not capable of reaching an average of 200 reward points, not even when tested in the conditions it was trained in.

Finally, similarly to what had been observed before, the DDPG variants are the best-performing algorithms under windy conditions, presenting by far the highest amount of reward points. The three algorithms have similar performance regarding the accumulated rewards, diverging only by $1 - 3\%$ among them, being SAC performing slightly better in all situations, followed by TD3 and then DDPG. The latter, however, presents significantly lower consistency in its results, at times presenting standard deviations twice as high as its other variations. Additionally, it is worth mentioning that the SAC-trained agent presented the lowest amount of steps per episode in the three measured wind conditions (See table B.2 in appendix B), which not only renders the algorithm higher scores but also that the SAC agent can find better solutions, that is faster, than other algorithms.

4.2.3. Discussion

The performed tests present insightful results on the performance of the algorithms. It reaffirms the fact that PIDs are not suitable for unstable, non-linear and complex systems, hence have limited performance when applied to the uncertain automatic landing problem herein considered. Moreover, A2C showed incredibly unstable training, especially when it is attempted to train under uncertain conditions, and it showed a lack of robustness toward external disturbances. PPO showed incredibly fast convergence and beat other contestants with respect to training time by a large margin; More importantly, however, the algorithm was not able to show consistent results as well as lacking robustness. Therefore, A2C and PPO are ruled out to be used in the upcoming phase of this project. More interestingly, however, is the performance of DDPG and its two variants, TD3 and SAC, these algorithms showed high performance in the tests by accumulating high amounts of rewards in all performed evaluations, whilst still maintaining low deviation between episodes. Although, DDPG has significantly higher deviations if compared to its other two counterparts, and slightly lower averaged rewards, and therefore it is also disconsidered for further use.

Through these tests it is clear that the SAC and TD3 are the two best-performing algorithms, hence the most suitable for the automatic landing system to be developed in the main phase of this thesis. It is, however, not straightforward to determine which of the two is the most suitable one. SAC performs slightly better than TD3 in all metrics in the second experiment. TD3 on the other hand has more efficient training, requiring fewer steps with lower computational expense to converge to stable agents performances. The study in [17], where SAC was implemented in a flight control system, indicates that the algorithm presents low training reliability due to low stability, and recommends the investigation of a TD3-based controller to solve such problem. Additionally, the studies performed by the authors of the respective algorithms are inconclusive as to which one performs best since they have been published in the same year with a short period of time between each other. Their results also present biases due to a lack of parameter tuning, stochasticity and the use of different network characteristics.

In summary, SAC has slightly better performance than TD3 and can find better solutions by solving the lunar lander problem with fewer steps per episode. However, having in mind the issues with SAC pointed out by in [17], an extra experiment was designed to analyse the training consistency of SAC and TD3 and evaluate if training consistency is a decisive factor.

4.3. Training Consistency Test & Discussion

Other important metrics that have not been considered in the previous tests are the consistency of training, that is the consistency of which the agents are capable of learning the behaviours presented in the data in the tests and the stability of training. Hence, another test has been designed in which four agents are trained using SAC and TD3 such that the consistency of their results can be analysed. The training and test set-up is the same as the second round of tests in the previous section, that is, the agents are trained under windy conditions with $w_{power} = 10$, and evaluated for 1000 episodes with $w_{power} = 10$, $w_{power} = 15$, and $w_{power} = 20$. The second round of tests has been chosen to be replicated since it most closely resembles real-life training situations.

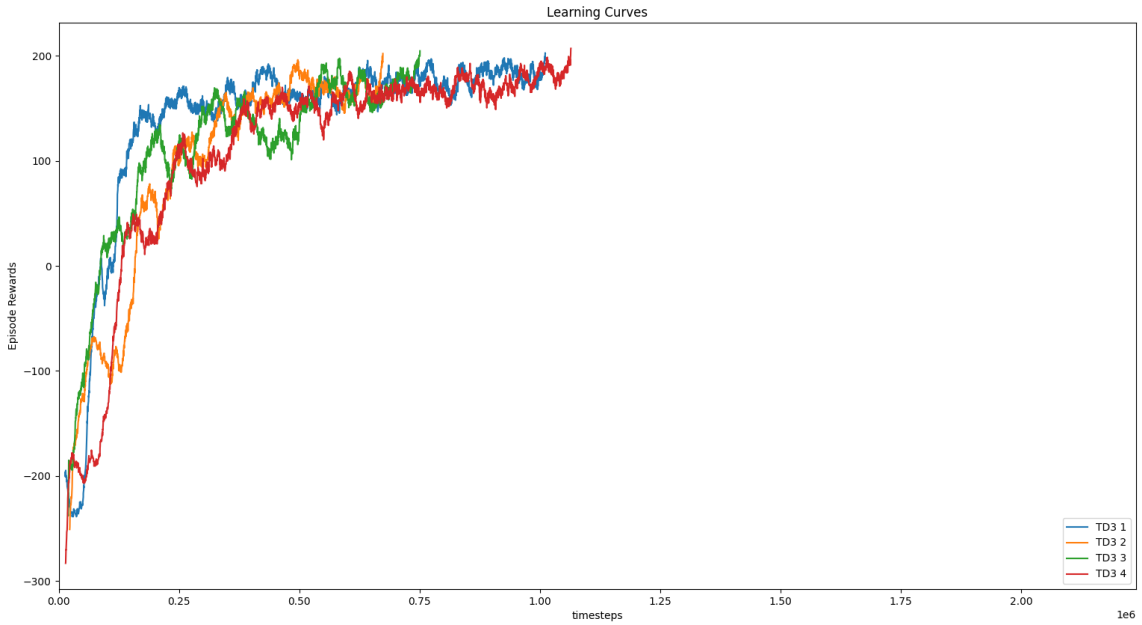


Figure 4.4: 4 TD3 learning curves

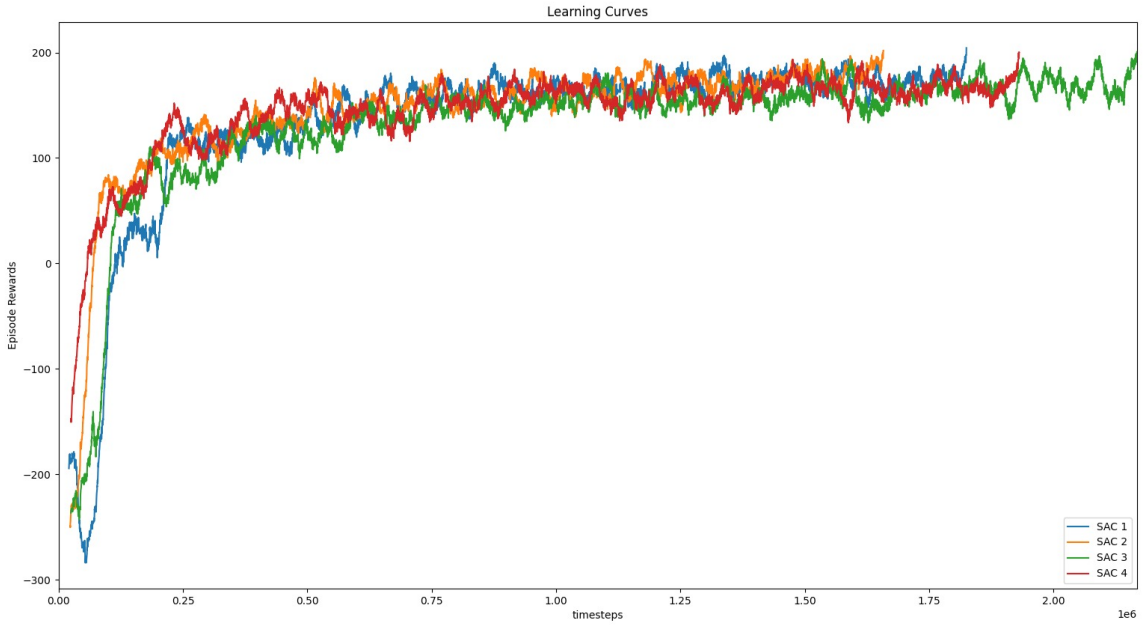


Figure 4.5: 4 SAC learning curves

From figures 4.4 and 4.5 it is possible to see that both algorithms have similar tendencies with a very high learning rate at the start of training and significantly reducing once significant reward levels have been reached. Additionally, it is observed that all curves are close together and present hardly any deviations between an agent and another, meaning that both algorithms seem to have stable and consistent training, and it is expected that different agents would perform similarly. That being said, it is also possible to see that SAC presents more constant learning behaviour than TD3, especially in the initial phase of training, where in the first 250000 steps all TD3 agents present learning drops. Furthermore, SAC presents the four learning curves more tightly together, showing less training variation than TD3.

Table 4.5 contains the average rewards and deviation for each of the agents in 1000 evaluations

in three different wind conditions. Once again it is possible to see that SAC scored consistently higher values than TD3 does. Additionally, SAC is more consistent in its agent's training, which can be seen by the deviation among the results. TD3's higher deviation is due to one of its agents $TD3_3$ performance being much lower than the others. However, the second and third tests $TD3_2$ also scores lower and with higher deviation than the others, an issue that cannot be seen in the first test.

