Shielded Reinforcement Learning for Flight Control

Giulia Gatti

Delft University of Technology

TUDelft

Shielded Reinforcement Learning for Flight Control

by

Giulia Gatti

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on March 17, 2023 at 9:30.

Student number:	4437438	
Project duration:	September, 2021 - March, 2023	
Thesis committee:	Dr. ir. M.M. van Paassen	TU Delft, Chair
	Dr. ir. E. van Kampen	TU Delft, Supervisor
	Ir. A. Bombelli	TU Delft, Examiner
	Ir. I. El-Hajj	TU Delft

Cover Image from C. v. Grinsven

TU Delft

Preface

Writing the preface is a moment I have not heard many people talk about, surely because it is overshadowed by the continuous struggles that the thesis is. However, for the first time, it feels real that this document is completed and with it, my time as a Delft University of Technology student. Working on this thesis about reinforcement learning has been possible with the help of many people to whom I would like to express my gratitude.

First and foremost, I would like to thank Dr.ir. Erik-Jan van Kampen for his impeccable supervision. Your continuous guidance and encouragement has been imperative to the success of this research. You showed me that no progress is, in the end, also progress and were supportive throughout these many months. I would like to thank my friends, most of whom went through a similar experience, for making the long study days and nights more bearable. Sam, thank you for always being present and knowing the right things to say when no one else did. Your support has pushed me to the finish line. Thank you to my family, without whom I would have not be able to come to Delft as a clueless 18 years old. Your utmost and unconditional support made all of this possible. This degree is as yours as much as it is mine.

Giulia Gatti
Delft, March 2023

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

Abbreviations

AC	Actor-Critic.
ADP	Approximate Dynamic Programming.
AFCS	Automatic Flight Control System.
ANN	Artificial Neural Networks.
CurL	Curriculum Learning.
DASMAT	Delft Aircraft Simulation Model and Analysis Tool.
DDPG	Deep Deterministic Policy Gradient.
DHP	Dual Heuristic Programming.
DP	Dynamic Programming.
DQN	Deep Q-Networks.
EASA	European Union Aviation Safety Agency.
EFCS	Electronic Flight Control System.
FuNs	Feudal Networks for HRL.
GDHP	Global Dual Heuristic Programming.
HAMs	Hierarchy of Abstract Machines.
HDP	Heuristic Dynamic Programming.
HER	Hindsight Experience Replay.
HRL	Hierarchical Reinforcement Learning.
IATA	Interational Air Transport Association.
ICAO	International Civil Aviation Organization.
IFC	Initial Flight Condition.
ILS	Instrument Landing System.
INDI	Incremental Nonlinear Dynamic Inversion.
IRL	Inverse Reinforcement Learning.
ISD	Independent Sub-Task Discovery.
KL	Kullback–Leibler.
LfD	Learning from Demonstration.
LHP	Learning Hierarchical Policy.
LOC-I	Loss of Control (in-Flight).

MAHRL Multi-Agent Hierarchical Reinforcement Learning.

MDPs Markov Decision Processes.

MIMO Multiple-Input-Multiple-Output.

MRPs Markov Reward Processes.

MSBE Mean-Squared Bellman Error.

nMAE Normalized Mean Absolute Error.

PG Policy Gradient.

PIDNN Proportional Integral Derivative Controller with Neural Network.

PPO Proximal Policy Optimization.

RBF Radial Basis Function.

RDRL Deep Reinforcement Learning.

ReLu Rectified Linear Unit.

RL Reinforcement Learning.

RQ Research Question.

SAC Soft Actor-Critic.

SAC-X Scheduled Auxiliary Control.

SCS Safety Checking Shield.

SHERPA Safety Handling Exploration with Risk Perception Algorithm.

SIP Safe Initial Policy.

SIPS Safe Initial Policy Shield.

SMDPs Semi-Markov Decision Processes.

SML Safe Modification Layer.

SRL Safe Reinforcement Learning.

TD Temporal Difference.

TD3 Twin Delayed DDPG.

TRPO Trust Region Policy Optimization.

UAS Unmanned Aerial System.

UAV Unmanned Aerial Vehicle.

UNI Unification with Sub-Task Discovery.

Greek Symbols

α Angle of attack.

α Backtracking coefficient.

α Learning rate.

α_{stall} Stall angle of attack.

β	Pessimistic Q-learning parameter.
β	Sideslip angle.
β_{ω_2}	Terminal state for level two sub-task.
δ	Kullback-Leibler constant.
δ_ϕ	Rudder deflection angle.
δ_a	Aileron deflection.
δ_e	Elevator deflection.
δ_r	Rudder deflection.
ϵ	Exploration probability.
η	Temperature parameter.
η_θ, η_ψ	Step size.
Γ	Safe policy.
γ	Discount factor.
γ	Flight path angle.
γ_{ref}	Reference flight path angle.
λ	Constraint regularization parameter.
θ	Policy patameter vector.
$\mathcal{E}$	Input noise.
$\mathcal{E}$	Set of edges in acyclic graph describing a curriculum.
$\mathcal{V}$	Set of vertices in acyclic graph describing a curriculum.
ω^1	Level one sub-task.
ω^2	Level two sub-task.
Ω_Γ	Set of sub-tasks of the main task Γ.
Ω_Γ^1	Set of sub-tasks of the level two sub-task ω^1.
Ω_Γ^2	Set of sub-tasks of the level two sub-task ω^2.
$\Omega_{\text{hierarchy}}$	Set of sub-tasks for the hierarchy.
ω_t	Sub-task.
ϕ	Roll angle.
ϕ^s	Safety specification.
ϕ_0	Initial value-function parameter.
Π	Policy space.
π	Policy.
π^*	Optimal policy.
π_Γ	Highest level policy in the hierarchy.

π_{ω_t} Sub-task policy.

$\pi_{\text{hierarchy}}$ State-sub-task-action mapping for the hierarchy.

ψ Value-function parameter.

ψ Yaw angle.

ρ Deviation from safe action.

σ Ornstein-Uhlenbeck variance.

τ DDPG smoothing factor.

τ Trajectory produced by the updated policy.

θ Ornstein-Uhlenback mean attraction constant.

θ Pitch angle.

θ Randomly chosen weight for DDPG policy network.

θ' Updated policy.

θ, π_θ Parametrized policy approximation.

Roman Symbols

$\hat{a}_t$ Adjusted action.

$\hat{q}_{\text{ref}_\alpha}$ Adjusted reference pitch rate according to M_{SR_α}.

$\hat{q}_{\text{ref}_n}$ Adjusted reference pitch rate according to M_{SR_n}.

$\hat{q}_{ref}$ Adjusted reference pitch rate.

$\hat{A}_t$ Temporal difference error correction.

$\hat{H}_k$ Hessian of sample average KL-divergence.

$\hat{Q}$ Target action-value function.

$\mathcal{D}$ Memory buffer.

$\mathcal{D}^{\mathcal{T}}$ Set of transitions samples from tasks $\mathcal{T}$.

$\mathcal{H}(\mathcal{P})$ Entropy.

$\mathcal{L}_\theta(\theta')$ Approximation of the expectation.

$\mathcal{P}$ Probability operator.

$\mathcal{S}_{\text{safe}}$ Safe state space.

$\mathcal{S}_{\text{unsafe}}$ Unsafe state space.

$\mathcal{S}_{\text{risk}_H}$ High risk state space.

$\mathcal{S}_{\text{risk}_L}$ Low risk state space.

$\mathcal{T}$ Set of tasks.

$\tilde{r}_t$ Instantaneous reward at time t.

A Primitive action space.

a' Action network.

a'_t Proposed safe action.

A^θ Advantage function.

a_0 Initial action at time $t = 0$.

a_{safe} Safe action.

a_t Action at time t.

C Curriculum.

C Delayed steps.

c_i Constraint function for safe contrained criterion algorithm.

c_{ω_t} Time steps for carrying out sub-task ω_t.

D_k Replay buffer.

E Expectation operator.

e_ϕ Pitch angle error.

e_{ω_y} Pitch angular rate error.

g Function relating vertices and values in $\mathcal{D}^{\mathcal{T}}$.

h Altitude.

h_i Objective function for safe contrained criterion algorithm.

I_ϕ Pitch angle integral.

$J(\theta)$ Actor objective function.

$J_{V_\psi^{\pi_\theta}}(\psi)$ Critic objective function.

$J_{V_\theta^{\pi_\theta}}(\psi)$ Actor objective function.

$J_{V_\theta^{\pi_\theta}}(\psi)$

K Maximum number of backtracking steps.

k Randomly chosen weight for DDPH critic network.

k Scaling factor.

$k_{1,2}$ Randomly chosen weights for policy and twin-critic SAC networks.

L Loss function.

M_{SR} Safety Range Model.

M_{SR_α} Safety range model for angle of attack safety.

M_{SR_n} Safety range model for load factor safety.

N Number of steps in one episode.

N Replay buffer capacity.

n Load factor.

n_{crit} Critical load factor.

p Roll rate.

q Pitch rate.

q'_{ref} Proposed reference pitch rate.

$Q(s_t, \omega_t)$ Action-value function at time t for following sub-task policy ω_t.

$Q^*(s_t, a_t)$ Optimal action-value function.

$Q^\pi(s_t, a_t)$ Action-value function at time t for following policy π.

$Q^\pi{}_\theta(s_t, a_t)$ Action-value function for parametrized policy π_θ.

$q_{ref_{safe}}$ Safe reference pitch rate.

q_{ref} Reference pitch rate.

r Yaw rate.

R_α Reward function related to angle of attack's safety.

R_n Reward function related to load factor's safety.

R_s Markov Reward Process.

R_t Discounted reward at time t.

R_t Reward function related to tracking.

r_t Reward at time t.

s_0 Initial state at time $t = 0$.

s_t State at time t.

T Horizon of sampled trajectories.

T Simulation time.

t Time step.

u_0 Initial trimmed control inputs.

$V^*(s_t)$ Optimal state-value function.

$V^\pi(s_t)$ State-value function at time t for following policy π.

V_{TAS} True airspeed.

w Risk metric parameter.

x Random value for reference flight path angle initialization.

x_0 Initial trimmed states.

y_j Targets.

1

Introduction

1.1. Background

Over the last 15 years, the aviation industry has experienced a 64% increase in air traffic (Mazareanu, 2022). Along with this boost, the fatalities per number of flights performed globally has drastically reduced by almost 68%. The majority of these fatalities are caused by in-flight loss of control (LOC-I) which results from the deviation of an aircraft from its intended path or from its operational flight envelope (IATA, 2015). However, the aforementioned downward trend in fatalities could be back-tracked to improvements in automation. Particularly due to advancements in flight control systems that allow in-flight operations to proceed more safely without incurring deviations from the flight envelope.

Currently, Automatic Flight Control Systems (AFCS) are generally made up of linear uncoupled gain-scheduled controllers that use look-up tables to handle coupled-dynamics. Tuning is achieved with the help of a linearized system dynamics model for given operating points inside the flight envelope (Cook, 2012). Considering that AFCS are based on known system dynamics, adjusting to unfamiliar conditions may be a challenging task and would result in an increase in the pilot workload. Nonlinear dynamic controllers, such as ones implementing adaptive nonlinear control and adaptive backstepping, have been a rewarding solution. These methods are able to adjust to unexpected conditions without needing gain-scheduling. However, one drawback of nonlinear dynamic methods is the dependency on the model (Sonneveldt et al., 2007; Steinberg, 2001). If the on-board model would incorrectly describe the aircraft due to e.g. poor modeling or changes in the aircraft during flight (in case of damage), the controllers would not be able to guarantee optimal performance.

Recent advancements in the development of flight controllers have been focused on creating learning controllers based on machine learning techniques. Steinberg (2001) identified Reinforcement Learning (RL) as one of the machine learning techniques that could be implemented to improve flight control performance. Reinforcement learning is a framework inspired by how humans approach learning where an agent tries to accomplish a task by trial and error. Learning by interaction allows the controller to not be dependent on the dynamics of the environment and be better suited at adjusting to unknown situations (Dally, 2021). Reinforcement learning not only provides a framework for dealing with uncertainties but it does so in an efficient manner. Ecoffet et al. (2021) found that RL scores better than humans in many video games and hint that this framework could be used to potentially substitute the need of a supervisor for many higher complexity tasks. One example of high complexity task is represented by flight control, which until recently has been typically performed by non-learning controllers. Clarke and Hwang (2020) show that a Deep Reinforcement Learning agent is able to successfully control a fixed-wing aircraft through a series of maneuvers that the agent was not trained for. Similarly, Milz and Looye (2020) and Wang et al. (2020) both demonstrated the ability of a Deep Deterministic Policy Gradient agent to control the attitude of a fixed wing aircraft for conditions different from the trained ones. The robustness of RL agents is shown by Dally (2021), who successfully demonstrated the ability of a deep RL agent to control an aircraft not only when different types of failures occur but also in different flight conditions, such as ones including wind or turbulence. The

examples discussed above are only a sample in the myriad of research devoted to RL in flight control. The general interest in the topic allows for many different aspects of RL in flight control to be investigated.

To make reinforcement learning more appealing to flight control applications, numerous safety paradigms can be applied to a given controller in order to guarantee in-flight safety. Garcıa and Fernández (2015) give a comprehensive overview and classification of the two main approaches in safe RL (SRL) which allow safe learning: modifying the optimization criterion or modifying the exploration process. In RL, the agent needs to optimize a policy in order to maximize the reward. However, the actions taken in order to increase the accumulated reward may not always be safe as no metric of risk is explicitly included in the basic RL approaches (Heger, 1994). Therefore, additional paradigms that explicitly enhance the agent's safety are required. Some SRL algorithms are characterized by the modification of the optimization criterion. Constrained criterion presented by Di Castro et al. (2012) is the most notable of this class. In this case, the aim is to maximize the expectation of the return for a policy subjected by certain constraints. This is particularly advantageous for risky domains as the best policy has to be found within the domain of safe policies Γ. Regarding SRL by modifying the exploration process, most of the approaches increase safety by providing the agent with external knowledge. A baseline approach is to record some demonstrations from a human teacher and provide it to the agent to be used for safe exploration (Driessens and Džeroski, 2004). Xiong (2021) introduces the concept of a Safety Modification Layer (SML), which compares the states reached by a learning agent with known safe states. This knowledge is given to the SML upon initialization of the algorithm as a safe policy Γ. The agent learns from this policy in order to avoid unsafe situations. The contribution by Xiong (2021) is particularly relevant as the environment is a fixed-wing aircraft alike in this research. Another approach is proposed by Matiisen et al. (2019) where a student-teacher dual system is developed. A teacher gives the student advice to increase the reward while performing tasks that have already a high learning curve. This is aimed at performing one task at the best of the agent's abilities before moving on to a less known, potentially unsafe task. Geramifard et al. (2013) implement the teacher-advice method on UAV's fuel planning. In this paper, a safe function is defined, which is based on a constrained function that gives information on which states are allowed or not due to the risk of being unsafe. Once the safe function is defined, a teacher consults it and gives the agent advice on a strategy aimed at avoiding constrained states. This method combines constrained criterion with the teacher-student dual system, an approach similar to shielding developed by Alshiekh et al. (2018).

The flexibility provided by reinforcement learning allows for many different flight control tasks to be carried out. This adaptability, together with robustness many of the researches in the field prove as summarized by Chen and Li (2020), make reinforcement learning an excellent candidate for further decreasing airborne accidents. As mentioned before, incidents due to LOC-I are rather prominent as they can occur during any airborne phases of the envelope. LOC-I is mostly due to failing to prevent or recover from stall and is prevalent during initial climb and landing (IATA, 2015). According to EASA (2021), the stall of an aircraft is highly linked to its angle of attack. Once the stall angle of attack is reached, the stall recovery procedure dictated by EASA needs to be followed. At this end of this procedure, the pilot is required to return to the indicated flight path angle. Additionally, although not strictly linked to stall, the load factor is of great importance when speaking of safety in flight control. An excessive value can lead not only to passenger's harm, but also to structural problems to the aircraft. Therefore, investigating the effects of the angle of attack and the load factor on flight safety is of crucial importance in the efforts to reduce airborne fatalities.