Table 4.5: TD3 and SAC algorithms performance in 1000 episodes of the Lunar Lander problem trained with $w_{power} = 10$ and evaluated with wind, $w_{power} = 10$, $w_{power} = 15$, and $w_{power} = 20$

	Wind Power 10		Wind Power 15		Wind Power 20	
Algorithm	Mean reward	Standard deviation	Mean reward	Standard deviation	Mean reward	Standard deviation
$TD3_1$	276.3	23.8	274.6	24.1	272.4	28.3
$TD3_2$	276.0	24.3	270.9	37.9	261.8	48.9
$TD3_3$	265.4	54.7	267.6	47.6	247.7	80.4
$TD3_4$	278.0	20.0	275.1	24.1	270.7	25.0
Mean	273.9	30.7	272.1	33.4	263.1	45.6
Deviation	4.9	13.9	3.0	9.9	9.7	22.0
SAC_1	283.3	19.8	280.7	20.1	276.5	31.0
SAC_2	278.7	19.2	277.6	22.4	277.4	23.3
SAC_3	280.2	21.9	276.9	24.5	270.6	29.8
SAC_4	277.7	21.2	276.8	23.0	267.4	29.0
Mean	279.9	20.5	278	22.5	272.9	28.3
Deviation	2.1	1.0	1.5	1.5	4.1	2.9

At length, it is possible to conclude that SAC is the most suitable algorithm to be used in the automatic landing system to be designed. This is due to its robustness, proved by averaging higher average scores than other algorithms in all tests performed but one, and the consistency of its performance, proved by averaging lower standard deviation amongst episodes for all tests. Note that the latter point is the decisive factor that sets SAC apart from TD3 (and DDPG), which also showed high-level performance, but a higher deviation of results, both between episodes and agents. The training consistency test presented in this chapter showed that TD3 has more unstable training than SAC, reassuring that SAC is the best choice for the next phase of this project. Additionally, the only drawback to SAC observed in this preliminary study is its training time, which can be much higher than other algorithms; This factor however, does not play such an important role in the implementation of the algorithm, since they are trained offline, and once training is finished, there are no more time constraints holding SAC back. It can also be seen in table B.2, in appendix B, SAC presents the lowest amount of steps per episode, meaning that with roughly the same computational time per step, SAC is capable of having a lower total computational time overall.

The lower amount of steps per episode also shows that SAC probably has learned a better policy than other algorithms. Finally, it is worth mentioning that the tests presented in this section may be of limited value due to the low number of agent samples taken into consideration, more agents could not have been trained due to the time constraints of the project and more accurate results would require more agent data and possibly longer training. On the other hand, the learning curves show that both algorithms have similar trends amongst their agents, and therefore extreme deviations are not expected. Furthermore, the lower performance of $TD3_2$ and $TD3_3$ could not have been predicted from the results of the baseline conditions and analysis of the learning curve.

4.4. Conclusions

The Preliminary study presented in this chapter has the aim of determining which of the model-free actor-critic algorithms is the most robust of them, in other words, which one of them is capable of performing the best under different uncertain circumstances, through generalisation. To determine that, two tests were designed in which agents were evaluated under various wind situations. It was found that the policy gradient methods, i.e. DDPG, TD3 and SAC, perform significantly better than other algorithms. Between them, TD3 and SAC perform better than DDPG and present similar results. To determine which one of the two would be the best choice a third round of tests was designed in which the performance of multiple agents from these two methods was analysed. It has then been concluded that SAC has higher consistency of results amongst different agents, in addition to scoring slightly higher average rewards than TD3 in nearly all performed tests. The findings of this chapter along with the literature study present in chapter 3 allow for research question Q2.3 to be answered.

> RQ2.3 How well does the proposed algorithm perform when applied to simple flight control systems?

The performance of the offline trained SAC controller is evaluated through the implementation of OpenAI Gym's Lunar Lander problem, where an agent must learn how to land a spacecraft within given boundaries. To simulate an atmosphere and the uncertainties that come with it, the influence of wind in different magnitudes was introduced. All analysed algorithms go pass through two tests, to analyse their robustness to the unknown and their robustness to uncertainties, shown in as shown in section 4.2. On the former, the agents are trained in an undisturbed environment, and tested in an environment where disturbances are present. On the latter, the algorithms are trained with disturbances and tested with the same disturbances, but with higher intensity. The Soft Actor-Critic controller presented better overall performance and consistency of results than the five other controllers. Accumulating rewards much higher than required in all situations the algorithm was exposed to, whilst still maintaining lower deviation between episodes. Additionally, an additional test was made on training consistency where the performance across different controllers trained with the same algorithm was analysed. The SAC algorithm showed higher rewards and lower standard deviation in all tested wind conditions. RQ2.3 is then hereby answered.

To completely answer RQ2.2, the most suitable RL algorithm for this research is the offline-trained model-free DRL SAC algorithm. This is because it was shown to be the most robust of the analysed methods and achieved higher rewards in virtually all tests performed in a simple flight control system.

Through the work presented in chapters 3 and 4 research questions **RQ1** and **RQ2** and their sub-questions are answered. Research questions **RQ3** and **RD4** are answered in chapter 2, with additional results in chapter 5, where the SAC controller is designed and implemented on a high-fidelity simulation model of the Cessna Citation 500 aircraft and its robustness tested in different failure cases.

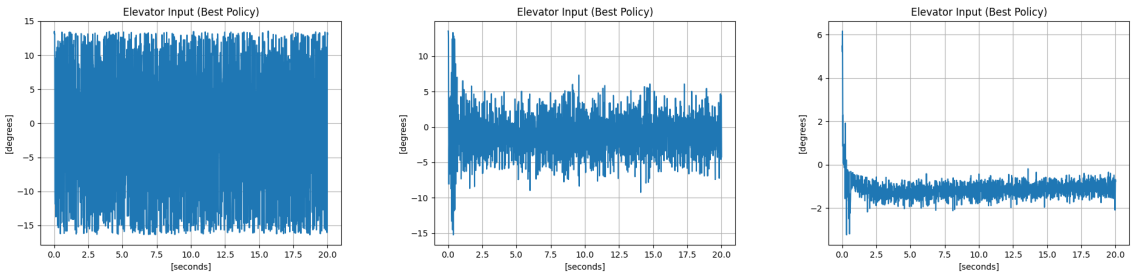
5

Additional Results

This chapter presents additional results regarding the effect of the use of different values for the CAPS Temporal and Spacial parameters. Furthermore, it also presents additional sensor failure cases, where the LOC and GS's transmitted signals are biased. Finally, the use of alternative controllers for the landing task is also explored by means of evaluating a roll controller that has been trained with pitch control active and a full attitude controller.

5.1. Effect of CAPS

As mentioned in section 2 the smoothness regularisation alters the policy with regard to two aspects: (i) sequential actions should not vary greatly from one another, and (ii) similar states should be mapped to similar resulting actions. The authors mention that very likely the temporal parameter would be sufficient to ensure smooth actions, but also that the inclusion of the spatial term could improve policy robustness to domain shifts and unmodelled dynamics, therefore both terms were included in this research.



(a) CAPS40 **(b)** CAPS400 **(c)** CAPS40000

Figure 5.1: Elevator deflection command of controllers with different CAPS with reference signal $\theta_{ref} = -1.7[deg]$

Previous studies on the use of SAC-based controllers tested on the PH-LAB model [68] [59] have used the CAPS parameters with values of $\lambda_T = \lambda_S = 400$. A brief search to evaluate the effects of different CAPS values on the output of the controller was performed. For this experiment, three different agents were trained to follow a fixed pitch reference signal of $\theta_{ref} = -1.7[deg]$, the agents were trained using the same set-up described in 2, being the only difference between them the value of the CAPS coefficients, which were set to $\lambda = 40$, $\lambda = 400$, $\lambda = 40000$. Figures 5.1a, 5.1b and 5.1c show the pitch attitude control command of agents, the output is taken using the training actions, that is, stochastic actions. Increasing CAPS coefficients indeed reduce the variance between subsequent actions, as can be seen from the decrease in oscillation amplitude from one agent to the next. Figure 5.2 shows the differences in the aircraft pitch tracking performance, it can be seen that both the CAPS with $\lambda = 40$ and $\lambda = 400$ track the reference with no steady-state error, however, the CAPS with $\lambda = 400$ presents a much smoother response. The response from the CAPS set to $\lambda = 40000$ is even smoother than that of the $\lambda = 400$, however, it presents a steady state error of around $0.8°$. These results show that indeed

the CAPS parameters set to around $\lambda = 400$ provide the best balance between action smoothness and tracking, therefore, the value of $\lambda = 400$ used in [68] was also used throughout this research.

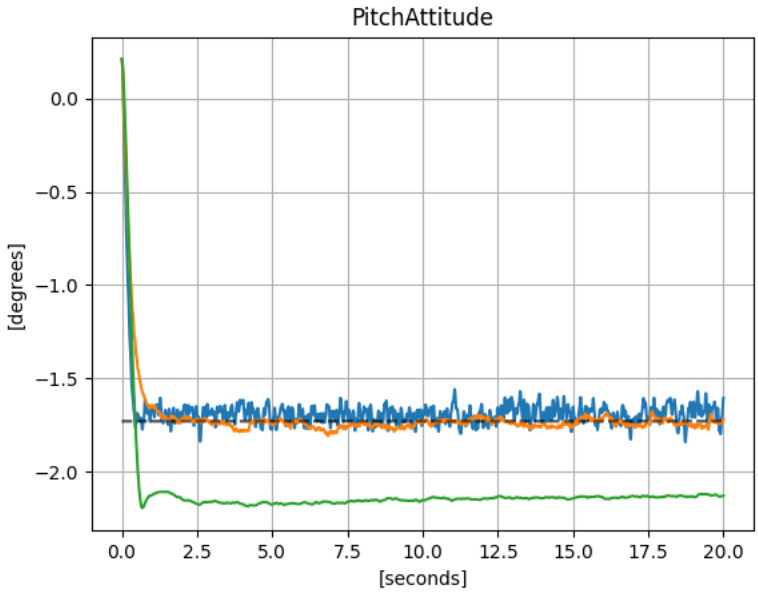


Figure 5.2: Pitch attitude control performance of the controllers with different CAPS. Blue = 40, Orange = 400, Green = 40000

5.2. ILS Bias

Table 5.1 shows the controller's performance when the transmitted GS signal is biased. When the bias is positive the performance of both controllers is similar at first, as can be seen at bias $0.4deg$. However, upon increasing the bias to $0.5deg$ the performance of the PID greatly decreases, this is because the PID controller is not able to pull the aircraft up fast enough, resulting in the aircraft landing much before the runway, hence failing to land the aircraft properly. The RL controller is still able to control the aircraft and land almost in the same place as it had with the bias of 0.4. This is most likely due to the fast and aggressive response the RL controller learns during training. Additionally, the RL controller is able to keep a higher landing vertical speed, while the PID drifts towards the lower acceptable values. When the bias is negative, on the other hand, the PID controller performs better than the RL controller. Likely, due to the less round flare trajectory of the RL controller and its fast response the aircraft ends up landing much further on the runway.