This research will focus on the development of a state-of-the-art method to further advance the efforts in improving flight safety. This will be achieved by proposing a controlled based on a reinforcement learning algorithm equipped with a safe RL paradigm that can maintain safety throughout the flight envelope. In the context of this work, safety is defined as maintaining the aircraft in flight conditions that are within certain limits designed to keep the aircraft in a safe state. This will be done by monitoring states such as the angle of attack and and the load factor while a flight path controller maintains the aircraft on its designated flight path. The developed approach will be tested on a high-fidelity simulation model of the Cessna Citation PH-LAB, a research business jet operated by Delft University of Technology in the Netherlands.

1.2. Research Aim

In this section, the aim of this research will be delineate by identifying a clear objective.

> *The objective of this thesis is to improve the safety of a State-of-the-Art learning flight control system for a fixed wing aircraft by implementing the most promising reinforcement learning algorithm.*

To fulfill the research objective, several research questions will be answered throughout this thesis.

RQ-1 **What are the main challenges in flight control?**

- *RQ-1.1* How is safety in flight control defined and assessed?
- *RQ-1.2* What is the state-of-the-art of flight control?

RQ-2 **Why is reinforcement learning being introduced in flight control?**

- *RQ-2.1* How is safety in reinforcement learning defined and assessed?
- *RQ-2.2* What is the state-of-the-art in reinforcement learning?
- *RQ-2.3* What is the state-of-the-art of reinforcement learning in flight control?
- *RQ-2.4* Which flight control task is more relevant to approach with reinforcement learning and why?

RQ-3 **How can safety in flight control be improved using reinforcement learning techniques?**

- *RQ-3.1* What RL methods can be implemented and what are their characteristics in order to improve safety of a flight control system of a fixed wing aircraft?
- *RQ-3.2* What are the necessary requirements to be set on the RL method and how do they relate to the requirements of the flight control system?

RQ-4 **What reinforcement learning algorithm can be combined with a safety enhancing technique to improve safety?**

1.3. Report Structure

This report will unfold as follows. Part I contains the scientific article where a detailed overview of this research is given. The fundamental topics discussed throughout the research are given in Section II. In Section III, the controller design is presented. The results of the simulations are given in Section IV. The concluding notes and recommendations are discussed in Section V.

Part II will give an overview of the preliminary research conducted prior to defining the objectives of this work. Chapter 2 will give the reader knowledge about the fundamentals of reinforcement learning together with a taxonomy of most common methods. Algorithms that focus on improving safety are discussed in Chapter 3. Once the reinforcement learning algorithms are presented, their State-of-the-Art application in flight control is discussed in Chapter 4. In Chapter 5, a selection of the discussed algorithms will be applied to a simple system to study their workings.

Part III will present additional results. A study on turbulence is conducted in Chapter 6. The robustness of the developed controllers to various reference signals and initial flight conditions is assessed in Chapter 7 and Chapter 8 respectively. All the models used in this research are verified and validated in Chapter 9.

The thesis concludes in Part IV. Chapter 10 presents the main conclusions while Chapter 11 gives the author's recommendations for further research.

Part I

Scientific Article

Shielded Reinforcement Learning for Flight Control

Giulia Gatti *
Delft University of Technology, 2629HS, Delft, The Netherlands

In-flight loss of control has been consistently identified as the main cause of airborne fatalities over the last 15 years. Recent research has been focusing on improving current automatic flight controllers by introducing reinforcement learning and developing techniques to enhance in-flight safety. In this research, an offline Deep Deterministic Policy Gradient (DDPG) controller is equipped with a shield, an additional controller able to monitor the flight path angle and suggest safe actions if a risky state space is reached. The safe actions are proposed by the Safe Initial Policy (SIP) model, a pre-trained agent with knowledge about safe states and imposed by the Safety Range M_{SR} model, a simple rule based system. The shielded DDPG controller is successful in a conventional step down approach from top of descent with a normalized Mean Absolute Error of 24.0%. The controller is robust to many different initial flight conditions, reference signals, biased sensor noise and severe turbulent flow applied with a realistic patchy turbulence model.

I. Introduction

OVER the last 15 years, the aviation industry has experienced a 64% increase in air traffic [1]. Along with this boost, the fatalities per number of flights performed globally has drastically decreased by almost 68%. The majority of these fatalities can be linked to in-flight loss of control (LOC-I) which results from the deviation of an aircraft from its intended path or from its operational flight envelope [2]. The aforementioned downward trend in fatalities could be backtracked to improvements in automation, particularly due to advancements in flight control systems that allow in-flight operations to proceed more safely without incurring deviations from the flight envelope.

Recent advancements in the development of flight controllers have been focused on creating learning controllers based on machine learning. [3] identified Reinforcement Learning (RL) as one of the machine learning techniques that could be implemented to improve flight control performance. Reinforcement learning is a framework inspired by how humans approach learning, where an agent tries to accomplish a task by trial and error. Learning by interaction allows the controller to not be dependent on the dynamics of the environment and be better suited at adjusting to unknown situations [4]. Reinforcement learning not only provides a framework for dealing with uncertainties but it does so in an efficient manner. [5] found that RL scores better than humans in many video games and hint that this framework could be used to potentially substitute the need of a supervisor for many higher complexity tasks. In flight control research, RL has been widely investigated. [6] proposes a research focused on applying RL methods to robust and adaptive flight control tasks. In the paper, a non-learning controller is used as a benchmark to compare the efficacy of a DDPG learning controller when working on the automatic landing of a large cargo aircraft. Similar work has been done by [7], who developed an automatic landing DDPG controller for a fixed-wing aircraft. The flexibility of RL in flight control is showcased by its use with many different aircraft and learning tasks. Notable examples are [8] where a TD3 controller for a Flying-V aircraft is developed and [9], where a DDPG controller is trained for the control of an Unmanned Aerial System (UAS).

*M.Sc student, Control and Simulation Division, Faculty of Aerospace Engineering, Kluyverweg 1, 2629HS Delft, the Netherlands

The flexibility provided by reinforcement learning allows for many different flight control tasks throughout the flight envelope to be carried out. This adaptability, together with the robustness many of the researches in the field prove, make reinforcement learning an excellent candidate for further decreasing airborne accidents. As mentioned before, incidents due to LOC-I are rather prominent as they can occur during any airborne phase of the envelope. LOC-I happens mostly due to failing to prevent or recover from stall and is prevalent during initial climb, descent and landing [2]. Descent can be demanding in terms of performance, as it occurs at steep pitching angles and fast vertical velocities. Therefore, it is interesting to design a learning controller for this section of the flight envelope. According to [10], EASA's stall recovery procedure indicates the return to the intended flight path once the aircraft has returned to a safe condition. Therefore, this research will focus on developing a learning flight path controller able to avoid unsafe flight conditions from the top of descent.

The contribution of this paper is the development of a state-of-the-art learning flight path controller assisted by a *shield* to further advance the efforts in improving flight safety. The shield will act as a separate controller that can propose a safe action when the environment nears risky states. This way, the learning DDPG flight controller will be able to maintain the aircraft within the safe flight envelope without being directly equipped with knowledge about safe boundaries. The developed approach will be tested on a high-fidelity simulation model of the Cessna Citation PH-LAB, a research business jet operated by Delft University of Technology. This paper is structured as follows. The fundamentals of reinforcement learning and its safe counterpart are given in Section II. The design of the learning controller and of the shield are detailed in Section III. Section IV presents the results of the research and discuss the relevant findings. Finally, Section V contains the conclusions and the recommendations given by the author.

II. Fundamentals

This section will discuss the keystone topics of this research. Section II.A will present the main notions of reinforcement learning. Section II.B will present the specific RL algorithm used to develop the flight controller while Section II.C will present the shielding technique.

A. Reinforcement Learning

Reinforcement learning (RL) is a machine learning method that takes inspiration from the way humans learn. An agent interacts with the environment and receives a reward for an action taken, leading to a learning approach based on trial and error. The cornerstone of reinforcement learning are Markov Decision Processes (MDPs) which describe the interactions between the agent and the environment. At a certain time step t, the agent goes from state s_t to state s_{t+1} after an action a_t is taken and a reward r_{t+1} is awarded. This transition is, according to the Markov Property, only dependent on the agent's current state and can be expressed by Eq. (1)

$$\mathcal{P}\left\{s_{t+1}, \tilde{r}_{t+1} \mid s_t, a_t\right\} \tag{1}$$

where $\tilde{r}_{t+1}$ is the immediate reward acquired by the agent after a_t is performed [11]. The goal of RL is to maximise not the immediate reward, but rather the discounted reward accumulate by the agent over the whole training until the terminal time $t + N + 1$. This is defined as R_t and given by Eq. (2)

$$R_t = \tilde{r}_{t+1} + \gamma\tilde{r}_{t+2} + \ldots + \gamma^N \tilde{r}_{t+N+1} = \sum_{N=0}^{\infty} \gamma^N \tilde{r}_{t+N+1} \tag{2}$$

where γ is the discount factor used to balance the influence of immediate and future rewards [11]. The agent learns by following a policy π, formally defined as the probability of an agent in a state s_t performing an

action a_t. The effectiveness of a given policy is established by the value-function given by Eq. (3) in MDPs' notation [11].

$$Q^{\pi}\left(s_t, a_t\right) = E_{\pi}\left\{\sum_{N=0}^{\infty} \gamma^N r_{t+N+1} \mid s_t = s, a_t = a\right\} \tag{3}$$

Many RL algorithms divide the notion of policy and value-function between two stakeholders: an *actor* and a *critic*. The actor is the policy structure that chooses the actions taken by the agent while the critic gives an estimation of the value-funciton $Q^{\pi}\left(s_t, a_t\right)$ [12].

B. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) is an off-policy algorithm first developed by [13] as a bridge between Deep Q-Network (DQN) and Actor-Critic (AC) algorithms. DDPG implements the dual *actor-critic* structure of AC algorithms together with the idea of using neural networks as function approximators from DQN. As it can be seen from Algorithm 1, both the critic $Q_{\bar{k}}$ and the actor π_{θ} are indeed represented by neural networks.

Algorithm 1 Deep Deterministic Policy Gradient [13]. Adapted from [4].

1: Initialize randomly-chosen weights θ and k for policy π_{θ} and critic $Q_{\bar{k}}$ networks, respectively
2: Initialize weights $\overline{\theta} \leftarrow \theta$ and $\overline{k} \leftarrow k$ for policy $\pi_{\overline{\theta}}$ and critic $Q_{\overline{k}}$ target networks, respectively
3: Initialize initial state s_0, random process N and smoothing factor τ
4: Initialize memory buffer $\mathcal{D}$
5: Sample initial action $a_0 \sim \pi_{\theta}\left(a_0 \mid s_0\right)$
6: **for** each step t **do**
7: Execute action $a_t = \pi_{\theta}\left(a_t \mid s_t\right) + N$
8: Sample r_t and $s_{t+1} \sim \mathcal{P}\left\{s_{t+1} \mid s_t, a_t\right\}$
9: Store transition $\left(s_t, a_t, r_t, s_{t+1}\right)$ in $\mathcal{D}$
10: Sample a random mini-batch of n transitions $\left(s_i, a_i, \tilde{r}_i, s_{i+1}\right)$ from $\mathcal{D}$
11: Compute targets: $y_i = r_i + \gamma Q_{\overline{k}}\left(s_i, \pi_{\overline{\theta}}\left(s_i\right)\right)$
12: Update critic with one-step gradient descent by minimizing the loss:

$$L = \frac{1}{n}\sum_{i=0}^{n}\left(y_i - Q_k\left(s_i, a_i\right)\right)^2$$

13: Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta} J \approx \frac{1}{n}\sum_{i=0}^{n} \nabla_a Q_k(s,a)\Big|_{s=s_i, a=a_i} \nabla_{\theta}\pi_{\theta}(s)\Big|_{s=s_i}$$

14: Update the target networks weights:

$$\overline{\theta} \leftarrow (1-\tau)\theta + \tau\overline{\theta}$$

$$\overline{k} \leftarrow (1-\tau)k + \tau\overline{k}$$

15: **end for**

The state and action spaces for DDPG are continuous, and in order to alleviate the computational strain, the algorithm uses a deterministic policy π_{θ} to map the states to specific actions.

The Q-function is updated with the loss funciton L, aimed at minimizing the mean-squared Bellman error similarly to DQN algorithms. The concept of experience replay is also used in DDPG, i.e. granting both actor and critic to be updated by sampling a mini-batch of transitions $(s_i, a_i, \tilde{r}_i, s_{i+1})$ stored in $\mathcal{D}$ otherwise known as the *replay buffer*. This allows the agent to learn from a set of tuples that are uncorrelated with each other.

When updating the critic, [13] found that automatically applying Q-learning resulted in instability of the learned policy. This is due to the fact that the Q_k used to calculate the loss function is also used to determine the target value y_i. To avoid this, target functions with copies of both actor and critic are determined with weights θ and k slightly lagging the original ones. By doing so, the target values y_i are bound to change more slowly, allowing for better learning stability. [13] pinpoints exploration as a major challenge of algorithms dealing with continuous action spaces. To guarantee sufficient exploration, the policy π_θ is supplemented with noise sampled from an Ornstein-Uhlenbeck process.

C. Safe Reinforcement Learning via Shielding

Safe reinforcement learning has been the focus of many researches in the last decades. Generally speaking, the main approaches aim to either modify the agent's exploration process by e.g. providing external knowledge or to modify the optimization criteria by e.g. constraining the learning [14]. One interesting method developed from merging external knowledge and constrained learning is **shielding** [15]. This technique allows the agent to learn without constraints in the safe state space but gives advice and can overrule an action suggested by the learning agent. A *shield* observes the states that a certain action leads to and intervenes only if the this action leads to an unsafe state space. [15] adapts the same definition of safety as [14], hence as the concept of not visiting any troublesome state. For this definition to apply, the shield must have some indication on which states are dangerous or not. Therefore, shielding uses external knowledge as well as constrained learning.

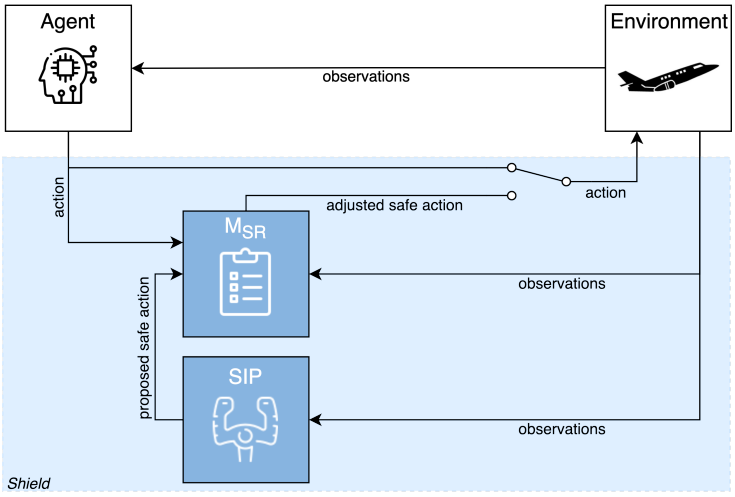


Figure 1. Control loop including the shield. The blocks in blue show the Safe Initial Policy (SIP) model and the Safety Range M_{SR} model.

In the context of this work, the shield is adapted from [16] who in turn used the technique first developed by [15]. Fig. 1 shows the control loop for a so called *post-posed shield*, where the agent is free to explore the state space without constraints until a the boundary between safe and unsafe action space is approached. The shield is composed of two separate entities, a Safe Initial Policy (SIP) subsystem and a Safety Range (M_{SR}) subsystem. The SIP can differ based on the complexity of the problem. It can span from a rule-based model to a learning agent. At time t, the agent proposes an action a_t while the shield observes the effect of a_t on the environment. The SIP receives observations from the environment and proposes a safe action a_{safe} for the

current state s_t if the latter is considered risky. This action a_{safe} is transferred to the M_{SR} which stores it at time t. Depending on the value of s_t, the M_{SR} can decide to overrule a_t with a_{safe} or with its scaled version, if the action proposed from the SIP is too conservative. If the action a_t made by the agent does not lead to unsafe states (as checked by M_{SR}), $\hat{a}_t \leftarrow a_t$ is executed; otherwise, the chosen adjusted safe action $\hat{a}_t \leftarrow a'_t$ is.

M_{SR} plays different roles. First, it determines whether a state s_t is safe for a given action a_t. Second, it receives the proposed safe action from the SIP and scales it if the action is deemed too conservative. Finally, it checks whether the action made by the agent should be overruled by $\hat{a}_t$ or not. The advantage of not always following the SIP becomes apparent when the environment approaches a risky state space. The agent is able to learn an improved policy in close to risky situations, while being able to fall back on a strictly safe policy generated by the SIP. Since the aim of the SIP is to provide a safe action whenever a risky state is being approached, it does not have to perform a tracking task particularly well. As a_t is not overruled by the M_{SR} in safe states, the performance of the agent is not always deteriorated by $\hat{a}_t$. This allows the agent to learn a policy that can maintain an adequate performance during safe operations. The adaptation in this research modifies the process of learning: the agent is not trained with the shield, but the latter is added in a post-learning setting. This allows the design of the shield to be more flexible so to better analyze the effect of this safety technique on the control task.

III. Controller Design

Having discussed the fundamental concepts used in this research, this section will discuss their integration with the flight controller. Section III.A will introduce the simulation model, Section III.B will discuss the development of the DDPG algorithm and the shield. Finally, Section III.C will give an overview of the training process.

A. Cessna Citation 500 Model

In this research, the environment that the DDPG agent will control is a high-fidelity non-linear simulation of a Cessna Citation 500 business jet aircraft [4]. The model was built by researchers at the Delft University of Technology as the need of a standard flight CAD package for control purposes arose. The Delft University Aircraft Simulation Model and Analysis Tool (DASMAT) is the virtual copy of the dynamics of the Cessna Citation 500 PH-Lab, shown in Fig. 2.

Figure 2. Cessna Citation 500 PH-LAB. [†]

The dynamics model used in this research is non-linear and trimmed upon initializing the simulation. The aircraft has twelve states as shown in Eq. (4) and three control inputs given in Eq. (5).

$$x = [p, q, r, V_{\text{TAS}}, \alpha, \beta, \theta, \phi, \psi, h]^\top \tag{4}$$

[†]Image from A. Wilson (with permission).

$$u = [\delta_e, \delta_a, \delta_r]^\top \tag{5}$$

The aircraft is trimmed for flight conditions that represent a standard top of descent maneuver. The values of x_0 and u_0 can be found in Table 1.

Table 1. Initial states and control inputs.

State	Initial value	Unit
Roll rate p_0	0	[deg/s]
Pitch rate q_0	0	[deg/s]
Yaw rate r_0	0	[deg/s]
True airspeed V_{TAS_0}	150	[m/s]
Angle of attack α_0	2.4	[deg]
Angle of sideslip β_0	0	[deg]
Roll angle ϕ_0	0	[deg]
Pitch angle θ_0	2.4	[deg]

State	Initial value	Unit
Geometric altitude h_{e_0}	10000	[m]
Horizontal position along Earth x-axis x_{e_0}	0	[m]
Horizontal position along Earth y-axis y_{e_0}	0	[m]
Control Input	**Initial value**	**Unit**
Elevator deflection δ_e	-1.0	[deg]
Rudder deflection δ_r	0	[deg]
Airleron deflection δ_a	0	[deg]

B. Shielded DDPG Controller Design

The shielding framework aims to perform flight path angle tracking in a safe manner without providing the main learning agent with information about unsafe conditions. As mentioned in Section I, flight path control has been chosen as the last step of EASA's stall recovery procedure dictates to return to the indicated flight path angle γ. Therefore, it is interesting to develop a controller able to return to the intended γ after reaching risky state spaces.

The shielding framework can be visualized in Fig. 3 where in black is the loop used by the DDPG agent to train while in blue is the post-training addition of the shield. The DDPG agent will be tasked with controlling the flight path angle γ of the Cessna Citation 500 by outputting a reference pitch rate q_{ref}. The agent trains with the aim of minimizing the tracking error $\gamma_e = \gamma_{\text{ref}} - \gamma$. The reward function is the negative of the absolute value of γ_e, as given by Eq. (6).

$$r = -|\gamma_e| \tag{6}$$

From the reward function one can gather that the agent does not have any information regarding safe or safety critical states. This allows the agent to reach an optimal policy for tracking γ_{ref} without the constraints or knowledge required to maintain safety. The agent is provided six observations to understand the aircraft's dynamics and to monitor its performance with respect to the reward acquired. The first observation is the flight path angle error γ_e, useful for the agent to distinguish between an adequate or a poor performance. The second observation is the elevator deflection angle δ_e as the action generated by the agent q_{ref} results in an otherwise unknown actuator angle. The third and fourth observations are the pitch angle θ and the angle of attack α, as they are directly influence the flight path angle according to the relation $\gamma = \theta - \alpha$. The fifth observation is the pitch rate q as it directly affects θ. Finally, the last observation is the reference pitch rate $\hat{q}_{\text{ref}}$. During learning, $\hat{q}_{\text{ref}} = q_{\text{ref}}$ as the shield is not active. However, once the system is simulated, the action proposed by the agent may not be the one fed to the aircraft dynamics model. Therefore, it is crucial for the agent to know whether its proposed action is adequate or not and what action should be suggested at the next time step. The agent has been designed with the hyperparameters given by [13] and [17]. A summary is presented in Table 2. Most of the hyperparameters are taken directly from [13] as a research on hyperparameters tuning was not the aim of this study. However, in an attempt to improve the training time, the batch size was increased to 1024 as suggested by [17].

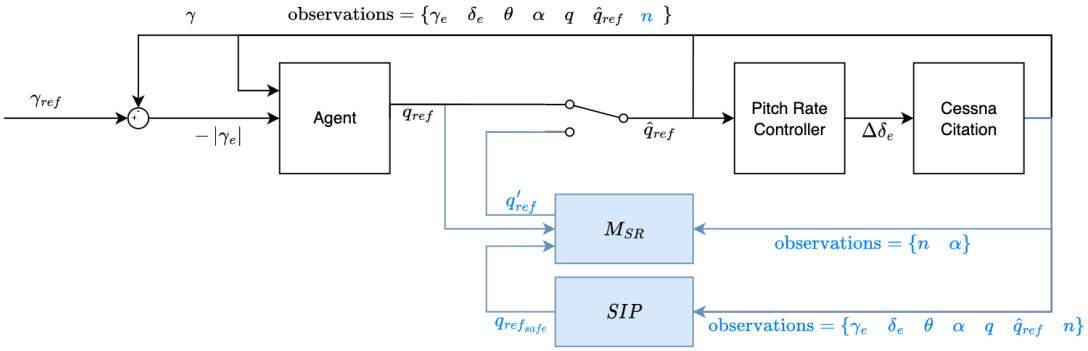


Figure 3. Shielding framework for safe flight path angle tracking.

The DDPG agent's topology also follows the design given by [13]. Both the actor and the critic are composed of two hidden layers using rectified non-linearity (ReLu) activation function with 400 and 300 units respectively. Similarly to [13], actions were not included in the network until the second hidden layer of the Q-network. One major difference from the default design is the addition of a normalization layer in the penultimate feature of the actor network. According to [18], by normalizing the penultimate layer of the actor network, variance in the training results can be drastically reduced and training time can be decreased. The actor network uses an output tanh layer to bound the actions of the agent. The parameters defining the Ornstein-Uhlenbeck exploration process follow the default values as defined in [13] and are given in Table 2.

Table 2. Hyperparameters for DDPG agent as given by [13] and [17]. The same hyperparameters are used for the training of the SIP agent described in Section III.B.1.

Parameter	**Value**
Discount factor γ	0.99
Mini batch size $\lvert\mathcal{B}\rvert$	128
Experience buffer length	10^6
Target smooth factor τ	0.001
Optimizer	Adam
Ornstein-Uhlenback mean attraction constant θ	0.15
Ornstein-Uhlenbeck variance σ	0.10

Actor		**Critic**	
Parameter	*Value*	*Parameter*	*Value*
Learn rate λ	$1 \cdot 10^{-4}$	Learn rate λ	$1 \cdot 10^{-3}$
Gradient Threshold	1	Gradient Threshold	1
L2 Regulatization factor	$1 \cdot 10^{-5}$	L2 Regulatization factor	$2 \cdot 10^{-4}$
Hidden layers	2	Hidden layers	2
Layers activation	ReLu	Layers activation	ReLu
Neurons L1/L2	400/300	Neurons L1/L2	400/300

The controller described above and represented by the *Agent* block on Fig. 3 does not, by itself, provide any notion of safety nor learns a policy that keeps the agent in a safe state space. The shield shown in blue in Fig. 3, is implemented in the loop after training is complete. This additional system consists of two elements: the Safe Initial Policy Model (SIP) and the Safety Range Model M_{SR}.

1. Safe Initial Policy Model

The Safe Initial Policy (SIP) model is actively in charge of proposing a safe action to the DDPG agent tracking the flight path angle γ_{ref}. Depending on the complexity of the safety problem, the SIP can vary from a simple rule based model to a more intelligent one. The control problem to be handled in this research is considered of high complexity, due to the non-linear nature of the model, the amount of variables involved and their coupled nature. The proposed SIP is a previously trained agent able to propose an action a_{safe}. The agent is defined with the same hyperparameters and network as the main DDPG agent described above in Section III.B to avoid issues related to compatibility. When referring to Fig. 3, the SIP agent is trained with the same black loop as the DDPG agent but with the addition of n in the observation matrix, shown in blue. This is deemed

necessary as n is an integral part of the reward function and gives the agent knowledge about its performance.

The agent is trained according to the constrained criterion proposed by [14]. During learning, the agent aim is to track the flight path angle γ_{ref} while avoiding violating certain safety constraints. In the context of this work, the agent learns to track γ_{ref} without violating constraints in the angle of attack α and the load factor n. The angle of attack is the main parameter that influence the stall pattern of an aircraft which, according to [19], is still a main contributor to overall aviation accidents. Additionally, the load factor is considered relevant as going over certain limits can be dangerous to both passengers [20] and to the aircraft's structural integrity as discussed in [21].

The SIP agent is trained with notions of α_{stall} and n_{crit} as part of the reward function. Similarly to [7], the reward function is composed of different parts each characterizing different notion to learn and is described as follows:

$$r = R_t + R_\alpha + R_n \tag{7}$$

The first part is R_t related to learning to track γ_{ref} with sufficient accuracy. This is the same reward function as for the DDPG agent described in Eq. (6). The second part is R_α, given in Eq. (8). According to [22], a CS-25 aircraft should include an enhanced stall protection mechanism in their flight controllers. A controller on board a CS-25 type aircraft, a class to which the Cessna Citation 500 belongs to, should not allow the angle of attack to exceed the α_{stall}. The choice of α_{stall} follows from [20]. R_α ensures that the agent learns to track the signal without exceeding the critical angle of attack α_{stall} of 20 degrees and without turning α into a negative angle which is set arbitrarily as the lower bound.

$$R_\alpha = \begin{cases} 0 & \text{if } 0^\circ < \alpha < 20^\circ \\ -|\alpha| & \text{otherwise} \end{cases} \tag{8}$$

The last part of the reward function is R_n. It awards a reward if the agent is able to track the reference γ_{ref} without exceeding arbitrarily set limits in the load factor as shown in Eq. (9). According to [23],

> *"the positive load factor command limit with electronic flight control system (EFCS) functioning in its normal mode and the airplane in its normal trim state for the flight condition must not be more than 2.5 g with the high-lift devices retracted, and 2.0 g with the high-lift devices extended."*

To allow the agent's policy to be robust to both configurations, it is chosen to have a tighter upper bound so that the constraint can be true in both cases. In this research, the lower bound of the load factor is set 0.35 while the upper one is set to 2.

$$R_n = \begin{cases} 0 & \text{if } 0.35 < n < 2 \\ -|n| & \text{otherwise} \end{cases} \tag{9}$$

It should be noted that both $|\gamma_e|$ and $|\alpha|$ in Eq. (6) and Eq. (8) are in radians. This is done in order to better correlate the observations and the reward scheme.

2. Safety Range Model

Once the SIP proposes a safe action, it may not be necessary to overrule the action of the DDPG agent as an action a_t is only overruled if it leads to an unsafe state. Therefore, there is the need of a subsystem that decides which action should be fed to the pitch rate controller and in turn to the elevator actuator. The safety range model M_{SR} is a subsystem of the shield in charge of deciding what action is sent to the environment. This model is a simple rule based system, however it is crucial for the shield to work smoothly.

The M_{SR} decides which action $\hat{a}_t$ will be passed to the environment by observing two different states:

the angle of attack α and the load factor n. The M_{SR} receives the values of α and n at a time t and compares them to set values that are considered safe. Fig. 4 shows a simplified representation of how the states are categorized by the M_{SR}. The M_{SR} makes decisions based on in which state space $\mathcal{S}$ the state s_{t+1} will be. First, if the agent at state s_{t+1} would stay in the green area, the action $\hat{q}_{\text{ref}}$ sent to the pitch rate controller is

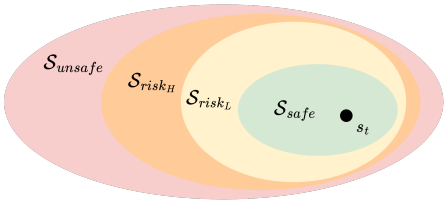


Figure 4. Graphic representation of safe (green), low risk (yellow), high risk (orange) and unsafe (red) state spaces $\mathcal{S}$.

the same action q_{ref} as the one proposed by the DDPG agent. Second, if the agent at state s_{t+1} would go from $\mathcal{S}_{\text{safe}}$ to a low risk state space $\mathcal{S}_{\text{risk}_L}$,it means that the action proposed by the agent could be on the trajectory towards and unsafe state space. However, the action proposed by the SIP might be too drastic and could lead to a premature deterioration in tracking accuracy. For this reason, if $s_{t+1} \in \mathcal{S}_{\text{risk}_L}$ then $\hat{q}_{\text{ref}} \leftarrow q'_{\text{ref}}$ is a scaled version of $q_{\text{ref}_{\text{safe}}}$. The scaling factor k is decided arbitrarily so to avoid jumps in pitching rate and is smaller than 1. Finally, if the agent's state $s_{t+1} \in \mathcal{S}_{\text{risk}_H}$, the action from the SIP is fed to the pitch rate controller, hence $\hat{q}_{\text{ref}} \leftarrow q'_{\text{ref}} \leftarrow q_{\text{ref}_{\text{safe}}}$. This way, if the agent is in a high risk state space $\mathcal{S}_{\text{risk}_H}$, the safe action can help redirect the agent towards a lower risk state space. By doing so, the shield avoids the agent reaching an unsafe state space $\mathcal{S}_{\text{unsafe}}$.

The M_{SR} checks for the conditions described above for both α and n with two equal and parallel systems. Each system independently analyses the states and compares them to the safety constraints. For clarity, the variables $\hat{q}_{\text{ref}_\alpha}$ and $\hat{q}_{\text{ref}_\text{n}}$ given in Eq. (10) and Eq. (11) can be defined. Each represent the end product of the two subsystems in the M_{SR}.

$$\hat{q}_{\text{ref}_\alpha} = \begin{cases} q_{\text{ref}} & \text{if } 2^\circ \leq \alpha \leq 8^\circ \\ k \cdot q_{\text{ref}_{\text{safe}}} & \text{if } 1^\circ \leq \alpha < 2 \quad \text{or} \quad 8 < \alpha \leq 12 \\ q_{\text{ref}_{\text{safe}}} & \text{otherwise} \end{cases} \tag{10}$$

$$\hat{q}_{\text{ref}_\text{n}} = \begin{cases} q_{\text{ref}} & \text{if } 0.8 \leq n \leq 1.2 \\ k \cdot q_{\text{ref}_{\text{safe}}} & \text{if } 0.5 \leq n < 0.8 \quad \text{or} \quad 1.2 < n \leq 1.5 \\ q_{\text{ref}_{\text{safe}}} & \text{otherwise} \end{cases} \tag{11}$$

If both $\hat{q}_{\text{ref}_\alpha}$ and $\hat{q}_{\text{ref}_n}$ are equal, the M_{SR} will not have to make a choice between two different options. If the two values are not the same, the M_{SR} will always chose the *safest* option between $\hat{q}_{\text{ref}_\alpha}$ and $\hat{q}_{\text{ref}_n}$. In this research, the unscaled value proposed by the SIP $q_{\text{ref}_{\text{safe}}}$ is always considered the safest action. The majority of the intermediate values at which the different conditions for Eq. (10) and Eq. (11) are defined, have been chosen arbitrarily during the design of the controller. Although the upper limits for both the angle of attack and load factor are defined by previous research as given in [20], or by governing bodies as explained in Section III.B.1, these limits have been set to a lower value to provide some leeway for stall prevention and to be more relevant for the flight envelope presented during the agent's training.