Table 5.1: Landing performance in the Biased GS failure case

	PID	RL	PID	RL	PID	RL	Unit
Bias	0.4		0.5		-0.4		[deg]
Longitudinal Position X_e	-80.24	-75.25	-116.76	-81.32	143.17	237.81	[m]
Lateral Position Y_e	-0.01	-0.24	-0.01	-0.24	0.01	-0.24	[m]
Lateral Velocity	0.0	0.0	0.0	0.0	0.0	0.0	[m/s]
Altitude Rate	-0.36	-0.68	-0.32	-0.7	-0.36	-0.18	[m/s]
Pitch angle	1.78	1.67	1.83	1.66	1.83	1.96	[deg]
Roll angle	0.0	0.0	0.0	0.0	0.0	0.0	[deg]

Table 5.2 shows the controller's performance when the transmitted LOC signal is biased. Both controllers have very similar lateral control performance. With a positive bias of $0.2deg$ both controllers land the aircraft just inside of the boundary, with very similar performance and positioning. With a negative bias of $-0.2deg$ both of the controllers land the aircraft just outside of the runway boundary, with similar touchdown parameters. Interestingly, however, the RL controller lands the aircraft further on

the runway, with a larger error. Indicating that there is a stronger influence of the Roll Attitude controller on the Pitch attitude control, or vice versa, than what is found on the PID.

Table 5.2: Landing performance in the Biased LOC failure case

	PID	RL	PID	RL	Unit
Bias	0.2		-0.2		[deg]
Longitudinal Position X_e	2.35	2.47	-2.71	12.68	[m]
Lateral Position Y_e	7.72	7.91	-8.34	-8.19	[m]
Lateral Velocity	-0.29	-0.29	0.32	0.32	[m/s]
Altitude Rate	-0.44	-0.48	-0.42	-0.45	[m/s]
Pitch angle	1.73	1.78	1.73	1.79	[deg]
Roll angle	-0.01	-0.01	-0.0	-0.0	[deg]

5.3. Alternative Controllers

This section presents results on the use of alternative controllers, that are not necessarily direct substitutes for the inner loop controllers in the ALS controller structure proposed in this research. This includes a Roll Attitude controller that was trained in a cascaded fashion with the Pitch controller and a full attitude controller where the agent controls all three control surfaces.

5.3.1. Roll Attitude - Trained with Active Pitch control

Deflections of the aileron have effects on the elevator and vice versa. In the article, in section 2, the proposed Pitch and Roll attitude controllers were trained separately and then both were put to work together in the ALS. The resulting controllers proved to be robust enough that when combined the effects they cause on each other are corrected. However, their interaction may be improved even further by including such interactions during training. This section proposes a roll attitude controller that is trained while a pre-trained pitch attitude controller is active. The hypothesis is that the roll controller will learn to identify the patterns of the pitch attitude controller and shape its own actions in such a way that the elevator effects on the aileron are minimized while still accomplishing the roll control task at hand. The observation space contain the roll error ϕ_e and the roll rate p, and the action space contains only the aileron deflection δ_a, identical to the previous roll attitude controller.

Such roll attitude controller is trained using the same pitch attitude controller in section 2 and both are implemented in the ALS control structure. Once again the PID Autoflare controller was retuned to suit the new RL inner controllers. The landing results at touchdown are presented in table 5.3 for the Nominal case, in the presence of GS noise and in the presence of LOC noise. First, the performance of the controllers in the nominal case is ideal, presenting very small errors in both the x and y directions as well as the other parameters. The GS noise was once again simulated using Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = 0.1$. In the case of GS noise, the RL controller is able to land the aircraft with a performance similar to that of the RL controller in section 2 in the longitudinal direction, in the lateral direction, albeit having a similar deviation, the average error is much smaller. The LOC noise was simulated using Gaussian noise with mean $\mu = 0$ and standard deviation $\sigma = 0.015$, instead of $\sigma = 0.03$ presented in section 2, this is because the controllers lose control of the aircraft steering it towards ϕ values over $60deg$, which ends up in premature termination of the simulation, and in no landing of the aircraft. The increased performance of the roll attitude controller under GS noise indicates that the pitch controller's effects on the roll controller can be reduced through parallel training. The parallel-trained roll controller, however, has shown to be much more LOC noise sensitive than the independently trained controller.

Table 5.3: Landing performance in the nominal case

Metric at touchdown	Nominal case	GS Noise		LOC Noise		Unit
		μ	σ	μ	σ	
Success rate	-	100.0	-	17.7	-	%
Longitudinal Position X_e	0.79	13.27	53.84	8.72	10.93	[m]
Lateral Position Y_e	0.03	-0.21	1.71	9.58	1.65	[m]
Lateral Velocity	0.01	0.01	0.0	0.2	0.09	[m/s]
Altitude Rate	-0.47	-0.45	0.09	-0.45	0.02	[m/s]
Pitch angle	1.77	1.78	0.05	1.78	0.01	[deg]
Roll angle	0.0	0.0	0.0	0.11	0.02	[deg]

The decreased effects of the elevator on the roll controller is interesting and could generate better controllers, if not for the increased noise sensitivity in the lateral direction. Therefore, further study on this type of training for the roll attitude controller could be interesting. It is important to note that both controllers were trained using the same reference signal and hyperparameters, therefore the sensitivity issue could be improved by hyperparameter tuning and a brief study on different signal references.

5.3.2. Full Attitude Controller

None of the previous controllers made use of one important control surface, the rudder. As mentioned previously, all control surfaces have an influence on each other, therefore, another interesting possibility is a controller that is capable of controlling all three control surfaces of the aircraft, therefore resulting in a full attitude controller. The hypothesis, in this case, is that the agent not only learns how to perform both pitch and attitude control but also learns how one control surface affects the other and therefore learns how to take optimal actions.

This section presents a controller that has been trained to follow a sinusoidal roll reference and a cosinusoidal pitch reference, such that the signals are off-phase with each other. Another reference signal is given in the form of a sideslip angle, which is set to zero, to maintain the aircraft flying straight. The reward function is then set as shown in equation 5.1. Equations 5.2 and 5.3 show the error and scaling vectors, and the agent's action and observation spaces, respectively. The resulting controller is implemented in the ALS substituting both the pitch and the roll attitude controllers, as shown in figure 5.3, and the longitudinal and lateral outer loops are combined. Once again, the auto flare controller was retuned to ensure adequate landing. Table 5.4 shows the touchdown results of the controller in the nominal case, where there is little error in either direction and the other variables are also within the acceptable boundaries.

$$\tilde{r}(t) = -\frac{1}{3}||clip[e_r \odot e_e(t); 0, 1]|| \tag{5.1}$$

$$e_r = \frac{6}{\pi}[1., 1., 4.] \qquad \text{and} \qquad e_e = [\theta - \theta_{ref}, \phi - \phi_{ref}, \beta - \beta_{ref}] \tag{5.2}$$

$$a := [\delta_e, \delta_a, \delta_r]^T \qquad \text{and} \qquad s_{obs} := [\theta_e, \phi_e, \beta, q, p]^T \tag{5.3}$$

Table 5.4: Landing performance in the nominal case

Metric at touchdown	RL	Unit
Longitudinal Position X_e	-0.39	[m]
Lateral Position Y_e	-0.39	[m]
Lateral Velocity	0.33	[m/s]
Altitude Rate	-0.46	[m/s]
Pitch angle	1.67	[deg]
Roll angle	-0.16	[deg]

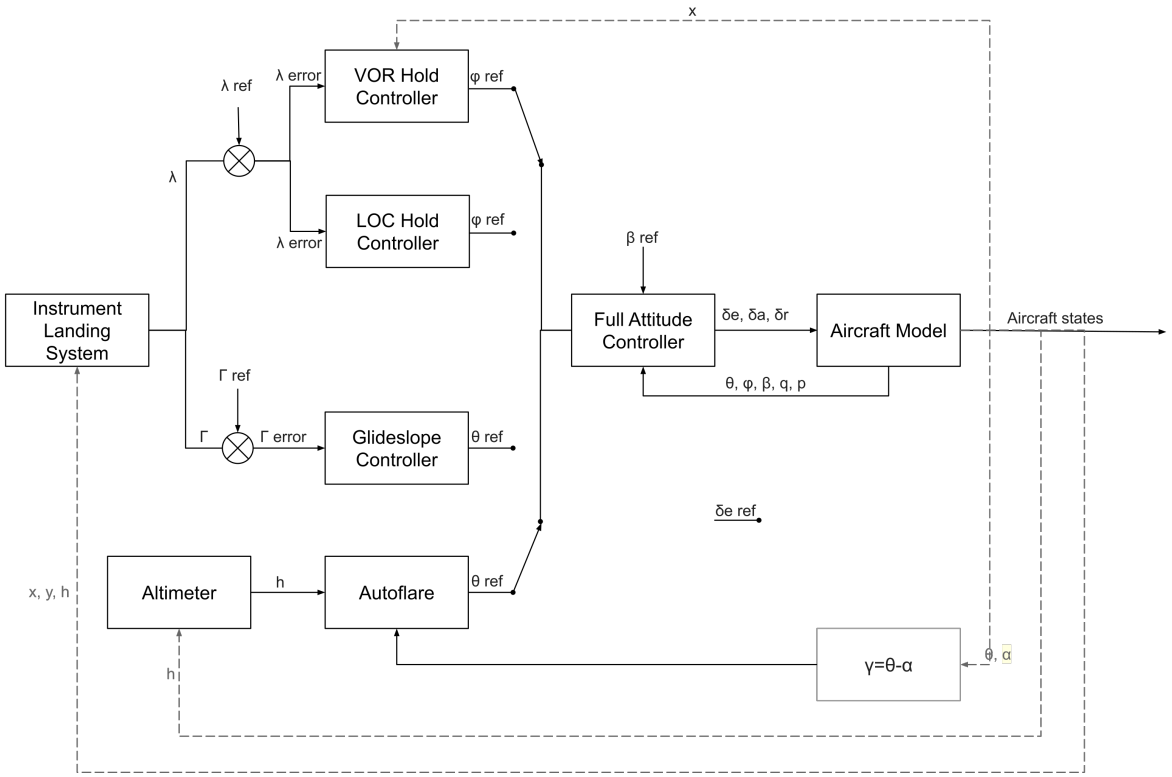
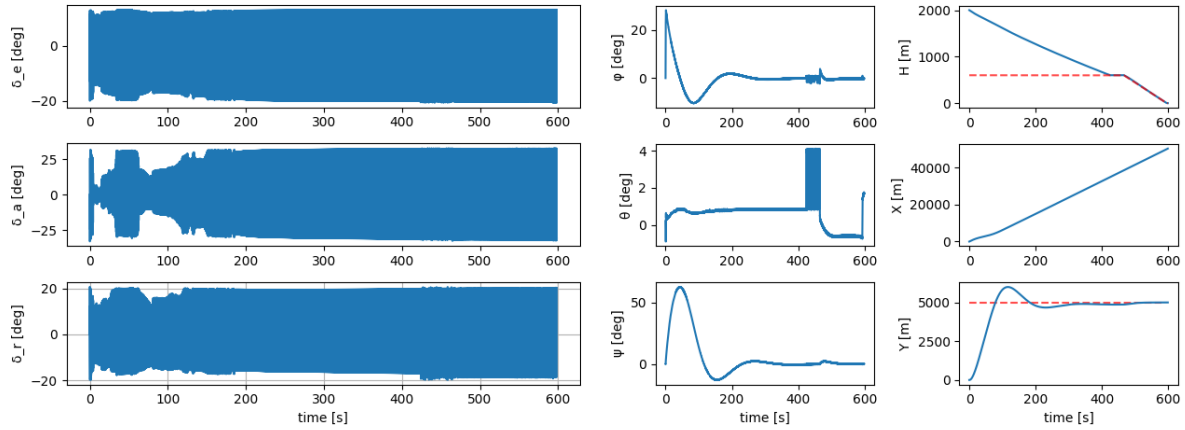


Figure 5.3: Modified ALS with full attitude controller

Although the landing position of the controller is satisfactory, the input command to all three control surfaces is extremely oscillatory, not even allowing to determine what exactly the average signal is, as can be seen in figure 5.4. The elevator command output by the controller is not realistic, since it is not possible for the elevator to change direction in such a manner. To avoid such oscillatory behaviour the aircraft simulation model contains a low-pass filter that eliminates the high frequency of the elevator control inputs, however, it is not interesting to overload this filter and it also may affect the actor's policy. Additionally, this aggressive policy is also not efficient since the actual pitch angle does not follow such oscillations. The most likely cause for this learned policy is overtraining which results in the agent attempting to diminish the tracking error beyond what is necessary. Possible solutions to avoid such behaviour are the use of alternative training reference signals, the addition of parameters for early termination of training, or the addition of reward penalties for oscillatory policy and/or certain aircraft states. The latter has been recommended because it was observed that out of the two actor network input types, reference error and angular rates, the reference error was shown to be smooth, while the angular rate was also oscillatory, indicating that this might be the root cause of this issue. Additionally, there is an overshoot in the lateral direction trajectory, this is due to the VOR hold controller not being retuned.



(a) Input commands **(b)** Aircraft output

Figure 5.4: Full Attitude input commands and aircraft outputs for the landing task

6

Verification & Validation

Verification and Validation are crucial steps in any research, they are necessary to ensure the validity of the results and understanding of its limitations. For this research, it is necessary to perform verification and validation of both the model utilized, to ensure that the aircraft model indeed behaves as it is supposed to, as well as the reinforcement learning algorithm, to ensure that the controllers generated with it are consistent. The following two sections detail the verification and validation of these parts.

6.1. Verification

Verification is performed to ensure that the software is designed as per its requirements, in other words, to ensure that it does what it was intended to do.

6.1.1. Simulation Model

The Citation simulation model was designed by TU delft Control & simulation department of the AE faculty of the TU Delft University using the Delft University Aircraft Simulation Model and Analysis Tool, or DASMAT. The model was designed in MATLAB and has been compiled into C code, such that it can be read and processed by Python, the programming language used in this research. To ensure that the model has been adequately converted from the already validated MATLAB model to a Python model the same simulation is performed in both environments and compared to each other. The results are presented in figure 6.1 and show that there are no visible differences between both responses. Since the Matlab/Simulink model has already been verified and validated in [29], the similarity between its results is sufficient to validate the model conversion.

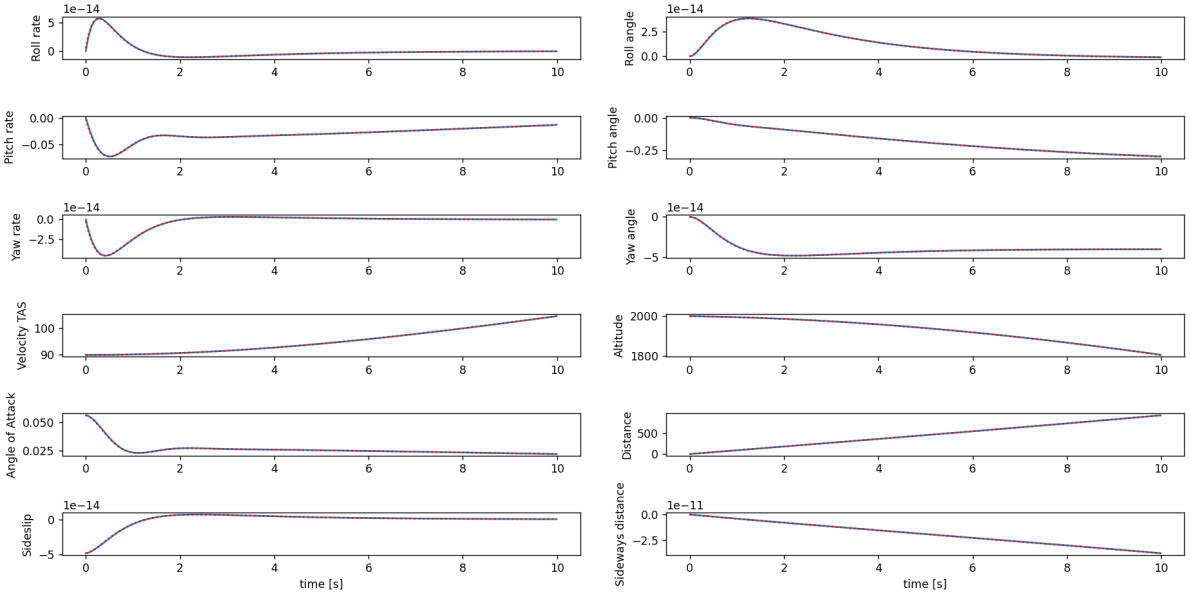


Figure 6.1: Aircraft responses to no inputs between the MATLAB and the converted Python model

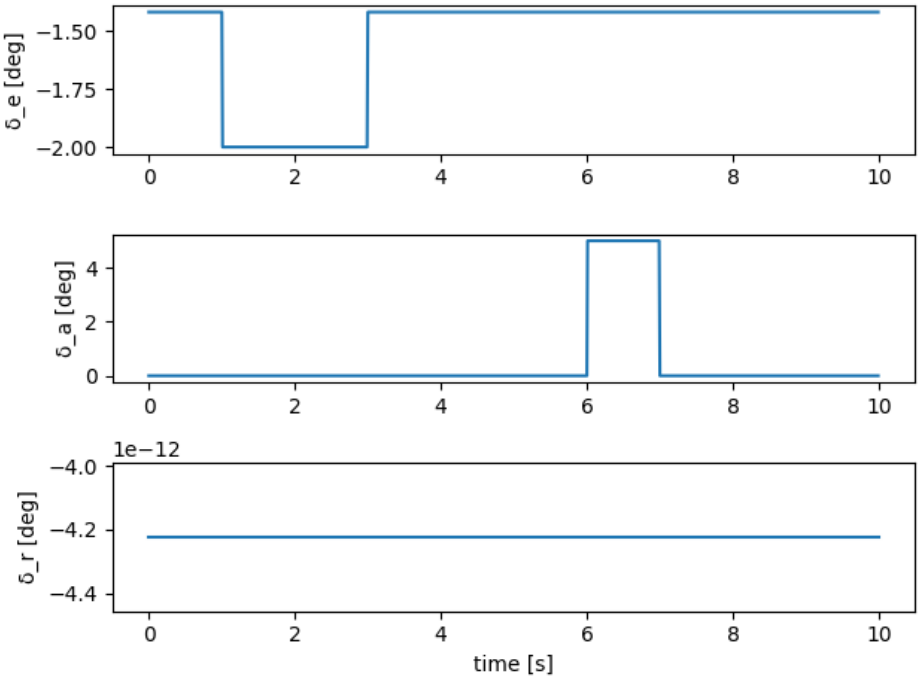


Figure 6.2: Elevator and aileron input commands

Nonetheless, a basic verification of the model can be done through a sanity check in which certain inputs are given to the aircraft, and its reactions are analyzed to determine whether it behaves as expected or not, based on basic aircraft dynamics. To verify that a negative elevator deflection is fed to the model, in this case, it is expected the command induces a positive response to pitch rate, pitch angle and angle of attack. Additionally, a positive aileron deflection is fed to the model, in this case, a negative roll rate, troll angle, sideslip and yaw rate are expected. The control inputs to the model are shown in figure 6.2 and the aircraft responses in figure 6.3, therefore it can be seen that the aircraft responses indeed correspond to the expectations.