C. Training

In this research, the shield will be deployed post-learning, meaning that the DDPG agent will be trained before the shield is inserted in the loop. Hence, the DDPG agent will train with the model shown in Fig. 3

with the blue selection, representing the shield, excluded from the system. This effectively removes any notion of safety in the loop and sets $q_{\text{ref}} = \hat{q}_{\text{ref}}$. Training is performed with short episodes of 20 seconds each whilst the sampling rate is set to 100 Hz. The sampling rate matches the DASMAT model sampling rate to avoid implementation issues. The reference flight path angle γ_{ref} to train with has been chosen as to model a conventional step down approach designed according to [24]. The reference flight path angle has been modeled as follows:

$$\gamma_{\text{ref}} = \begin{cases} 0^\circ + x & \text{for } 0 < t \leq 5 \\ -3^\circ + x & \text{for } 5 < t \leq 15 \\ 0 + x & \text{for } 15 < t \leq 20 \end{cases} \tag{12}$$

where x is a random value drawn from a uniform distribution of interval [-0.5; 0.5] degrees upon initialization. This allows the trained agent to generate a more robust policy. The rest of the states were initialized as the trimmed conditions given in Table 1.

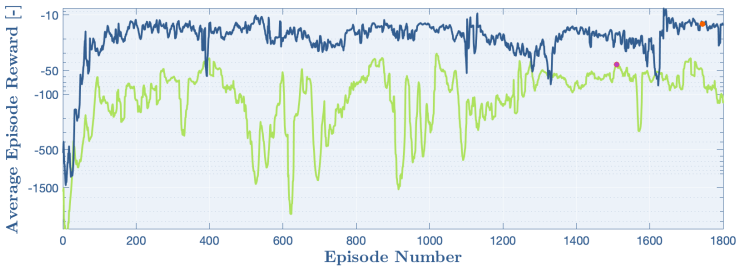


Figure 5. Average reward during training of a DDPG agent (in blue) and SIP agent (in lime green) for flight path angle tracking. The policies used for reproducing the results shown in this paper are at episode 1744 (marked in orange) and episode 1510 (marked in purple).

The results of the training for the DDPG agent is shown in blue in Fig. 5 where a successful policy is found at episode 1744. It can be seen that the learning follows a standard training curve typical of DRL agents solving complex control tasks [8]. A high rewarding plateau is reached around episode 1650, suggesting that the agent is sufficiently trained. The SIP, whose characteristics, such as the observations, reward function and networks hyperparameters are given in Section III.B.1, is trained in an analogous manner as the DDPG agent. The results of the training can be seen in lime green on Fig. 5. It is necessary to remember that the reward function consists of three different parts, hence an episode with similar average rewards can highly differ in terms of performance. In this case, a policy was deemed successful if it was able to remain within the safety limits imposed in the reward function while following, albeit with higher tracking error, the reference flight path angle. It can be seen that the average reward greatly oscillates up until the 1100th episode, before converging to a plateau.

IV. Results

The response of the Cessna Citation 500 model to a conventional step down approach with an unshielded and shielded DDPG controller are discussed in this section. First, the results of the simulation without the shield are evaluated in Section IV.A. Second, the results of the SIP agent training are presented in Section IV.B to show the ability of the agent to perform its intended task. Third, the response for the complete model including the shield are presented in Section IV.C. Fourth, the performance of the unshielded and shielded controllers are compared in Section IV.D. Finally, the robustness of the controllers is assessed in Section IV.E.

$$\text{nMAE\%} = \frac{\text{mean}(|\gamma_e|)}{\text{mean}(|\gamma_{\text{ref}}|)} \tag{13}$$

In this research, the tracking accuracy is determined by the normalized Mean Absolute Error percentage (nMAE%) calculated as shown in Eq. (13).

A. Flight Path Tracking with Unshielded DDPG Controller

The aim of this research is to develop a controller able to safely track the reference flight path angle of a Cessna Citation 500 at top of descent. As mentioned in Section III.B, the safe RL controller is deployed post-learning, while the tracking task is delegated to a DDPG controller with no notion of safety. In the upcoming sections, the results of the learning controller tasked solely track γ_{ref} will be discussed.

1. Nominal Flight Conditions

The responses shown in Fig. 6 are the results of the DDPG controller trained according to Section III.C. The results shown are for a profile for which the DDPG agent was not trained. The reference flight path angle varies between 0 degrees and steps of -3, -4 and -2 degrees respectively. This allows the altitude h to mimic a step down approach where the altitude is gradually decreased after top of descent. The γ_{ref} tracking is performed with a small steady state error throughout the 100 seconds of runtime. Although for $t \leq 5s$ the error is less than 0.02 degrees, the maximum error reached for the remaining of the simulation is 0.4 degrees at $t = 60s$ during the horizontal portions of the signal. This consistent steady state error might be caused by either the policy learned by the agent or by a non-optimal PID tuning of the pitch rate controller. However, as already mentioned, the error is small for $t \leq 5s$ where $\gamma_{\text{ref}} = 0$. These are the same conditions the agent is originally trained for, as given in Eq. (12). Therefore, the steady state error is most likely caused by a partially faulty policy. A solution to reduce the steady state error, proposed by [25], is to include the integral error of

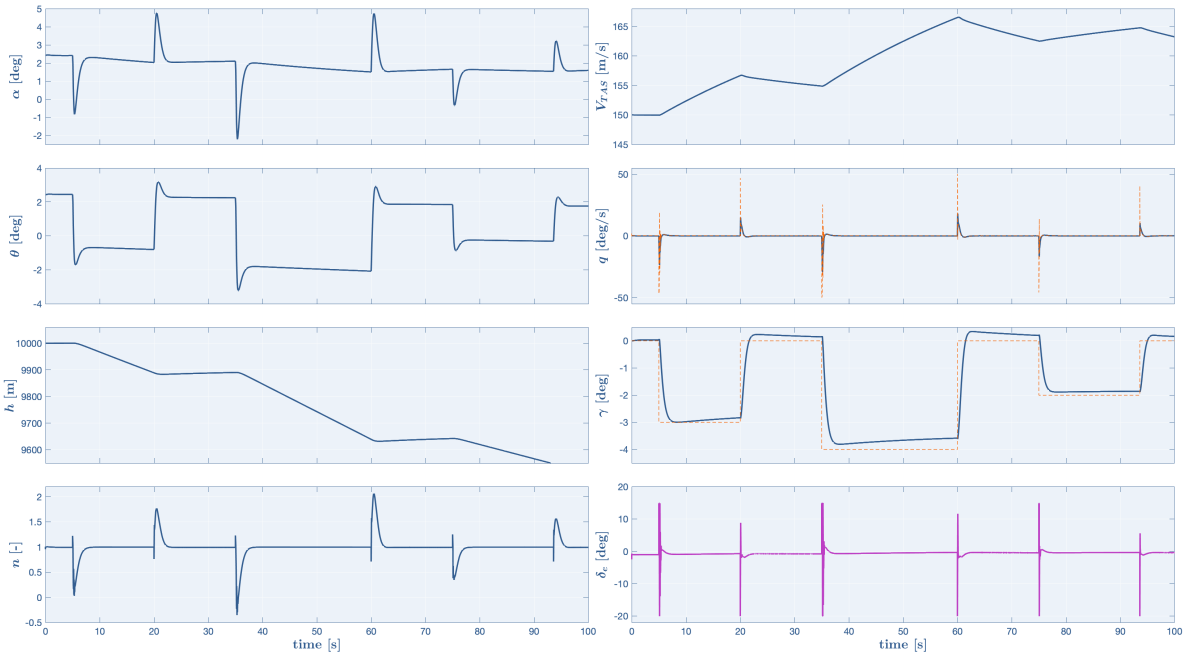


Figure 6. Flight path tracking with unshielded DDPG controller in nominal flight conditions. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple.

the flight path angle as an observation. The DDPG controller generally shows tracking with less oscillatory behavior and less overshoot after the reference angle drastically changes than research with similar control

tasks such as [6] and [25]. This might be due to the longer training times: the DDPG agent in this research has been trained 18 times longer than the agent proposed by [25]. The controller performs well in terms of tracking. The nMAE% is equal to 17.1%, showing that the DDPG controller can successfully track γ_{ref}. The maneuver implies harsh changes in flight path angle which in turn result in large changes in θ and α. The variation of angle of attack is particularly interesting: as this agent holds no notion of safety, α clearly crosses the boundaries of safety defined in Section III.B.2. In this maneuver, the angle of attack does not reach the value of α_{stall} but does indeed cross the lower bound of safety discussed previously. This breach in safety can be also seen in the response of the load factor n. Not only does the load factor reach a value of 2 at $t = 60s$ but it also crosses 0 reaching values of -0.35. This is clearly not ideal for passenger safety nor structural integrity of the aircraft. Overall, the percentage of simulation time during which either α or n are in a high risk state space $\mathcal{S}_{\text{risk}_H}$ is 2.5% while the agent is 5.9% of times in an unsafe state space. Prolonged time in a unsafe state space is not ideal, therefore this issue will be addressed by implementing the shield.

2. Turbulent Flight Conditions with Biased Sensor Noise

The working of the DDPG agent is also tested in a more realistic environment. To increase the fidelity of the model with respect to a real-world application, turbulence is introduced in the environment and biased sensor noise is added. The turbulence model used in this research is developed by [26] and referred to as a *patchy turbulence model*. The term *patchy* refers to the non-Gaussian nature of the turbulent flow. The turbulent field enforced is symmetric and has non-zero gust velocities u_g and w_g. The variance of the gust velocities σ is set to 1.3, mimicking severe turbulent conditions as given by [27]. The turbulence model proposed by [26] comes from efforts in making flight simulators more realistic [28]. The biased sensor noise is added according to [29] to pitch rate and angle sensors, to the elevator deflection sensor as well as the angle of attack sensor. An overview of the values used can be seen in Table 3.

Table 3. Cessna Citation PH-LAB aircraft sensor characteristics. Values retrieved from [29].

Observed state	δ_e **[rad]**	θ **[rad]**	α **[rad]**	q **[rad/s]**
Noise σ^2	$5.5 \cdot 10^{-7}$	$3.2 \cdot 10^{-5}$	$4.0 \cdot 10^{-10}$	$6.3 \cdot 10^{-4}$
Bias	$2.4 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	[-]	$3.0 \cdot 10^{-5}$

The performance of the trained DDPG agent in turbulent flight conditions is shown in Fig. 7. It can be seen that the flight path angle tracking shows a similar behaviour as in the nominal conditions shown in Fig. 6. The response still shows a steady state error, especially pronounced for $60s \leq t \leq 75s$. The angle of attack shows a slight amplified behaviour with respect to its nominal condition counterpart. This can be traced back mostly to the effects of turbulence as α is defined with respect to the oncoming flow. The load factor however, shows a greater noisy trend. The amplitude of oscillations is small but of high-frequency. These high-frequency noisy oscillations can be attributed to the action proposed by the agent q_{ref}, which in turn introduces disturbances in the elevator deflection and in the load factor. This is clearly due to the poor training of the agent in trading off control efforts and tracking accuracy. An idea to improve this behaviour would be to add a control effort penalty in the reward function of the agent described in Section III.B. When looking at the safety limits, the upper bound of n increases by 0.1 while the lower bound further decreases by -0.3 compared to the nominal conditions. The amplification of the peaks is mostly due to the turbulent flow. The agent spends 2.1% of the simulation in $\mathcal{S}_{\text{unsafe}}$, less than in the nominal flight conditions. In terms of tracking performance, the controller's nMAE% of 16.1% is slightly lower than the nMAE% in nominal conditions which has a nMAE% equal to 17.1%, showing that the controller is robust to atmospheric disturbances and sensor noise. Although not trained for turbulent environment, sensor noise or for this particular flight profile, the trained agent is still able to track γ_{ref} and is therefore considered a good candidate for the shield implementation.

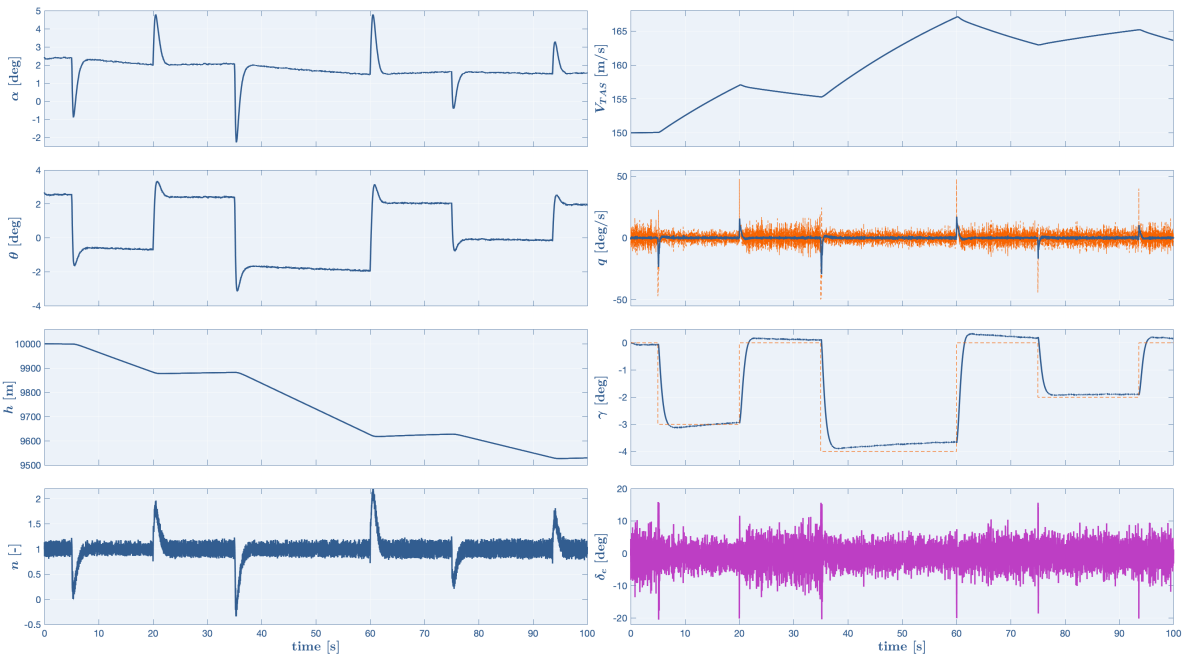


Figure 7. Flight path tracking with unshielded DDPG controller in turbulent flight conditions and added biased sensor noise. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple.

B. Safe Initial Policy as Controller

In Section III.B.1, the model in charge of proposing a safe action for each time step t was described. In this research, the SIP takes the shape of a trained agent, designed analogously to the controller described in Section III.B but with a modified reward function and an additional observation variable to account for safety. The results of the training can be seen in Fig. 8. It should be noted that the flight profile shown below is not the profile with which the agent was trained. As can be seen, the agent successfully tracks the reference flight path angle although its accuracy greatly decreased with respect to the results on the DDPG agent shown in Fig. 6. The reason can be attributed to the necessary trade-off the agent has to make between the performance section of the reward function R_t and the safety ones R_n and R_α. When analyzing the responses, it can be seen that the learned policy successfully avoids the unsafe state space for both the angle of attack and the load factor. The agent avoids $\mathcal{S}_{\text{risk}_H}$ as well as $\mathcal{S}_{\text{unsafe}}$ and remains in $\mathcal{S}_{\text{risk}_L}$ and $\mathcal{S}_{\text{safe}}$ during the whole simulation time. The response of the angle of attack in Fig. 6 reaches a maximum of 4.8 degrees and a minimum of 2.2 degrees. In contrast, the SIP agent not only lowers the maximum peak to 3.7 degrees but most notably has minimum peak of 1 degree. A similar trend can be seen for the load factor. The maximum load factor has a value of 1.8 and minimum of 0.7. The results of the SIP show great improvements as the maximum load factor is decreased by 0.2 while the minimum is increased by 1.1. This demonstrates that the agent learned a policy that complies with the safety specifications while still following the reference flight path angle. The improved safety comes at the price of tracking performance. Comparing the performance of the SIP and the nominal DDPG case, it can be seen that the DDPG controller's nMAE% is around 17.5%, which is lower than its safe counterpart with a nMAE% of 34.6%. In fact, the SIP agent in nominal conditions performs the worst out of all options given in Table 4. Therefore, although the safety specifications are met, the tracking performance of this agent can not produce results that could be confidently used as a main controller for real life applications.