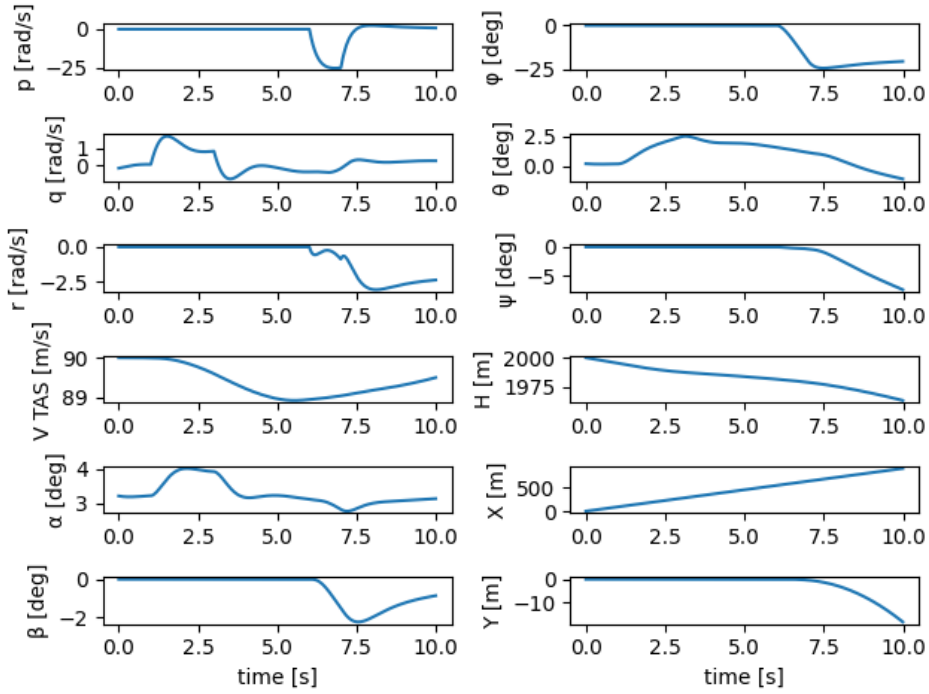


Figure 6.3: Aircraft responses to elevator and aileron commands

6.1.2. Reinforcement Learning Algorithm

The SAC algorithm was implemented following its original paper [25] and previous papers [68] and [17] as reference. The model can be verified by the upward trend of the learning curves presented in section 2, which indicates that the model indeed is learning from its experiences with the environment. The upward trend not only indicates that the algorithm is capable of understanding the results from its previous actions but also that it is capable of interpreting them and making better decisions as it gains more knowledge.

Additionally, the results section 2 show that the reinforcement learning controllers can be used to substitute the attitude PID controllers in the ALS controller structure and the simulations in the verified model prove that the controllers indeed are capable of performing the tasks that they were intended to do.

6.2. Validation

Validation is performed to ensure that the software is designed within acceptable performance boundaries, in other words, to determine the degree to which the generated data is valid/accurate.

6.2.1. Simulation Model

The Cessna Citation 500 simulation model was validated in [29], where its responses to inputs were compared against real flight data collected with the Cessna Citation 550 PH-LAB research aircraft. The differences are measured in terms of the root-mean-square error (RMSE) for a given task, the study reports an RMSE of 8.38% for the longitudinal force coefficient, 12.65% for the longitudinal moment coefficient, and on the lateral direction these errors are reported to be 7.34% and 8.58% for the same coefficients. The relatively low RMSE validates the simulation model as being an accurate representation of the PH-LAB aircraft, and although not perfect, it is sufficient to be used in research and validates initial iterations of controllers.

6.2.2. Reinforcement Learning Algorithm

The algorithm can be validated through an analysis of its performance in different scenarios. The high amount of rewards at the end of the training process that can be seen in the learning curve in section 2 signifies that the algorithm is capable of keeping the errors low as is expected from it. Additionally,

section 2 presents the performance of the SAC algorithm in a cascaded manner with the outer PID controller in the landing task in a series of situations, where both sensor and actuator failures were present. The controllers are validated through the results, which show that the PID-RL based ALS controller is capable of landing the aircraft in nominal condition and in a series of sensor and actuator failures.

6.2.3. Limitations

Even though both the aircraft model and the SAC algorithms have been verified and validated, there are still limitations present. For example, actuator inputs pass through a first-order low-pass filter to account for realistic dynamics, however, the transport delay is not taken into account. The environment is considered to be fully observable and there is no sensor delay. Simulation and controller frequency are set to $100Hz$, however, the effects of it and different frequencies have not been tested. Additionally, there are no aircraft configuration changes throughout the simulations, which is not realistic for a landing task, although the ground contact is not considered in the simulation the landing gear still has aerodynamic effects on the aircraft's performance that should be taken into account.

7

Conclusion & Recommendations

7.1. Conclusions

The majority of aircraft accidents between 2011 and 2020 occurred in the final approach and landing phases of flight, additionally, in 2019 80% of aircraft fatalities occurred due to in-flight loss of control and hard landings. Therefore, the need for better fault-tolerant controllers specific to these phases of flight is evident. The research contained in this report contributes to the field of aviation through an investigation into the use of modern model-free fault-tolerant DRL flight controllers for automatic landing systems. The goal of this study is to make advancements in the field of aviation, by finding modern and robust controllers capable of dealing with the issues in the aforementioned, most troublesome phases of flight. Chapter 1 introduced the topic of the research and set a series of research questions to be answered and to guide the research. Research questions **RQ1** and **RQ2** were answered through a literature study and preliminary analysis, presented in chapters 3 and 4. Research questions **RQ3** and **RQ4** were answered through the works presented in chapters 2 and 5. This chapter concludes this thesis research through a discussion of all the research questions and a review of the research objective.

> RQ1: What are the state-of-the-art methods that are currently employed/proposed in Automatic Landing Systems and what are their problems/challenges?
>
> > RQ1.1 What are the control methods utilised?
> >
> > RQ1.2 Under what circumstances do these systems become faulty/present issues?
> >
> > Q1.3 What are the requirements for such systems? (airfield, weather conditions, onboard equipment...)

Current Automatic Landing Systems (ALSs), as most flight control systems do, rely on classical control theory to design their controllers. Although the use of more modern techniques for flight control can be found, for example, the F35 fighter uses Non-linear Dynamic Inversion (NDI) for increased stabilisation, the vast majority of ALSs, especially in general aviation, still rely on Proportional-Integral-Derivative (PID) controllers. An investigation was done on the topic of landing systems in order to understand how the landing is performed, the instruments and requirements imposed upon it, and the challenges involved in designing (automatic) landing systems. ALS requires two ground-based pieces of equipment, namely, the ILS and a Radio Altimeter, which provide longitudinal and lateral guidance to the aircraft (RQ1.3). The onboard equipment is limited to suitable receivers and computers that are able to process the information provided by the ground-based equipment. It was also determined that uncertainty in the environment in the form of *disturbances* and *actuator failure* are the main barriers to enabling wider use of ALSs in aviation (RQ1.2). These problems are closely linked to the use of classical control theory, since it naturally lacks robustness, does not handle complex and non-linear systems, and suffers from environment uncertainties. Therefore the field requires the development of more *robust* and *adaptive* controllers.

There have been multiple attempts at remedying the problems related to sub-optimal, or poor, performance of classical control theory methods in the face of uncertainties of the landing environment. Ranging from optimal control, with H_{inf} and H_2, to Sliding Mode Control and Dynamic Inversion (RQ1.1), some of the proposed methods have shown promising test results and interesting implementations, however, the majority of them have only considered decoupled aircraft dynamics and/or only consider linearised models. Therefore, the field would certainly benefit from performing more studies and the development of novel adaptive and robust controllers.

RQ2: What characteristics of Reinforcement Learning are suitable to mitigating ALS's issues (from Q1)?

RQ2.1 What are the current state-of-the-art RL algorithms?

RQ2.2 Which RL framework is suitable for resolving ALSs issues?

Q2.3 How well does the proposed algorithm perform when applied to simple flight control systems?

The topic of Reinforcement Learning was introduced in which its basic elements were presented along with a discussion of the general characteristics of the field. It was determined that there are two groups of algorithms within RL that provide the most benefits to ALSs and their environment, namely *Incremental Approximate Dynamic Programming (iADP)* and model-free actor-critic *Deep Reinforcement Learning (DRL)* (RQ2.1). Both are capable of handling control tasks in continuous space and time through the utilisation of Artificial Neural Networks (ANNs) and handle highly dynamic and complex systems through control design that is model-independent, meaning that no prior knowledge of the environment is required. The iADP field utilises techniques to rapidly generate accurate incremental models that are used to reduce model dependence on model-based methods, such as Dynamic Programming (DP), this approach is usually applied in an online fashion which results in highly adaptive controllers. On the other hand, DRL methods rely on their high generalisation power to attenuate uncertainties of the environment, these methods are model-free, hence learning optimal policies whilst allowing the environment dynamics to be completely unknown by the agent. Most commonly, these methods are implemented in an offline manner and through their robustness can achieve certain levels of fault tolerance.

In view of the current needs of the ALS field, the characteristic of RL methods, the results of previous papers on the field of flight control and RL-based controllers, and the nature of this research, it has been concluded that a **robust controller** is the most viable option to be studied. This is because DRL controllers are trained offline and provide high generalisation power, that is capable of operating regardless of relatively small changes in the environment, therefore creating a more solid controller. Current aviation lacks fault-tolerant control that is capable of operating regardless of internal and external disturbances. Therefore, an offline-trained model-free deep reinforcement learning controller was chosen to be investigated in the ALS system (RQ2.2).

Additionally, the performance of the five DRL algorithms in a simple flight control problem was evaluated, along with the classic PID. It was shown that the Soft Actor-Critic algorithm is the best performing in all tests and virtually every parameter analysed (RQ2.3). Being the most robust, consistent and showing higher rewards, it was chosen as the most suitable actor-critic DRL algorithm for the ALS studied in this research.

RQ3: How can the proposed RL framework be implemented in an ALS control design to increase robustness and adaptability?

RQ3.1 What are the characteristics of the landing environment and how can it be modelled?

RQ3.2 What are the characteristics of the RL controller that allow optimal implementation?

A high-fidelity model of the Cessna Citation 500 aircraft was chosen as the training and testing environment in this research. It was chosen to keep in line with previous research on the use of RL for fault-tolerant flight control, and also due to being an accurate model that has been validated through real-flight data comparison in test flights on the PH-LAB aircraft. Additionally, in order to comply with

the instruments and requirements of ALS systems, a model of an ILS was designed to provide the controllers with the required information. Next, a landing task was designed, in which the aircraft starts at position $x = 0m$, $y = 0m$ and $h = 2000m$, reduces its altitude to $h = 60[m$ and performs final descend and lands in an airfield positioned in $x = 50000m$, $y = 5000m$ and $h = 0m$. Hereby answering RQ3.1.

A general cascaded controller structure was designed to perform the proposed landing task, and it contains pitch and roll attitude controllers on the inner loop, GS and LOC controllers for the longitudinal direction, and VOR and LOC hold controllers for the lateral direction, in the outer loop. Each of the controllers in the general structure was designed as a PID controller, resulting in a full PID ALS, to serve as a baseline comparison for the DRL controller and as a representation of current control methods. Since the majority of the dynamic changes are mainly reflected on the control surfaces and being the attitude controllers the closest to them it is expected that these would be the most affected during the failure cases to be analysed. Therefore, it was determined that the SAC controllers would be trained as attitude controllers, more specifically pitch and roll controllers, to provide longitudinal and lateral guidance to the aircraft. The SAC controllers were trained to follow reference signals where the input was the error and the respective angular rate, and the output was an actuator deflection angle. Through this method, it was possible to replace the pitch and roll attitude controllers in the ALS structure without any changes to it. Hereby RQ3.2 is answered.

RQ4: How does the proposed Automatic Landing System method perform compared to other methods?

RQ4.1 How does the simulated landing performance of the proposed method compare to those of classic methods? (with respect to relevant touchdown variables)

RQ4.2 How much training is required for the proposed algorithm to achieve the landing conditions requirements? And how consistent are the results?

RQ4.3 How robust is the proposed system to environment changes?

The PID and the RL-ALS controllers were tested on the designed landing task in nominal condition, that is with ideal sensors and no disturbances, with realistic ILS characteristics, noise and bias added to the ILS-GS and ILS-LOC signals, and under elevator actuator loss of efficiency. The performance was analysed by evaluating the X and Y landing positions, lateral velocity, altitude rate, and pitch and roll angles, all evaluated at touchdown. It was observed that the RL and PID controllers perform similarly under the nominal condition, as well as with realistic ILS signals. The RL-controller presented better behaviour under ILS-GS noisy signals, where it was capable of landing the aircraft inside the acceptable landing area with less variation than the PID, being successful 100% of the simulation runs against 78% for the PID. Significantly lower variation in landing position was observed under LOC signal noise, where both the PID and the RL controllers were capable of maintaining consistent performance. In this case, however, the PID was able to maintain lower mean errors in Y direction whilst maintaining about the same overall variance as the RL controller. Additionally, the PID showed to rapidly fail under elevator loss of efficiency, due to the lack of adaptive resources, such as gain scheduling. The RL on the other hand was capable of controlling the aircraft even with a reduction of 70% in actuator efficiency, this is most likely due to the resulting RL controller generating fast and high deflection, whilst still being stable at the same time. No large differences between the performances of the RL and classical controllers were observed in the additional tests performed, reassuring the capabilities of the RL controller. Hereby RQ4.1 is answered.

The SAC algorithm showed to be able to learn to control pitch and roll attitude relatively fast, if compared to common DRL algorithms in other tasks. The agents were trained in around 250 simulation runs of $20s$ refreshed at $100Hz$, summing up to a total of 5×10^5 timesteps, and stabilising with optimal performance at around 2×10^5 timesteps. The training was consistent overall, however, two main issues were noticed. First, the algorithm showed to be unstable, presenting dips in learning even in lower stages of training; this was remedied by keeping exploration low, at the expense of potentially resulting in less robust controllers. Second, the algorithm showed oscillatory behaviour and signs of overtraining; these issues were remedied by reducing the amount of training to 5×10^5 timesteps, from the usual 1×10^6, and the use of sinusoidal training reference signal. Hereby RQ4.2 is answered.

This research also shows that the SAC algorithm used to create pitch and roll controllers is robust enough that the controllers can be trained separately from each other and still work in conjunction with

each other in a larger controller structure. It also shows that model-free DRL algorithms, as expected, are indeed capable of creating robust controllers that can be insensitive to noise, in certain situations. In addition to that, the robustness of the SAC algorithm can still be retained even if the reference signal is dictated by a PID controller, technically a more error-prone method in the face of noise and non-linearities. Even though the robustness capabilities of the algorithm are shown in the tests performed with GS noise, there are still improvements that can be made in order to make the ALS more robust and fault-tolerant, for instance, training under the failed system improves performance, as seen reported in chapter 3, higher interaction between longitudianl and lateral control is also expected to improve the performance of the ALS. Hereby RQ4.3 is answered.

To conclude this research, the objective of this thesis is reviewed

> "The main research goal is to contribute to the development of Automatic Landing Systems that are capable of performing under unforeseen circumstances with enhanced robustness and control accuracy, to increase repeatability and safety by means of exploring the use of different Reinforcement Learning frameworks applied to control techniques"

This research explores the use of different Reinforcement Learning frameworks through the evaluation of five different RL algorithms in a simple flight control task, resulting in the selection of the most robust of them. The SAC algorithm was then used to create attitude controllers that were implemented alongside PID controllers in an ALS controller structure. The proposed controller was then tested under different circumstances and proved to have similar performance to classical methods, as well as lower variance and higher noise insensitivity in certain cases. The research shows that, albeit needing further research, RL-based controllers have the potential to create more robust and more accurate flight controllers, and by default Automatic Landing Systems as well.

The end product of this research is a study on the use of Reinforcement Learning techniques for fault-tolerant flight controllers, more specifically applied to Automatic Landing Systems. This research aims to aid in the development of better ALS with increased safety and reliability and to ameliorate the performance of aircraft in the final phases of flight to reduce the accidents and fatality numbers that are the motivators for this research. Finally, the results herein presented bring a step closer, albeit small, to the achievement of safe fully-autonomous flight.

7.2. Recommentations

Based on the results of the research herein presented and the insights it brought the following recommendations for future research are made:

- The designed ALS controller structure and the SAC-based controllers were designed in such a manner that the longitudinal and lateral control is performed by different controllers, and require them to be robust enough to be able to perform in conjunction. A better connection between longitudinal and lateral control could be made by having one controller in charge of both directions, therefore further research on full attitude controllers for the ALS inner loop is recommended. Additionally, to better overall performance with such controller, it is also expected that the resulting system is more robust, due to the higher interaction between the control surfaces.
- All tests performed were made using the algorithms trained in nominal condition, none of them was trained under the faulty situation, this was done by design to evaluate the raw robustness of the SAC algorithm. However, training in a faulty situation is known to improve the performance of the controller significantly by increasing its robustness and fault tolerance. Therefore, it is recommended that future research include a study on the benefits that training the attitude controller under faulty conditions brings to the final ALS controller.
- The majority of the hyperparameters utilised in this research were borrowed from previous studies [17] [68] or acquired from trial and error. No structured hyperparameter search was performed. Therefore an in-depth structured study to determine the optimal hyperparameters for SAC-based attitude controllers, and other methods, is recommended. The use of parallel coordinates plot is suggested as a means of analysing a sweep of selected evaluated parameters.
- One of the issues encountered in this research related to highly oscillatory commands coming from the agent, this may be due to over-training or other non-identified reasons. Truncated training and the use of different reference signals were utilised in this research as a workaround to this issue, however, these are not concrete solutions. Reducing the oscillations is still necessary, adding penalties to oscillations or detract rewards for angular rates could be better solutions, therefore, a study on such and alternative solutions is recommended.
- DRL algorithms such as the SAC suffer from low sample efficiency and may not operate as desired in cases where the environment changes are extremely large. As mentioned prior, a combination of robust and adaptive control is a solution to improve the performance of the ALS and allow for it to operate in a wider range of conditions. For example, a hybrid controller utilising the DRL SAC and iADP IDHP is proposed in [68], it was reported that the hybrid controller has increased performance if compared to either of the isolate counterparts. Therefore, it the investigation of the use of iADP, possibly IDHP, in combination with the SAC algorithms for the ALS controller is recommended.
- Even though tests including realistic aircraft sensor characteristics and realistic ILS errors were included, several assumptions were made in this research. Therefore, it is recommended to include models of the effects of realistic sensor and actuator transport delays to increase the confidence of the algorithm for use in real systems.
- The research herein contained presented some sensor and actuator failures, however, there are several more that can, and should, be tested. Therefore, further investigation of the performance of the ALS in additional failure cases is recommended in order to expand the flight envelope that the system is capable of operating.
- This research does not consider velocity changes, and the proposed methods make use of the auto-throttle that is built inside the Cessna Citation model, which kept the velocity constant. However, this is not realistic, since there should be velocity changes during landing, as well as other flight manoeuvres. Therefore, to enable fully autonomous control of the aircraft, eventually, it will be necessary to control the throttle command to track airspeed.

References

[1] *Aircraft Operations. volume I flight procedures*. ICAO, 2006.

[2] Karl Johan Åström, Linda Neumann, and Per-Olof Gutman. *A Comparison Between Robust and Adaptive Control of Uncertain Systems*. English. Technical Reports TFRT-7350. Department of Automatic Control, Lund Institute of Technology (LTH), 1987.

[3] S. N. Balakrishnan and Victor Biega. "Adaptive-critic-based neural networks for aircraft optimal control". In: *Journal of Guidance, Control, and Dynamics* 19 (4 1996), pp. 893–898. ISSN: 15333884. DOI: 10.2514/3.21715.

[4] Gabriel Barth-Maron et al. "Distributed Distributional Deterministic Policy Gradients". In: *CoRR* abs/1804.08617 (2018). arXiv: 1804.08617. URL: http://arxiv.org/abs/1804.08617.