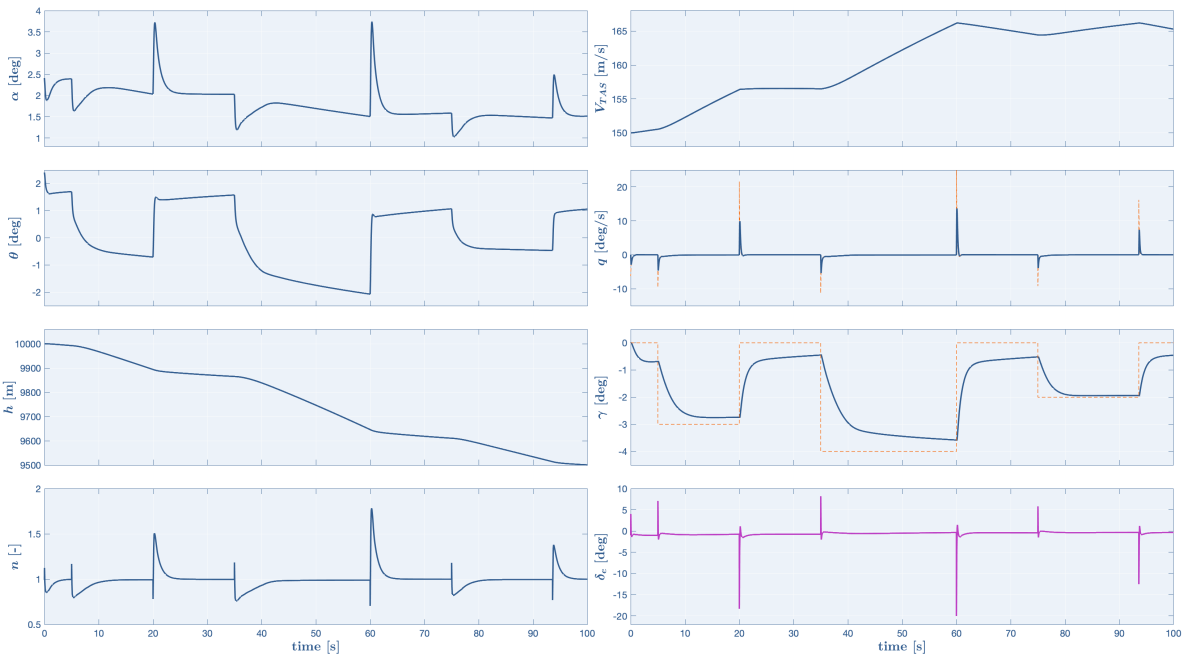


Figure 8. Flight path tracking with SIP controller in nominal flight conditions. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple.

C. Complete Shield

The focus of this research is to develop a flight path controller assisted by a shield, introduced to improve safety without sacrificing tracking performance. The shield has been implemented according to the description in Section III.B.1 and Section III.B.2.

1. Nominal Flight Conditions

The shield is tested on the same flight conditions as the DDPG agent is subjected to on Section IV.A.1. During the simulation, the shield is on for 68% of the time. This allows the controller to maintain the aircraft within safety limits without compromising γ_{ref} tracking. For $t \leq 5s$, the flight path angle is being tracked analogously to the response of Fig. 6. However, once the reference γ rapidly decreases from 0 degrees to -3 degrees, the agent quickly pitches down to follow the reference. The M_{SR_n} is the first to be shortly triggered while the M_{SR_α} remains on for more than 5 seconds. From the lower plots of Fig. 9, it can be seen that M_{SR_α} selects first $q_{\text{ref}_{\text{safe}}}$ and second $k \cdot q_{\text{ref}_{\text{safe}}}$. The activation of the shield maintains the angle of attack at a safe lower bound of 1 degree and the load factor of 0.5; this however worsens the tracking of γ_{ref}. At $t = 8s$, the shield oscillates between the scaled $q_{\text{ref}_{\text{safe}}}$ and the action proposed by the DDPG agent. This occurs due to α moving between different regions of Eq. (10) in a short span of time. The quick switch between the two actions can clearly be seen in the pitch rate signal. This results in oscillations in the elevator deflection and consequently in many of the state responses shown in Fig. 1.

The introduction of this disturbance in the responses however, does not particularly affect the tracking nor the safety performance of the agent. Overall, the shape and magnitude of the responses with a SIP controller shown in Fig. 8 are not much different from the responses in Fig. 6. This way, when going from one action to another, the fluctuation in q_{ref} is kept small. Additionally, it has been explained in Section III.B.1 that the choice of introducing a scaled version of $q_{\text{ref}_{\text{safe}}}$ allows for actions that are less drastic in terms of

tracking performance to be selected when the shield is on. Therefore, the controller designer has the choice of modifying the scaling factor k either to minimize the introduced noise or to provide an action as close as possible to $q_{\text{ref}_{\text{safe}}}$ when the shield is triggered. Between 10 seconds and 20 seconds, the shield is not on, hence the flight path angle is well tracked. Afterwards, the aircraft pitches up to return to $\gamma_{\text{ref}} = 0°$. To avoid an unsafe value of n, the shield is activated. The responses show the same noisy behaviour as just discussed. After the γ_{ref} has returned to zero, the shield is inactive. At $t = 35s$, a more aggressive maneuver is started, where $\gamma_{\text{ref}} = -4°$ is started. Here, the shield is immediately activated due to α reaching a risky state. For the remainder of the simulation, the shield is mostly on as the angle of attack is consistently below 2 degrees.

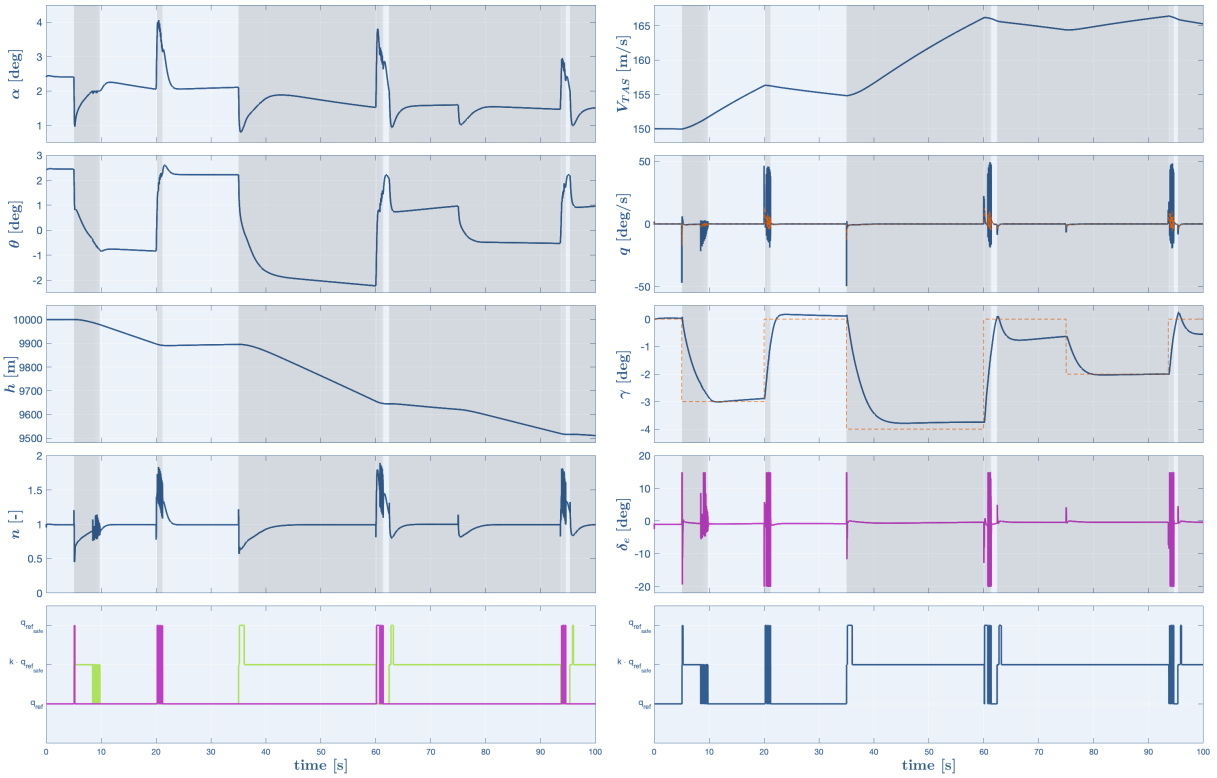


Figure 9. Flight path tracking with shielded DDPG controller in nominal flight conditions. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

At $t = 62s$, the shield is briefly off. The flight path angle reaches γ_{ref} before dropping to a lower value due to the shield being on to avoid unsafe values of α to be reached. This is a clear example of the advantage of using shielding: the trade-off between safety and performance is dynamic, allowing the controller to switch between different modes smoothly. By introducing the shield post-learning, the designer can adjust many aspects of the shield, such as the scaling factor and the safety limits imposed by M_{SR} or choice of response performance. The nMAE% of the shielded controller at nominal flight condition is 24.0%. This value is lower than the results of the SIP at the same IFC, making it clear that implementing a shield is a better solution in terms of tracking. The shield allows both α and n to always avoid $\mathcal{S}_{\text{unsafe}}$ and remain withing the safety specifications. The agent spends 1.5% of simulation time in $\mathcal{S}_{\text{risk}_H}$. This percentage can be decreased and potentially eliminated by increasing the bounds for which $\mathcal{S}_{\text{risk}_L}$ is defined as the designer can choose post-learning how much the agent is allowed to visit states that are close to the unsafe boundary. Therefore, since the shield allows the agent to perform the tracking task without exceeding the limits set in Eq. (10) and

Eq. (11), the shielded controller is considered successfully implemented.

2. Turbulent Flight Condition with Biased Sensor Noise

The same turbulent flight conditions and biased sensor noise discussed in Section IV.A.2 are applied to the Cessna Citation 500 model equipped with a shielded DDPG agent. This will allow the shield to be tested in a more realistic set-up. The results of the simulation can be seen in Fig. 10. As expected, the shield is triggered more often in a realistic environment due to higher disturbances. In this simulation, the shield is on 70% of the time, a slight increase from the nominal case. The performance of the controller is similar to the case discussed above in Fig. 9 proving robustness to real-life disturbances. More notably, the biased sensor noise affects the response strikingly more when the shield is off as can be seen by the amplitude of δ_e. This can

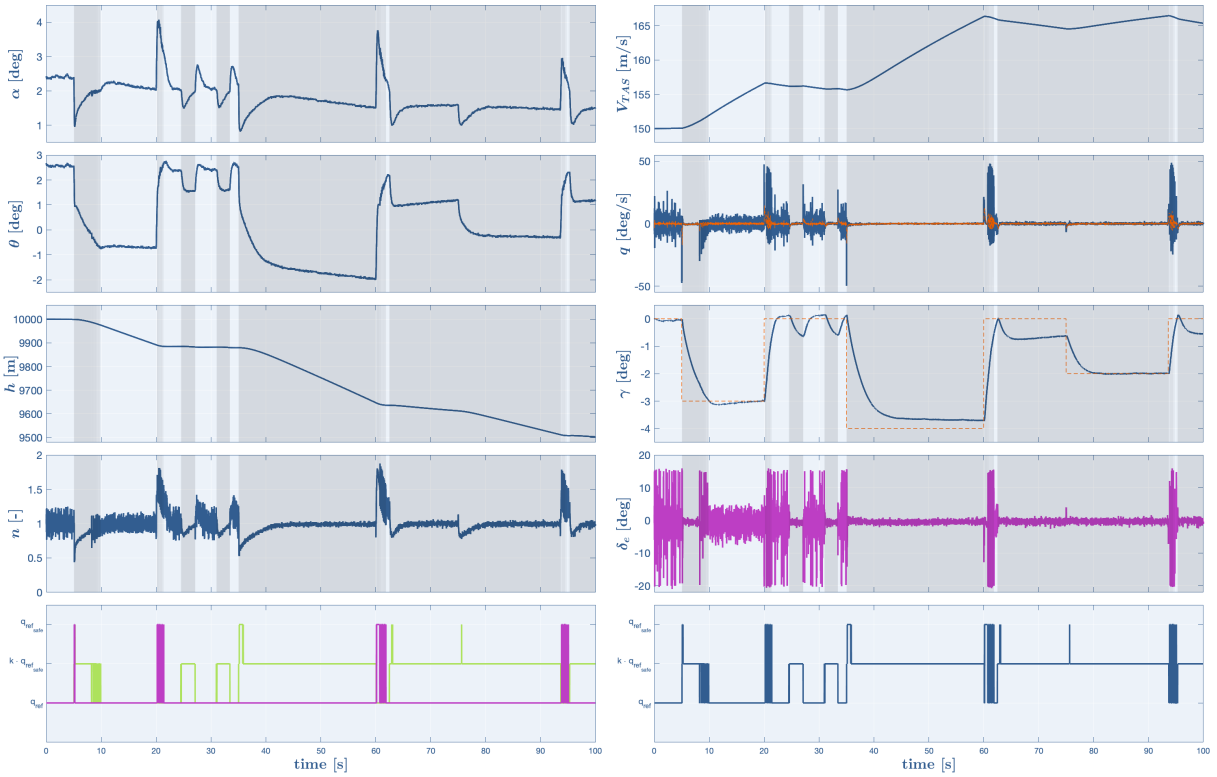


Figure 10. Flight path tracking with shielded DDPG controller in turbulent flight conditions and added biased sensor noise. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The $M_{\mathbf{SR}_\alpha}$ is shown in lime green while the $M_{\mathbf{SR}_n}$ in purple. The final $M_{\mathbf{SR}_\alpha}$ is also shown in blue. The grey areas represent the times at which the shield is actively on.

be attributed to the performance of the SIP agent. As the agent has learned to fly within certain limits both in load factor and angle of attack, it is more diligent with the control of the q_{ref} and in turn of the elevator deflection. The addition of atmospheric disturbances and sensor noise do not diminish the effectiveness of the shield. Both n and α are kept well below the safety limits in contrast with the unshielded counterpart discussed in Section IV.A.2. The agent remains in $\mathcal{S}_{\text{risk}_H}$ only 0.9% of the time and always avoids $\mathcal{S}_{\text{unsafe}}$. The tracking performance does decrease as the nMAE% is 26.2%, but it is a direct consequence of the shield being on for a longer section of the simulation. This result gives additional confidence to the efficacy of the shield in close to real-life conditions.

D. Comparison between Shielded and Unshielded Controllers at Nominal Flight Conditions

In Section IV.C.1, an analysis on the γ_{ref} tracking with the shield implemented was carried out. The conditions were analogous to the ones described in Section IV.A.1. A comparison between the responses for the shielded and unshielded controllers for the nominal flight conditions can be seen in Fig. 11.