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

[6] Bennylp. *BENNYLP/RL-taxonomy: Loose Taxonomy of Reinforcement Learning Algorithms*. URL: https://github.com/bennylp/RL-Taxonomy#DPG.

[7] Sarthak Bhagat et al. "Deep Reinforcement Learning for Soft Robotic Applications: Brief Overview with Impending Challenges Surgical Tool localization View project Soft Robotics View project Deep Reinforcement Learning for Soft Robotic Applications: Brief Overview with Impending Challenges". In: (2018). DOI: 10.20944/preprints201811.0510.v2. URL: https://www.researchgate.net/publication/329368817.

[8] Boeing. *Statistical summary of commercial jet airplane accidents*. Seattle, Washington: Aviation Safety, 2021.

[9] Eivind Bøhn et al. "Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization". In: *CoRR* abs/1911.05478 (2019). arXiv: 1911.05478. URL: http://arxiv.org/abs/1911.05478.

[10] Clark Borst. "Avionics - Landing Systems - Lecture Slides". Avionics lecture slides. Delft Univerity of Technology. Feb. 2022.

[11] L Buşoniu et al. "Reinforcement learning for control: Performance, stability, and deep approximators". In: *Annual Reviews in Control* 46 (Jan. 2018), pp. 8–28. ISSN: 1367-5788. DOI: 10.1016/J.ARCONTROL.2018.09.005.

[12] Daniel G. Canin, Jeffrey K. McConnell, and Paul W. James. "F-35 high angle of attack flight control development and flight test results". In: American Institute of Aeronautics and Astronautics Inc, AIAA, 2019, pp. 1–29. ISBN: 9781624105890. DOI: 10.2514/6.2019-3227.

[13] Runqi Chai et al. "Review of advanced guidance and control algorithms for space/aerospace vehicles". In: *Progress in Aerospace Sciences* 122 (Apr. 2021). ISSN: 03760421. DOI: 10.1016/j.paerosci.2021.100696.

[14] W. J. Charnley. "Blind Landing". In: *The Journal of Navigation* 12.2 (1959), pp. 115–140. DOI: 10.1017/S037346330001794X.

[15] J. Che and D. Chen. "Automatic landing control using Hinf control and stable inversion". In: *Proceedings of the 40th IEEE Conference on Decision and Control* 1 (2001), pp. 241–246.

[16] *Control theory*. Nov. 2022. URL: https://en.wikipedia.org/wiki/Control_theory#/media/File:Feedback_loop_with_descriptions.svg.

[17] K. Dally and E. van Kampen. "Soft Actor-Critic Deep Reinforcement Learning for Fault-Tolerant Flight Control". In: American Institute of Aeronautics and Astronautics Inc, AIAA, 2022. ISBN: 9781624106316. DOI: 10.2514/6.2022-2078.

[18] Yizhang Dong et al. “Self-learned suppression of roll oscillations based on model-free reinforcement learning”. In: *Aerospace Science and Technology* 116 (Sept. 2021). ISSN: 12709638. DOI: 10.1016/j.ast.2021.106850.

[19] Silvia Ferrari and Robert Stengel. “An adaptive critic global controller”. In: vol. 4. Feb. 2002, 2665–2670 vol.4. ISBN: 0-7803-7298-0. DOI: 10.1109/ACC.2002.1025189.

[20] Lex Fridman, Benedikt Jenik, and Jack Terwilliger. “DeepTraffic: Driving Fast through Dense Traffic with Deep Reinforcement Learning”. In: *CoRR* abs/1801.02805 (2018). arXiv: 1801.02805. URL: http://arxiv.org/abs/1801.02805.

[21] Scott Fujimoto, Herke van Hoof, and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods”. In: *CoRR* abs/1802.09477 (2018). arXiv: 1802.09477. URL: http://arxiv.org/abs/1802.09477.

[22] Adam Gjersvik. *Landing on the Moon with Deep Deterministic Policy Gradients*. 2019. URL: https://openai.com/blog/better-exploration-with-.

[23] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. “Variance reduction techniques for gradient estimates in reinforcement learning”. In: *Journal of Machine Learning Research* 5 (2004). ISSN: 15337928.

[24] Zhiyuan Guan et al. “Prescribed performance control for automatic carrier landing with disturbance”. In: *Nonlinear Dynamics* 94 (2 2018). ISSN: 1573269X. DOI: 10.1007/s11071-018-4427-3.

[25] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[26] Ahmad Hammoudeh and Ahmad Hammoudeh Rlr. “A Concise Introduction to Reinforcement Learning”. In: (2018). DOI: 10.13140/RG.2.2.31027.53285. URL: https://www.researchgate.net/publication/323178749.

[27] Jeffrey J. Harris and James Richard Stanford. “F-35 flight control law design, development and verification”. In: 2018. DOI: 10.2514/6.2018-3516.

[28] S. Heyer, D. Kroezen, and E. van Kampen. “Online adaptive incremental reinforcement learning flight control for a cs-25 class aircraft”. In: vol. 1 PartF. American Institute of Aeronautics and Astronautics Inc, AIAA, 2020. ISBN: 9781624105951. DOI: 10.2514/6.2020-1844.

[29] M. A. van den Hoek, C. C. de Visser, and D. M. Pool. “Identification of a Cessna Citation II Model Based on Flight Test Data”. In: *Advances in Aerospace Guidance, Navigation and Control*. Ed. by Bogusław Dołęga et al. Cham: Springer International Publishing, 2018, pp. 259–277. ISBN: 978-3-319-65283-2.

[30] Jonathan Hui. *RL-actor-critic methods: A3c, Gae, DDPG, Q-Prop*. Aug. 2021. URL: https://jonathan-hui.medium.com/rl-actor-critic-methods-a3c-gae-ddpg-q-prop-e1c41f268541.

[31] IATA. *Safety Report 2019*. 56th. International Air Transport Association IATA, Apr. 2020.

[32] Shaik Ismail et al. “Improved neural-aided sliding mode controller for autolanding under actuator failures and severe winds”. In: *Aerospace Science and Technology* 33 (1 2014), pp. 55–64. ISSN: 12709638. DOI: 10.1016/j.ast.2013.12.016.

[33] Charles C Jorgensen and C Schley. *A Neural Network Baseline Problem for Control of Aircraft Flare and Touchdown*. URL: http://direct.mit.edu/books/book/chapter-pdf/236807/9780262291293_caq.pdf.

[34] Jih Gau Juang and Kai Chung Cheng. “Application of neural networks to disturbances encountered landing control”. In: *IEEE Transactions on Intelligent Transportation Systems* 7 (4 Dec. 2006), pp. 582–588. ISSN: 15249050. DOI: 10.1109/TITS.2006.884885.

[35] E. Van Kampen, Q. P. Chu, and J. A. Mulder. “Continuous adaptive critic flight control aided with approximated plant dynamics”. In: vol. 5. American Institute of Aeronautics and Astronautics Inc., 2006, pp. 2989–3016. ISBN: 1563478196. DOI: 10.2514/6.2006-6429.

[36] Oleg Klimov. *Lunar Lander*. URL: https://www.gymlibrary.dev/environments/box2d/lunar_lander/. (accessed: 07.11.2022).

[37] Jens Kober, J. Andrew Bagnell, and Jan Peters. “Reinforcement learning in robotics: A survey”. In: *International Journal of Robotics Research* 32 (11 Sept. 2013), pp. 1238–1274. ISSN: 02783649. DOI: 10.1177/0278364913495721.

[38] William Koch et al. “Reinforcement Learning for UAV Attitude Control”. In: *CoRR* abs/1804.04154 (2018). arXiv: 1804.04154. URL: http://arxiv.org/abs/1804.04154.

[39] Fang Liao et al. “Fault-tolerant robust automatic landing control design”. In: *Journal of Guidance, Control, and Dynamics* 28 (5 2005), pp. 854–871. ISSN: 15333884. DOI: 10.2514/1.12611.

[40] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning.” In: *ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

[41] Johnson Lopez et al. “A fully autonomous unmanned aerial vehicle landing controller synthesis: Quantitative feedback theory and H ∞ technique comparison”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 226 (Feb. 2011), pp. 281–293. DOI: 10.1177/0954410011408661.

[42] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *CoRR* abs/1706.02275 (2017). arXiv: 1706.02275. URL: http://arxiv.org/abs/1706.02275.

[43] Romulus Lungu and Mihai Lungu. “Automatic landing control using H-inf control and dynamic inversion”. In: *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* 228 (14 Dec. 2014), pp. 2612–2626. ISSN: 20413025. DOI: 10.1177/0954410014523576.

[44] Romulus Lungu and Mihai Lungu. “Automatic landing system using neural networks and radiotechnical subsystems”. In: *Chinese Journal of Aeronautics* 30 (1 Feb. 2017), pp. 399–411. ISSN: 10009361. DOI: 10.1016/j.cja.2016.12.019.

[45] S.M.B. Malaek et al. “Intelligent autolanding controller design using neural networks and fuzzy logic”. In: *2004 5th Asian Control Conference (IEEE Cat. No.04EX904)*. Vol. 1. 2004, 365–373 Vol.1.

[46] “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529 (7587 Jan. 2016), pp. 484–489. ISSN: 14764687. DOI: 10.1038/nature16961.

[47] Volodymyr Mnih et al. “Asynchronous Methods for Deep Reinforcement Learning”. In: *CoRR* abs/1602.01783 (2016). arXiv: 1602.01783. URL: http://arxiv.org/abs/1602.01783.

[48] Volodymyr Mnih et al. “Playing Atari with Deep Reinforcement Learning”. In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[49] Borna Monazzah Moghaddam and Robin Chhabra. *On the guidance, navigation and control of in-orbit space robotic missions: A survey and prospective vision*. July 2021. DOI: 10.1016/j.actaastro.2021.03.029.

[50] Walter R. Fried Myron Kayton. “Avionics Navigation Systems”. In: (1997). DOI: 10.1002/9780470172704.