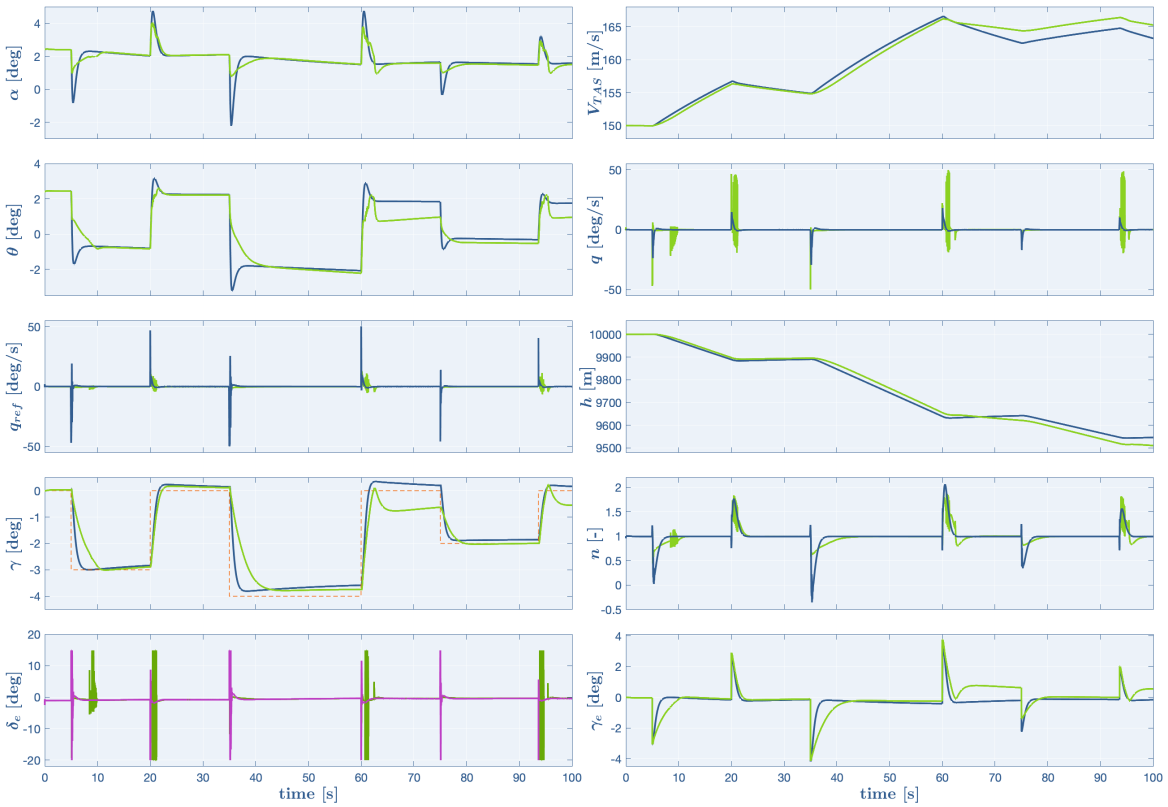


Figure 11. Comparison between flight path tracking of shielded and unshielded DDPG agent. The aircraft responses with no shield implemented are shown in blue. The shielded counterpart are shown in lime green. The control inputs for the unshielded case are shown in purple, while the shielded counterpart in dark green.

As already mentioned, the γ_{ref} tracking is sufficiently well performed by both models. From Table 4, it can be seen that the nMAE% for the unshielded case in the nominal flight conditions is of 17.1% while the nMAE% of its shielded counterpart is of 24.0%. These values support the γ_e plot shown in Fig. 11. The shielded controller is less accurate due to the SIP action being chosen for most of the simulation as shown in Fig. 9. In fact, when the system is controlled solely by the SIP agent as shown in Fig. 8, the nMAE% increases by almost two times with respect to the unshielded case in Fig. 6. This goes to show that when looking solely from a tracking performance standpoint, the SIP does not present a viable solution. By implementing the shield, the tracking performance with respect the nominal DDPG case in Fig. 6 worsens, but by a low enough quantity that it can still be considered successful. Therefore, in terms of safety, the addition of the shield provides a clear improvement in both the angle of attack and the load factor. From Table 4, it can be seen that shielded controller in nominal flight conditions and in turbulent ones outperform the unshielded DDPG controller in terms of safety. In both cases, the shield never allows either α or n to enter $\mathcal{S}_{\text{unsafe}}$, unlike the unshielded counterpart, as $\mathcal{S}_{\text{risk}_H}$ is only visited 1.5% and 0.9% of times. This represents a clear decrease from the unshielded cases showing that the agents remains consistently at a safe or low risk state space. The angle of attack is consistently lower when the shield is implemented. The maximum α in Fig. 6 reaches

4.7 degrees while the minimum is -2.1 degrees. With the addition of the shield, the aforementioned peaks in α are decreased to 4 degrees and 0.8 degrees. While the upper bound did initially not lead to an unsafe state to be reached, the improvement of the lower bound of α makes sure that the shielded response does not violate the safety requirements. The load factor's response also improves in terms of safety with the addition of the shield. From Fig. 11, it can be seen that the maximum load factor is decreased from 2.1 to 1.8 while the minimum is increased from -0.4 to 0.6. Clearly, the shield allows the controller to perform its task while maintaining safety throughout the maneuver. Therefore, although the tracking performance has slightly decreased, the goals regarding safety have been successfully met.

E. Robustness Analysis

The DDPG, SIP and shielded DDPG controllers are tested on different initial flight conditions (IFC) and different reference flight path angles so to assess their robustness. An overview is given in Table 4.

Table 4. Overview of robustness analysis results to different initial flight conditions and reference signals. All controllers were trained on the nominal flight conditions.

Simulation	**Initial altitude [m]**	**Initial $V_{\mathbf{TAS}}$ [m/s]**	**nMAE%**	$\mathcal{S}_{\text{risk}_{\mathbf{H}}}$%	$\mathcal{S}_{\text{unsafe}}$%
DDPG controller					
Nominal FC	10000	150	17.1%	2.5%	5.9%
Turbulent conditions and biased sensor noise	10000	150	16.1%	1.4%	2.1%
Sinusoidal reference flight path angle	10000	150	33.6%	15.7%	1.3%
Sawtooth reference flight path angle	10000	150	31.0%	1.6%	5.0%
IFC 1	7000	150	29.3%	56.6%	3.9%
IFC 2	7000	90	40.0%	0.5%	1.2%
IFC 3	2000	150	41.4%	34.4%	64.2%
IFC 4	2000	90	15.7%	1.4%	1.5%
SIP controller					
Nominal FC	10000	150	34.6%	0%	0%
Turbulent conditions and biased sensor noise	10000	150	35.0%	1.5%	0%
IFC 1	7000	150	34.3%	63.6%	0%
IFC 2	7000	90	45.1%	0%	0%
IFC 3	2000	150	35.9%	16.6%	83.2%
IFC 4	2000	90	33.3%	0%	0%
DDPG with shield					
Nominal FC	10000	150	24.0%	1.5%	0%
Turbulent conditions and biased sensor noise	10000	150	26.2%	0.9%	0%
Sinusoidal reference flight path angle	10000	150	37.0%	10.5%	0%
Sawtooth reference flight path angle	10000	150	48.0%	6.0%	0%
IFC 1	7000	150	34.4%	65.9%	0%
IFC 2	7000	90	40.7%	0.1%	0%
IFC 3	2000	150	35.9%	14.2%	85.8%
IFC 4	2000	90	21.9%	0.1%	0%

When looking at the nMAE% among the different IFC, it can be noticed that low altitude and low velocity have the best tracking performance. The lower altitude and therefore high atmospheric density seem beneficial to the controller's performance. IFC 1 also performs well across the board in terms of tracking performance. With a higher velocity comes a quadratic increase in dynamic pressure and consequentially an increase in control effectiveness. From the results presented in Table 4, it is not possible to define a clear relationship between altitude, velocity and tracking effectiveness. The cause can be found in the high non-linearity of the DASMAT model as well as the non-linear nature of a RL controller. Although no specific conclusion can be drawn regarding the relationship between tracking performance and IFC, all controllers achieve a nMAE% below 50% and can therefore be considered robust. The shielded and unshielded DDPG controllers were

additionally tested with different γ_{ref}. Both controllers are able to successfully track the different references with relatively low nMAE%. It should be noted that during the tracking of the sawtooth reference γ, the shield was triggered during the entire simulation. Therefore, the maneuver was effectively carried out by the SIP. From a tracking performance point of view, it is clear that the randomization of γ_{ref} during training has beneficial effect on robustness. In terms of safety, the shield consistently keeps the agent outside of $\mathcal{S}_{\text{unsafe}}$ as much as possible while performing the tracking task. Across the eight scenarios simulated, the shield is able to keep the time spent in $\mathcal{S}_{\text{unsafe}}$ equal to 0% for seven scenarios. The only case where the shield is not able to keep the agent out of the unsafe state space is IFC 3. In this scenario, the low altitude and high speed result in α consistently below the safety limits. As the shield is on 100% of the times and the controller used is effectively the SIP, this fault can be linked to the way the reward function of the SIP is designed. Therefore, the shield can be considered robust to changes in flight conditions and reference signals as seven out of eight of the scenarios are effectively kept away from $\mathcal{S}_{\text{unsafe}}$ at all times.

V. Conclusion

This research presents the development of an offline, model-free Deep Deterministic Policy Gradient (DDPG) flight path angle controller equipped with a shield, a safety enhancing technique, used for the safe control of a Cessna Citation 500. The shield is composed of a Safe Initial Policy (SIP) agent and a Safety Range (M_{SR}) model. The former is a trained DDPG agent with knowledge about state space safety, able to suggest safe actions to the main DDPG agent, while the latter is a rule based model in charge of overruling actions that would lead to unsafe state spaces. The shielded controller is able to successfully achieve a conventional step down approach with low tracking error and overall stable responses while maintaining the agent in a safe state space at all times. The effectiveness of the controller in real-life conditions is also proved by introducing biased sensor noise and atmospheric turbulence. Additionally, it is shown that the controller is robust by testing its performance in various initial flight conditions and reference signals. Therefore, this research shows the potential of shielded DDPG controllers for the development of safe, robust flight controllers.

Several steps can be done in order to improve the completeness of this research. The performance of the SIP is based on the trade-off between safety and tracking performance as defined by its reward function. Although it is necessary for the agent to propose an action relevant to the simulated flight envelope, this can cost in terms of safety. It is recommended to investigate the use of a hierarchical reward function. The M_{SR} overrules actions taken by the main DDPG agent and substitutes them with ones that will not lead to an unsafe state. However, since the M_{SR} evaluates two states, it is possible that an action taken to keep one state within safe limits at time t, it leads the second state to an unsafe state space at time $t + 1$. It is therefore suggested to implement a feedback loop in the M_{SR} to avoid such behaviour. This research contributes to the ongoing efforts in developing learning flight controllers able to guarantee airborne safety. The use of a shield is shown to be an effective safe reinforcement learning approach when paired with a model-free, offline deep reinforcement learning algorithm. By promoting the efforts in this field, safe learning controllers may soon be applied to real-life flight control applications so to reduce airborne LOC-I fatalities.

References

[1] Mazareanu, E., "Number of flights performed by the global airline industry from 2004 to 2022." *International Air Transport Association*, 2022.

[2] IATA, "Loss of control in-flight accident analysis report 2010-2014," *Montreal-Geneva: International Air Transport Association*, 2015.

[3] Steinberg, M. L., "Comparison of intelligent, adaptive, and nonlinear flight control laws," *Journal of Guidance, Control, and Dynamics*, Vol. 24, No. 4, 2001, pp. 693–699.

[4] Dally, K., "Deep Reinforcement Learning for Flight Control: Fault-Tolerant Control for the PH-LAB," *MSc thesis,*

Delft University of Technology., 2021. MSc thesis, Delft University of Technology, 2021.

[5] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J., "First return, then explore," *Nature*, Vol. 590, No. 7847, 2021, pp. 580–586.

[6] Milz, D. M., and Looye, G., "Design and evaluation of advanced intelligent flight controllers," *AIAA Scitech 2020 Forum*, 2020, p. 1846.

[7] Tang, C., and Lai, Y.-C., "Deep Reinforcement Learning Automatic Landing Control of Fixed-Wing Aircraft Using Deep Deterministic Policy Gradient," *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 1–9. doi: 10.1109/ICUAS48674.2020.9213987.

[8] Völker, W., Li, Y., and Van Kampen, E.-J., "Twin-Delayed Deep Deterministic Policy Gradient for altitude control of a flying-wing aircraft with an uncertain aerodynamic model," *AIAA SCITECH 2023 Forum*, 2023, p. 2678.

[9] Tsourdos, A., Permana, I. A. D., Budiarti, D. H., Shin, H.-S., and Lee, C.-H., "Developing flight control policy using deep deterministic policy gradient," *2019 IEEE International Conference on Aerospace Electronics and Remote Sensing Technology (ICARES)*, IEEE, 2019, pp. 1–7.

[10] Bernard, T., Stephane, L., and Boy, G. A., "Autonomous stall recovery dynamics as a prevention tool for general aviation loss of control," *International Conference on Applied Human Factors and Ergonomics*, Springer, 2017.

[11] Sutton, R. S., and Barto, A. G., *Reinforcement learning: An introduction*, MIT press, 1998.

[12] Dally, K., and Van Kampen, E.-J., "Soft Actor-Critic Deep Reinforcement Learning for Fault Tolerant Flight Control," *AIAA SCITECH 2022 Forum*, 2022, p. 2078.

[13] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[14] Garcıa, J., and Fernández, F., "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, Vol. 16, No. 1, 2015, pp. 1437–1480.

[15] Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U., "Safe reinforcement learning via shielding," *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] Zoon, J., "Safe Reinforcement Learning by Shielding for Autonomous Vehicles," *MSc thesis, Delft University of Technology.*, 2021.

[17] Liessner, R., Schmitt, J., Dietermann, A., and Bäker, B., "Hyperparameter Optimization for Deep Reinforcement Learning in Vehicle Energy Management." *ICAART (2)*, 2019, pp. 134–144.

[18] Bjorck, J., Gomes, C. P., and Weinberger, K. Q., "Is High Variance Unavoidable in RL? A Case Study in Continuous Control," *CoRR*, Vol. abs/2110.11222, 2021. URL `https://arxiv.org/abs/2110.11222`.

[19] Fala, N., "An Analysis of Fixed-Wing Stall-Type Accidents in the United States," *Aerospace*, Vol. 9, No. 4, 2022.

[20] van Ingen, J., de Visser, C. C., and Pool, D. M., "Stall Model Identification of a Cessna Citation II from Flight Test Data Using Orthogonal Model Structure Selection," *AIAA Scitech 2021 Forum*, 2021, p. 1725.

[21] Administration, F. A., *Airplane Flying Handbook (FAA-H-8083-3A)*, Skyhorse Publishing Inc., 2011.

[22] Agency, E. U. S., "Equivalent Safety Finding, Enhanced Stall Protection," Vol. ESF-B25.103-01, No. 2, 2021.

[23] Agency, E. U. S., "Equivalent Safety Finding, Normal Load Factor Limiting System," Vol. ESF-B25.143-01, No. 1, 2021.

[24] Errico, A., and Di Vito, V., "A methodology for efficient CDA trajectories automatic generation based on precision 3D curved approach in presence of obstacles," *17th AIAA Aviation Technology, Integration, and Operations Conference*, 2017, p. 4099.

[25] Wang, Z., Luo, W., Gong, Q., Cui, Y., Tao, R., Wang, Q., Liang, Q., and Wang, S., "Attitude Controller Design based on Deep Reinforcement Learning for Low-cost Aircraft," *2020 Chinese Automation Congress (CAC)*, 2020, pp. 463–467. doi: 10.1109/CAC51589.2020.9326889.

[26] Van de Moesdijk, G., "The description of patchy atmospheric turbulence, based on a non-Gaussian simulation technique," *Delft University of Technology, Department of Aeronautical Engineering, Report VTH-192*, 1975.

[27] De Prins, J., "Stochastic Aerospace Systems - Lecture Notes," *Delft University of Technology*, 2010.

[28] Jansen, C., "Non-Gaussian atmospheric turbulence model for flight simulator research," *Journal of Aircraft*, Vol. 19, No. 5, 1982, pp. 374–379.

[29] Grondman, F., Looye, G., Kuchar, R. O., Chu, Q. P., and Van Kampen, E.-J., "Design and flight testing of incremental nonlinear dynamic inversion-based control laws for a passenger aircraft," *2018 AIAA Guidance, Navigation, and Control Conference*, 2018, p. 0385.

Part II

Preliminary Research[1]

[1] The Preliminary Research was assessed as part of the AE4020 Literature Study course.

2

Reinforcement Learning

In this chapter, the key elements necessary for understanding the approaches utilized in this research will be presented. Section 2.1 discusses the building blocks of reinforcement learning. Following, a taxonomy of reinforcement learning algorithms and their suitability for this research will be given in Section 2.2. Finally in Section 2.3, an in-depth analysis of methods combining Deep Q-Learning with Actor-Critic methods will be given. The contents of this chapter will answer *RQ-2.2*.

2.1. Fundamentals

Reinforcement Learning (RL) is a machine learning method based on the interactions between a decision-making agent and the environment. This framework draws inspiration from the way humans approach learning by trial and error and received feedback.

The main cornerstones of RL are the agent and the environment. The learning occurs by letting the agent interact with the environment while trying to obtain the maximum reward for taken actions. Throughout this research, the words *environment* and *agent* will be widely used. Fig. 2.1 shows the links that connect these two concepts which will later be explained.

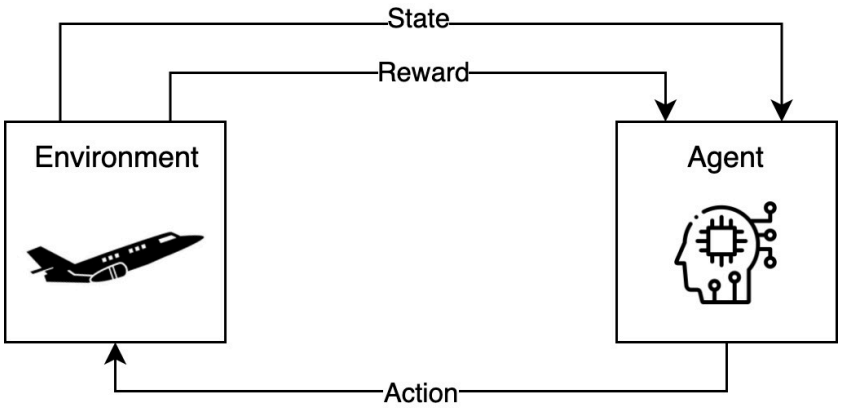


Figure 2.1: Classical reinforcement learning framework. Adapted from Dally (2021).

Since the aim of this research is to implement reinforcement learning methods to flight control, the environment can be considered to the fixed-wing aircraft to be controlled, while the agent is the flight controller to be designed. In the upcoming sections, a brief summary of the main RL concepts will be given. These follow from the theory provided by Sutton and Barto (1998).

2.1.1. Markov Decision Processes

Markov Decision Processes (MDPs) are a fundamental notation that defines most reinforcement learning systems. MDPs describe the sequential interaction in discrete time steps between the learning-agent

and the environment. Fig. 2.1 clearly shows the primary reinforcement learning framework. The environment is the *world* in which the agent operates; at a time step of t, the current condition of the agent is called a state s_t, which results from an action a_t taken by the agent in the environment. The agent also receives feedback from the taken action in the form of a reward r_t.
Given a state s_t and an action a_t, the probability of reaching a new state at time $t+1$ is given by

$$\mathcal{P}\left\{s_{t+1}, \tilde{r}_{t+1} \mid s_t, a_t, \ldots, s_0, a_0\right\} \tag{2.1}$$

where $\tilde{r}_{t+1}$ is the immediate reward achieved by the agent. However, Eq. (2.1) can be further simplified while accounting for the Markov Property, which states that "the future is independent of the past given the present" (Sutton and Barto, 1998). Keeping this in mind, Eq. (2.1) can be modified to arrive at Eq. (2.2).

$$\mathcal{P}\left\{s_{t+1}, \tilde{r}_{t+1} \mid s_t, a_t\right\} \tag{2.2}$$

Therefore, the transition from s_t to s_{t+1} is unaffected by past states: all information regarding the past is embodied by the current state.

2.1.2. Markov Reward Processes

Reinforcement learning is an iterative trial and error process with one of the main components being the reward acquired by the agent. After making an action at time t, the agent earns an immediate reward $\tilde{r}_{t+1}$ at time $t+1$. This concept can be shown via Eq. (2.3)

$$R_s = E\left[\ r_{t+1} \ \middle|\ s_t\ \right] \tag{2.3}$$

where R_s is the Markov Reward Process (MRP). However, the goal of RL is to maximize the accumulated reward over time for complex tasks such as the objective of this research. For this reason, one can define a new parameter named the discounted reward R_t. The latter can be defined by the sum of discounted rewards accumulated throughout learning until the terminal time step $t+N+1$. The mathematical formulation of the discounted reward R_t can be seen in Eq. (2.4)

$$R_t = \tilde{r}_{t+1} + \gamma\tilde{r}_{t+2} + \ldots + \gamma^N\tilde{r}_{t+N+1} = \sum_{N=0}^{\infty}\gamma^N\tilde{r}_{t+N+1} \tag{2.4}$$

where γ is the discount factor and N is the number of steps in one episode. The value of γ varies between zero and one, depending on how much the agent strives for immediate or future rewards respectively.

2.1.3. Policy and Value-functions

A policy π is the strategy followed by the learning-agent to acquire a reward. Formally, it is the probability of an agent choosing an action a_t in a state s_t and it is defined as given in Eq. (2.5).

$$\pi\left(a_t \mid s_t\right) = \mathcal{P}\left\{a_t \mid s_t\right\} \tag{2.5}$$

Once π is chosen, *state-value functions* $V^\pi(s)$ are used to evaluate the policy's effectiveness: they give an indication of whether a taken action performed in a state is advantageous for maximizing the reward. It is clear that value-functions are tightly connected to the concept of π, given that $V^\pi(s)$ is the expected return acquired by the agent for following a policy π when starting at state s_t. $V^\pi(s)$ can be formally expressed in MDPs as shown in Eq. (2.6)

$$V^\pi(s_t) = E_\pi\left\{R_t \mid s_t = s\right\} = E_\pi\left\{\sum_{N=0}^{\infty}\gamma^N r_{t+N+1} \mid s_t = s\right\} \tag{2.6}$$

where E_π is the expected value when following a policy π at time t. Similarly, the expected return when a given action a_t, which follows a policy π is taken in state s_t, the value-function is denoted as $Q^\pi(s)$ and named *action-value-function*. Alike Eq. (2.6), $Q^\pi(s)$ can be described using MDPs notation as follows.

$$Q^\pi(s_t, a_t) = E_\pi\left\{R_t \mid s_t = s, a_t = a\right\} = E_\pi\left\{\sum_{N=0}^{\infty}\gamma^N r_{t+N+1} \mid s_t = s, a_t = a\right\} \tag{2.7}$$

Naturally, the aim of reinforcement learning is to maximize the expected return: this can be obtained by following an optimal policy π^*. Once adopted, an optimal policy can reshape Eq. (2.6) and Eq. (2.7) into the following relations:

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t) \tag{2.8}$$

$$Q^*(s_t, a_t) = \max_{\pi} Q^{\pi}(s_t, a_t) \tag{2.9}$$

where V^* and Q^* are respectively the optimal state and action-value-functions .

2.2. Reinforcement Learning Taxonomy

Reinforcement learning may be performed in a variety of ways, using different algorithms each with varying properties. Zhang et al. (2019) describe in detail the various algorithms that belong to the Reinforcement Learning family while highlighting their differences and similarities. In a similar manner, this paper includes a classification of different RL methods in order to decide upon an algorithm to apply to this flight control problem.

Fig. 2.2 gives an overview of a few RL algorithms and how they are classified according to Zhang et al. (2019). In the figure, characterizing features of these RL methods are indicated by the blocks while the specific algorithms are given by the rounded rectangles. In the next sections, each of the branches of this tree will be analyzed to arrive at the algorithm that will be further investigated in the remainder of the research. It should be noted that the choice of a final algorithm will be strongly influenced by its ability to perform in a continuous state-action space. This is due to the control task to be performed, as modern flight control is usually implemented in continuous spaces (Pollack, 2019).

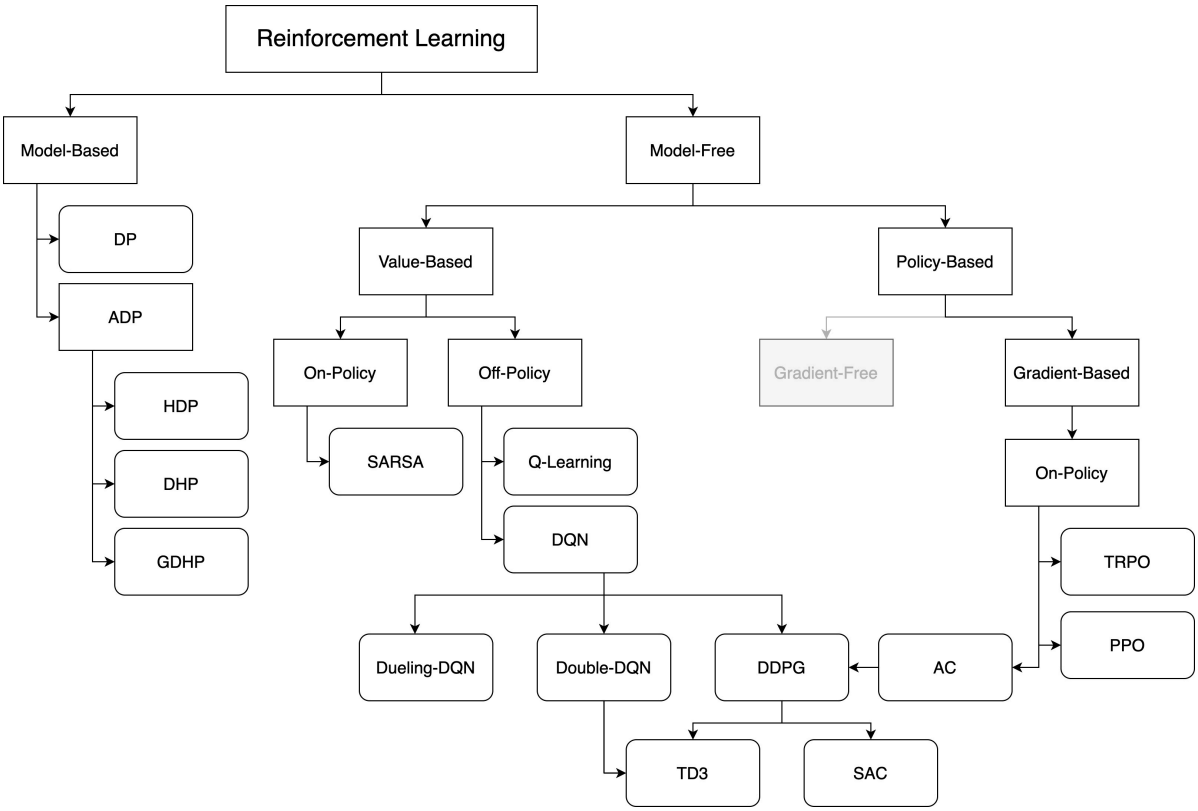


Figure 2.2: Taxonomy of Reinforcement Learning algorithms. Adapted from Zhang et al. (2019).

2.2.1. Online and Offline Learning

Before diving in into the taxonomy shown in Fig. 2.2, a distinction between offline and online learning has to be made. This is due to the fact that the way learning is carried is a main factor in the choice of

an algorithm class e.g. due to some method's sample efficiency.

The learning process can be carried either online or offline. If the learning is performed online, the data is acquired at the same time as the agent is performing its actions (Hofer and Gimbert, 2016). In the context of this research, online learning occurs *during flight*, as the controller is being learned. Online learning can be quite computationally expensive, as high volume of data might need to be processed and adjusted during trials.

Offline learning is performed *on the ground*. The agent has no time constraints to obey and has the liberty to explore as close as possible to the safe boundary layer of the action space (Levine et al., 2020). While a control learning strategy performed online is named *adaptive*, offline learning is labeled *robust*. Online learning is undoubtedly more relevant within the concept of safety as safety should be achieved in real time in case of encountering risks while exploring the environment. The agent must be able to remain within the safe state space while being able to maximize the accumulated reward. Offline learning on the other hand, does not imply risky actions which may be counterproductive in this research. It is therefore decided to develop a two-stage strategy in which the learning can first be conducted offline to gather knowledge about the environment in a safe way after which the agent can be better equipped to conduct safe learning in the online stage.

2.2.2. Model Dependency

In Reinforcement Learning, a *model* is the collection of knowledge gathered about the environment, i.e. a function which gives a prediction about the state transition and rewards of the agent. As explained in Section 2.1, the agent undergoes a process of trial and error by taking actions in the environment and awaiting feedback. This learning process can be conducted in two different ways which correspond to the main branches of the tree shown in Fig. 2.2.

Model-Based Algorithms

One way to carry out learning is by following the methodology of model-based RL algorithms. Knowledge of the probability of transition from s_t to s_{t+1} due to action a_t, the reward function, the action space $\mathcal{A}$ and state spaces $\mathcal{S}$ are assumed to be known in model-based RL algorithms (Huys et al., 2014). In this analysis, Dynamic Programming and Approximate Dynamic Programming are highlighted; However, many more algorithms can be placed under the model-based umbrella as reported in Zhang et al. (2019).

Dynamic Programming (DP) is at the roots of many RL algorithms as it provides a basic framework for a plethora of applications such as planning problems, game theory, economics and more. Although it represents a stepping stone of RL, it is not often implemented as the state and action spaces are often discrete and a full model of the environment is required. Since this is not always possible, especially in stochastic control tasks, DP algorithms have been further expanded to be more easily used in complex systems.

One modification that can be done to make DP more adaptable is to use function approximators to combat the curse of dimensionality caused by discrete spaces. This solving strategy is called Approximate Dynamic Programming (ADP) and is suitable for large and complex problems which are often stochastic (Powell, 2009). It is worth noting that ADP can be considered a model-*dependent* strategy rather than a model-based method. By using current state measurements rather than predictions for updating the policy, ADP detaches itself from other methods that strongly rely on a model (Heyer et al., 2020). This characteristic is particularly intriguing when an inexact model might be implemented: the agent will not be fully subjected to potentially incorrect information during the process of learning.

Many versions of Approximate Dynamic Programming have been developed in the years. Dally (2021) provide a classification in which three main structures of ADP are defined, namely Heuristic Dynamic Programming (HDP) (Tang and Lai, 2020), Dual Heuristic Programming (DHP) (Ni et al., 2015) and Global Dual Heuristic Programming (GDHP) (Sun and van Kampen, 2020).

Model-Free Algorithms

The second way learning can be carried out is shown in the right branch of Fig. 2.2. In model-free learning, the dynamics of the environment are not modeled and the agent tries to optimize learning by

looking directly for an optimal policy. One advantage of this strategy is that the agent will not inherit any bias introduced by a low-fidelity model. This is particularly attractive in flight control as high-fidelity models are computationally expensive to test.

On the opposite side of the model-dependency spectrum from DP, Temporal Difference (TD) learning is a class of model-free methods that uses values from past iterations to estimate current values. This approach is known as *bootstrapping* and it is well suited for continuous control tasks. The main idea of TD is to compute the error between the current value-function estimate V_t, the discounted value-function V_{t+1} and the reward for transitioning form s_t to s_{t+1}. The TD error approach has been extensively used in both on-policy and off-policy algorithms and is a crucial component in the operation of two central algorithms in RL: SARSA and Q-Learning. Both of these will be discussed further on in this report.

With the agent not relying on learning the environment, it is crucial to define *how* decisions regarding what actions to make are taken. This is a central theme in RL, for both model-free and model-based algorithms, which is often referred to as the *Exploration vs. Exploitation trade-off* (Louis and Yu, 2019). Exploiting the knowledge gained in past actions is one of the options in this trade-off. Performing repeatedly the same actions that follow the optimal policy π^* result in a high immediate reward. This is obviously attractive, however it does not lead to convenient results in the long term. Another option is to let the agent explore the environment in a less or more *greedy* manner. The greediness of the agent is defined as the desire of the agent of taking actions that will result in the highest result with the given knowledge. Greediness is expressed by the parameter ϵ which has values between zero and one and defines how much the agent wants to maximize the long or short term reward. This exploration can result into performing sub-optimal tasks which may lead to a negative immediate reward. This is to be expected, as the agent takes random decisions different from past actions. However, by allowing the agent to explore vast sections of the environment, learning is more prolific and a higher overall reward could be achieved.

Both these strategies have their strong and weak points, therefore the outcome of the trade-off could lead to allowing the agent to balance both exploitative and explorative actions. A popular strategy is to let the agent explore with frequency ϵ while exploiting gained knowledge for the remainder $(1-\epsilon)$ of the learning.

The main focus of this research is to improve the safety of a flight control system that will later be defined. Working with a model-free algorithm was deemed to be the best fit as model-base strategy would give the agent an inherently safe environment to learn. Therefore, for the remainder of this paper, model-based algorithms will not be discussed.

2.2.3. Value-Based and Policy-Based

Model-free algorithms can optimize their policy in two different ways, namely with a Value-based or a Policy-based approach. As it can be seen in Fig. 2.2, there exists an abundance of methods for both branches of model-free algorithms.