[51] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. “Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications”. In: (Dec. 2018). DOI: 10.1109/TCYB.2020.2977374. URL: http://arxiv.org/abs/1812.11794%20http://dx.doi.org/10.1109/TCYB.2020.2977374.

[52] E H J Pallett, Amraes S Coyle, and Msetp Blackwell. *Automatic Flight Control Fourth Edition*. ISBN: 9781405135412. URL: www.blackwellpublishing.com.

[53] Danil V. Prokhorov and Donald C. Wunsch. “Adaptive critic designs”. In: *IEEE Transactions on Neural Networks* 8 (5 1997), pp. 997–1007. ISSN: 10459227. DOI: 10.1109/72.623201.

[54] Manning Publications. *Neural network architectures*. July 2019. URL: https://freecontent.manning.com/neural-network-architectures/.

[55] D. M.K.K. Venkateswara Rao and Tiauw Hiong Go. “Automatic landing system design using sliding mode control”. In: *Aerospace Science and Technology* 32 (1 Jan. 2014), pp. 180–187. ISSN: 12709638. DOI: 10.1016/j.ast.2013.10.001.

[56] SUDHARSAN RAVICHANDIRAN. *Hands-on reinforcement learning with python: Master reinforcement learning and deep ... reinforcement learning by building intelligent app*. PACKT Publishing Limited, 2018.

[57] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[58] John Schulman et al. “Trust Region Policy Optimization”. In: (Feb. 2015). URL: http://arxiv.org/abs/1502.05477.

[59] Peter Seres. “Distributional Reinforcement Learning for Flight Control A risk-sensitive approach to aircraft attitude control using Distributional RL”. MA thesis. the Netherlands: Delft University of Technology, 2022. URL: https://repository.tudelft.nl/islandora/object/uuid:6cd3efd1-b755-4b04-8b9b-93f9dabb6108.

[60] Shyh Pyng Shue and Ramesh K. Agarwal. “Design of automatic landing systems using mixed H2/H∞ control”. In: *Journal of Guidance, Control, and Dynamics* 22 (1 1999), pp. 103–114. ISSN: 15333884. DOI: 10.2514/2.4356.

[61] David Silver et al. “Deterministic Policy Gradient Algorithms”. In: *31st International Conference on Machine Learning, ICML 2014* 1 (June 2014).

[62] Bo Sun and Erik Jan Van Kampen. “Incremental Model-Based Global Dual Heuristic Programming for Flight Control”. In: vol. 52. Elsevier B.V., 2019, pp. 7–12. DOI: 10.1016/j.ifacol.2019.12.613.

[63] R.S. Sutton and A.G. Barto. “Reinforcement Learning: An Introduction”. In: *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 1054–1054. DOI: 10.1109/TNN.1998.712192.

[64] Richard Sutton et al. “Policy Gradient Methods for Reinforcement Learning with Function Approximation”. In: *Adv. Neural Inf. Process. Syst* 12 (Feb. 2000).

[65] Richard S. Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: 2000.

[66] K. Tamkaya, L. Ucun, and I. Ustoglu. “H∞-based model following method in autolanding systems”. In: *Aerospace Science and Technology* 94 (Nov. 2019). ISSN: 12709638. DOI: 10.1016/j.ast.2019.105379.

[67] Chi Tang and Ying Chih Lai. “Deep Reinforcement Learning Automatic Landing Control of Fixed-Wing Aircraft Using Deep Deterministic Policy Gradient”. In: Institute of Electrical and Electronics Engineers Inc., Sept. 2020, pp. 1–9. ISBN: 9781728142777. DOI: 10.1109/ICUAS48674.2020.9213987.

[68] Casper Teirlinck. “Reinforcement Learning for Flight Control Hybrid Offline-Online Learning for Robust and Adaptive Fault-Tolerance”. MA thesis. the Netherlands: Delft University of Technology, 2022. URL: https://repository.tudelft.nl/islandora/object/uuid:dae2fdae-50a5-4941-a49f-41c25bea8a85?collection=education.

[69] Edward L. Thorndike. “The Law of Effect”. In: *The American Journal of Psychology* 39.1/4 (1927), pp. 212–222. ISSN: 00029556. URL: http://www.jstor.org/stable/1415413 (visited on 11/25/2022).

[70] Antonios Tsourdos et al. “Developing flight control policy using deep deterministic policy gradient”. In: 2019. DOI: 10.1109/ICARES.2019.8914343.

[71] Hado Van Hasselt. “Reinforcement Learning in Continuous State and Action Spaces”. In: *Adaptation, Learning, and Optimization* (Aug. 2013), pp. 207–251. DOI: 10.1007/978-3-642-27645-3_7.

[72] 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”. In: *IEEE Transactions on Neural Networks* 13 (3 May 2002), pp. 764–773. ISSN: 10459227. DOI: 10.1109/TNN.2002.1000146.

[73] Lilian Weng. *Policy gradient algorithms*. Apr. 2018. URL: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/.

[74] Steven D Whitehead and Long-Ji Lin. *Artificial Intelligence Reinforcement learning of non-Markov decision processes*. 1995, pp. 271–306.

[75] Yuhuai Wu. *OpenAI Baselines: ACKTR amp; A2c*. June 2020. URL: https://openai.com/blog/baselines-acktr-a2c/.

[76] Sibo Yang et al. “Investigation of neural networks for function approximation”. In: vol. 17. Elsevier B.V., 2013, pp. 586–594. DOI: 10.1016/j.procs.2013.05.076.

[77] Ye Zhou, Erik Jan van Kampen, and Qi Ping Chu. “Incremental model based online dual heuristic programming for nonlinear adaptive control”. In: *Control Engineering Practice* 73 (Apr. 2018), pp. 13–25. ISSN: 09670661. DOI: 10.1016/j.conengprac.2017.12.011.

[78] Meixin Zhu, Xuesong Wang, and Yinhai Wang. “Human-like autonomous car-following model with deep reinforcement learning”. In: *Transportation Research Part C: Emerging Technologies* 97 (2018). ISSN: 0968090X. DOI: 10.1016/j.trc.2018.10.024.

A

DRL algorithms

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

Figure A.1: DDPG algorithm, retrieved from [40]

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

Figure A.2: TD algorithm, retrieved from [21]

Algorithm 1 Soft Actor-Critic

Inputs: The learning rates, λ_π, λ_Q, and λ_V for functions π_θ, Q_w, and V_ψ respectively; the weighting factor τ for exponential moving average.

Initialize parameters θ, w, ψ, and $\bar{\psi}$.
for each iteration **do**
  (In practice, a combination of a single environment step and multiple gradient steps is found to work best.)
  for each environment setup **do**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \sim \rho_\pi(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1}\}$
  for each gradient update step **do**
    $\psi \leftarrow \psi - \lambda_V \nabla_\psi J_V(\psi)$.
    $w \leftarrow w - \lambda_Q \nabla_w J_Q(w)$.
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$.
    $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$).

Figure A.3: SAC algorithm, retrieved from [25]

Algorithm 3 Trust Region Policy Optimization

Input: initial policy parameters θ_0
for $k = 0, 1, 2, \ldots$ **do**
Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Form sample estimates for

- policy gradient $\hat{g}_k$ (using advantage estimates)
- and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$

Use CG with n_{cg} iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$
Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$
Perform backtracking line search with exponential decay to obtain final update

$$\theta_{k+1} = \theta_k + \alpha^j \Delta_k$$

end for

Figure A.4: TRPO algorithm, retrieved from [58]

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
for actor=$1, 2, \ldots, N$ **do**
Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
end for
Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
$\theta_{\text{old}} \leftarrow \theta$
end for

Figure A.5: PPO algorithm, retrieved from [57]

B

Testing results

B.1. Trained without wind

Table B.1: Actor-Critic DRL algorithms average performance in 1000 episodes of the standard Lunar Lander problem and with wind introduced with $w_{power} = 15$

Algorithm	Wind Power	Time per ep [min]	Steps per ep	Time per step [ms]	Mean reward	Standard deviation
PID	0	0.204	232.2	0.878	252.6	62.9
	15	0.202	216.2	0.934	57.8	175.3
DDPG	0	0.201	215.6	0.932	278.4	31.4
	15	0.201	243.2	0.826	266.1	51.9
TD3	0	0.226	188.1	1.201	283.4	20.5
	15	0.269	244.9	1.098	269.2	34.1
SAC	0	0.184	198.1	0.928	280.1	16.9
	15	0.337	255.6	1.318	252.8	71.8
A2C	0	0.783	462.5	1.693	221.3	39.5
	15	0.415	326.2	1.3	125.8	141.2
PPO	0	0.340	275.4	1.234	220.2	82.1
	15	0.421	286.1	1.471	151.2	135.3

B.2. Trained with wind

Table B.2: Wind trained Actor-Critic DRL algorithms performance in 1000 episodes of the Lunar Lander problem with wind power $w_{power} = 10$

Algorithm	Wind Power	Time per ep [min]	Steps per ep	Time per step [ms]	Mean reward	Standard deviation
PID	10	0.148	221.8	0.667	159.7	127.8
	15	0.124	214.2	0.578	127.0	129.8
	20*	0.206	291.3	0.707	77.6	78.0
DDPG	10	0.212	205.1	1.033	271.9	41.3
	15	0.236	214.4	1.100	271.3	39.1
	20	0.232	234.6	0.988	268.0	45.3
TD3	10	0.215	208.9	1.028	276.9	20.3
	15	0.190	213.6	0.889	274.5	24.8
	20	0.241	232.9	1.034	270.9	28.5
SAC	10	0.173	174.3	0.992	283.6	19.2
	15	0.191	186.6	1.023	279.8	20.1
	20	0.163	195.7	0.832	277.7	26.0
A2C	10	-	-	-	-	-
	15	-	-	-	-	-
	20	-	-	-	-	-
PPO	10	0.4284	287.8	1.488	195.1	108.4
	15	0.401	286.2	1.401	156.1	131.6
	20	0.545	333.5	1.634	150.224	135.1

The * accounts for the fact that the hill climbing PID tuning procedure was not capable of finding parameters that reached 200 in any given episode, being the best result a score of 158 reward points.