Policy-Based Algorithms

If the policy π is straightforwardly optimized, the method is said to be policy-based. In these cases, the policy is iteratively updated so that the expected return $J(\theta)$ can be maximized as follows in Eq. (2.10)

$$J(\theta) = E\left[\sum_{t=0}^{T} \gamma^t r_t\right] \tag{2.10}$$

where θ is the policy parameterized with a vector $\boldsymbol{\theta}$ and T is the horizon of sampled trajectories. Policy-based algorithms can be further classified into Gradient-Based and Gradient-Free. Although gradient-free methods are shown to be computationally efficient in arriving at an optimal solution, they use discrete Monte-Carlo notations rather than continuous MDPs (Wiering and Van Otterlo, 2012). For this reason, they will not be further investigated in this research.

Gradient-based optimization improves θ by using an estimate for the gradients on the expected return acquired from sample trajectories. θ is updated via gradient descent/ascent in the direction of $J(\theta)$ in

order to produce the highest return. The update follows the rule described below

$$\Delta\theta = \alpha\nabla_\theta J(\theta) = E_\theta\left[\sum_{t=0}^{T}\alpha\nabla_\theta\left(\log\theta\left(a_t \mid s_t\right)\right)Q^{\pi_\theta}\left(s_t, a_t\right)\right] \tag{2.11}$$

where α is the learning rate. Eq. (2.11) is the Policy Gradient Theorem proposed by Sutton and Barto (1998) for stochastic policies and later adapted by Silver et al. (2014) to include deterministic policies. Eq. (2.11) lays the foundation for many policy-based algorithms such as REINFORCE (Williams, 1992) which can be considered the baseline for these algorithms. REINFORCE is a policy-gradient method that updates θ based on an estimated return using Monte-Carlo algorithms utilizing episode samples. This algorithm depends on the full trajectory therefore it is quite impractical for many stochastic applications. However, REINFORCE gives insight on an issue affecting many policy-based algorithms: their high variance does not allow for a smooth convergence to a deterministic policy. In order to combat this, a variation of REINFORCE has been proposed where the state-value $V^\pi(s_t)$ is subtracted by the action-value-function $Q^\pi(s_t, a_t)$ in the update rule. This allows the variance to be reduced while keeping the process unbiased (Weng, 2018).

Value-Based Algorithms
As the name may suggest, value-based methods focus on optimizing the action-value-function $Q^\pi(s_t, a_t)$. Section 2.1.3 touched upon the definition of action-value-function as shown in Eq. (2.7) and its optimized version as given by Eq. (2.9). This approach gives a straightforward definition of the optimal policy π^* as $\pi^* \approx \arg\max_\pi Q^\pi$.

Algorithms which adopt this type of method have been readily used in simple optimization tasks as they usually have high sample efficiency, small variance (in contrast with policy-based methods) and are not prone to getting stuck into local optima. Discrete methods such as SARSA (Rummery and Niranjan, 1994) and Q-Learning (Watkins and Dayan, 1992) have been proven to be extremely popular throughout the years (Sutton and Barto, 1998). However, their simplicity makes them less appropriate for use in continuous state and action spaces. To remedy this flaw, different function approximators can be implemented which allow these discrete value-based methods to be applied in a variety of optimization problems.

The simplest of function approximators are linear methods. These are well understood due to the extensive years of research and their striking simplicity as they are indeed, made of linear functions (Zhang et al., 2019). A mix of weights and state-dependent real-valued vectors are combined to develop these approximators. The vectors can in turn have different configurations; the most common are polynomials, Fourier basis, Radial Basis Function (RBF), tile and coarse coding (Babuška and Kober, 2010). Although their simplicity allows for fast convergence, most methods rely on information on the environment provided by the user. As mentioned previously in Section 2.2.2, the less information about the environment to be provided the better. Therefore, linear methods will not be considered for the remainder of this research.

On the other hand, non-linear methods have been mainly generated in the form of Artificial Neural Networks (ANNs), computing systems developed to imitate biological neural networks. ANNs have been implemented in optimization techniques for decades and have been a central theme in Reinforcement Learning approaches due to their well-guaranteed convergence and accuracy (Wiering and Van Otterlo, 2012). ANNs are constituted of different items. Fig. 2.3 gives an example of a standard NN architecture. Neurons, the gray circles in the figure, are the building blocks of ANNs. They translate an input to an output, which is then sent to a number of other neurons. Neural networks are organized in layers. In Fig. 2.3, the network is made of an input layer with four neurons, two hidden layers with eight and six neurons respectively and finally an output layer composed of two neurons. Connections allow the information to flow in and out clusters of neurons. Each link representing a connection is given a weight, which provides information on the influence of the data to be transmitted. In Fig. 2.3, the weight of the links is represented by the different colors of the arrows. Once the weighted information arrives at the hidden layer, an adder adds up all the inputs together with a bias introduced by the presence of hidden layers. This action is known as linear combination. Finally, an activation function regulates

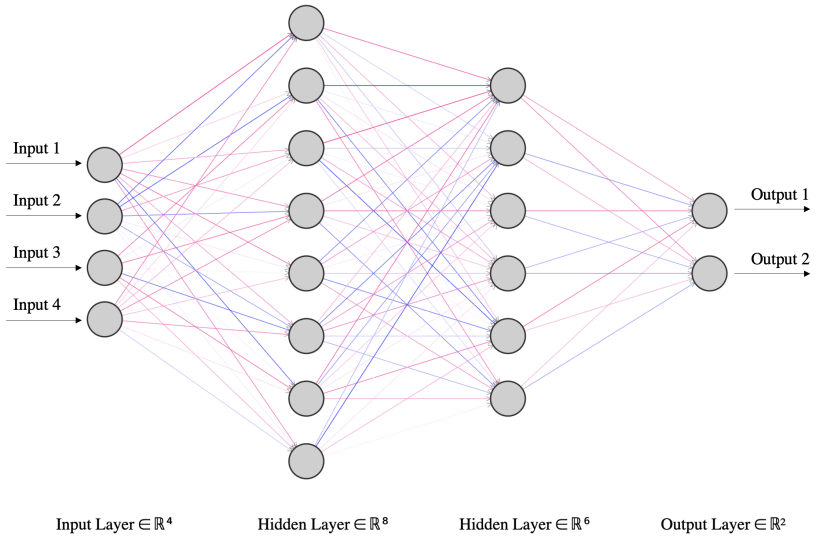


Figure 2.3: Example of a neural networks architecture with four input neurons, two hidden layers with eight and six neutrons respectively and a two neuron output layer. The different colors of the arrows between layers show the weight of the links.

the magnitude of the neuron's output so to keep its value typically between -1 and 1. Many types of activation functions can be implemented, with the most common being Rectified Linear Unit (ReLU) and the Hyperbolic Tangent (Dongare et al., 2012). ANNs are widely used in modern RL techniques, most commonly in Deep Reinforcement Learning (DRL).

Actor-Critic Algorithms

A class of algorithms that are found at the middle between value-based and policy-based is Actor-Critic (AC). This algorithm has a dual structure: the *actor* represents the policy structure that selects the actions to be taken, the *critic* gives the estimate of the value-function. A schematization of this process is shown in Fig. 2.4.

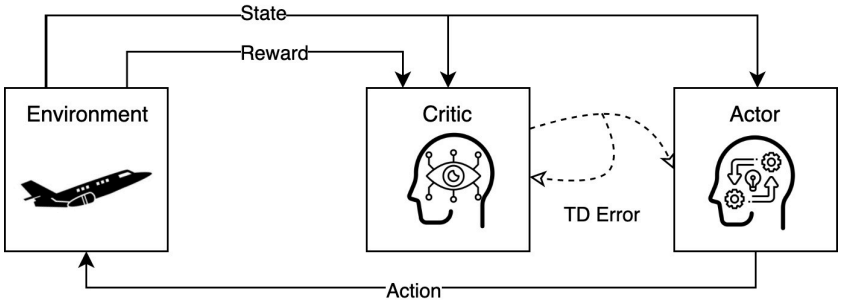


Figure 2.4: Actor-Critic framework. Adapted from Dally (2021).

As can be seen in the figure, given the new state and reward after an action is performed on the environment, the critic provides the actor with the TD error, as introduced in Section 2.2.2. However, the error needs to be monitored by the critic as it is trying to let the actor optimize the same policy that is being learned. The pseudocode for an Actor-Critic approach is shown in Algorithm 1.
For each iteration, the TD error is computed after the state, actions and reward for one episode has been collected. The TD error is used to update the policy π so that the actor can choose action a_t that maximizes the future reward. Once the error has been approximated, the critic $J_{V_\psi^{\pi_\theta}}(\psi)$ and the actor $J(\theta)$ are updated. Following, the policy and function parameters θ and ψ are updated. It should be noted that θ and ψ are used interchangeably to denote π_θ and $J_{V_\psi^{\pi_\theta}}(\psi)$ due to introducing neural networks as function approximators (Zhang et al., 2019).

Actor-critic algorithms have been extensively used in the field of Reinforcement Learning due to

Algorithm 1 Actor-Critic (Konda and Tsitsiklis, 2000).

Hyperparameters: step size η_θ and η_ψ, reward discount factor γ
Input: initial policy parameters θ_0, initial value-function parameters ψ_0
Initialize $\theta = \theta_0$ and $\psi = \psi_0$
for each step t **do**
    Run policy π_θ for one step, collection $\{s_t, a_t, r_t, s_{t+1}\}$
    Compute the TD error correction $\hat{A}_t = R_t + \gamma V_\psi^{\pi_\theta}(s_{t+1}) - V_\psi^{\pi_\theta}(s_t)$
    Update the critic by $J_{V_\psi^{\pi_\theta}}(\psi) = \sum_t \hat{A}_t^2$
    Update actor by $J(\theta) = \sum_t \log \pi_\theta(a_t \mid s_t)\, \hat{A}_t$
    Update policy and value-function parameters $\theta = \theta + \eta_\theta \nabla J(\theta)$ and $\psi = \psi + \eta_\psi \nabla J_{V_\psi^{\pi_\theta}}(\psi)$
end for
Return θ, ψ

their ability to combine the best between value and policy-based algorithms. By exploiting the TD error that can be retrieved after every step, AC is a far more sample efficient method compared to the non-combined classes of algorithms. With respect to pure policy-based methods, AC also decreases the variance of the estimate gradient due to the implementation of TD errors in the critic structure. Finally, AC methods can be used on continuous action spaces which makes them particularly attractive for this field of research. (Wiering and Van Otterlo, 2012)

2.2.4. Off-Policy and On-Policy

The last aspect of this classification is off-policy and on-policy algorithms. By now, the concept of a policy should be well understood as it is an intrinsic part of Reinforcement Learning and MDPs. As it can be seen from Fig. 2.2, both branches of model-free algorithms can be classified into on- and off-policy.

On-policy algorithms

On-policy approaches seek to assess or improve the policy that the agent follows while performing actions. This means that the agent is trying to improve a strategy whilst executing actions according to the latter.

The most well known on-policy method is **SARSA**, or State-Action-Reward-State-Action (Rummery and Niranjan, 1994). Although SARSA works in discrete state and action space and is therefore not relevant for this research, its understanding is crucial as it is the stepping stone for most on-policy algorithms. The working principle of SARSA is easily understandable due to its name: for each state s_t, an action is taken and a reward is awarded to the agent. The state is then transitioned to s_{t+1} in order to take a new action a_{t+1} while continually estimating Q^π and changing π to increase the value of Q^π. This process is shown by the update rule implemented in SARSA, given below in line 9 of Algorithm 2.

Algorithm 2 SARSA (Rummery and Niranjan, 1994).

1: Initialise $Q(s_t, a_t), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily
2: **for** each episode **do**
3: Initialize s_0
4: Select a_0 using policy that is based on Q
5: **for** each step in the current episode **do**
6: Select a_t from s_t using policy that is based on Q
7: r_{t+1}, $s_{t+1} \leftarrow$ env(s_t, a_t)
8: Select a_{t+1} from s_{t+1} using policy that is based on Q
9: $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$
10: **end for**
11: **end for**

Depending on the used policy, e.g. ϵ-greedy, the convergence of SARSA is more or less dependent on the action-value Q^π. The greedier the actions, the more SARSA is equivalent to Q-learning, an

off-policy algorithm shown in Algorithm 4 (Rummery and Niranjan, 1994).

In contrast with SARSA, **Trust Region Policy Optimization (TRPO)** is an algorithm that can be used in continuous action space. TRPO is an on-policy, gradient-based algorithm proposed by Schulman et al. (2015). As mentioned in Section 2.2.3, gradient-based methods use gradient descent/ascent in the direction of the expected rewards $J(\theta)$ to optimize the parameterized policy θ. However, the convergence of these algorithms is highly dictated by the step size at which the progression along the trajectory is conducted. If the step size is too large, the curvature might be ignored and the solution could sustain a performance collapse and hence diverge rather than converge. In an opposite scenario, where the step size would be too small, the learning could be too conservative and would not allow for any considerable progress.

TRPO was indeed developed in order to counteract these issues brought up by the descent/ascent step size. As usual, the goal of this algorithm is to find an updated policy θ' that improves the current policy θ. This can be done mathematically by Eq. (2.12)

$$J\left(\theta'\right) = J(\theta) + E_{\tau\sim\pi'_\theta}\left[\sum_{t=0}^{\infty}\gamma^t A^\theta\left(s_t, a_t\right)\right] \tag{2.12}$$

where $J(\theta)$ is given in Eq. (2.10) and τ is the trajectory produced by θ'. However, working with expectations can be quite tricky. Schulman et al. (2015) proposes to optimize the approximation of the expectation, defined as $\mathcal{L}_\theta(\theta')$. In order for this approximation to be valid, its error needs to be bounded.

$$\begin{aligned} &\max_{\theta'} \mathcal{L}_\theta\left(\theta'\right) \\ &\text{s.t. } E_{s\sim\rho_\theta}\left[D_{KL}\left(\theta\|\theta'\right)\right] \leq \delta. \end{aligned} \tag{2.13}$$

This can be done via a Kullback–Leibler (KL) divergence constraint bounded by constant δ. If this constraint is satisfied, it is reasonable to approximate the expectation and apply TRPO to optimize $\mathcal{L}_\theta(\theta')$. What still remains to be done is to actually calculate the gradient $\mathcal{L}_\theta(\theta')$. This can be done by the relation proposed by Schulman et al. (2015) shown in Eq. (2.14).

$$g = \nabla_\theta L_\theta\left(\theta'\right)|_\theta = E_{\tau\sim\theta}\left[\sum_{t=0}^{\infty}\gamma^t \frac{\nabla_\theta \theta'\left(a_t \mid s_t\right)}{\theta\left(a_t \mid s_t\right)} A^\theta\left(s_t, a_t\right)\right]\Bigg|_\theta \tag{2.14}$$

where $A^\theta = Q^\theta(s_t, a_t) - V^\theta(s_t)$ is the advantage function, already implemented in the REINFORCE algorithm described in Section 2.2.3.

Clearly, this algorithm is quite complex in its implementation: other than being computationally expensive due to the many steps required to estimate the conjugate gradient, its complexity in formulation makes it a less attractive algorithm compared with other approaches (Ha et al., 2018). For this reason, a simplified version was later developed by Schulman et al. (2017). **Proximal Policy Optimization (PPO)** aims to simplify TRPO by exploiting the similarity between θ and θ'. The main difference is in changing the TRPO hard constraint shown in Eq. (2.13) into a *regularized constraint* as shown below:

$$\max_{\theta'} \mathcal{L}_\theta\left(\theta'\right) - \lambda E_{s\sim\rho_\theta}\left[D_{KL}\left(\theta\|\theta'\right)\right] \tag{2.15}$$

where λ is the constraint's regularization parameter. For each constant δ set in Eq. (2.13), there exists a λ that gives in the same optimized result as following a TRPO approach. Here, λ is directly related to θ; the regularization parameter can be adjusted during training to satisfy the KL-divergence constraint. This version of PPO is named **PPO-Penalty**, as it penalizes KL-divergence via the coefficient λ. Another version proposed as well by Schulman et al. (2017) is called **PPO-Clip**. Its main difference from PPO-Penalty is that it does not include a $D_{KL}(\theta\|\theta')$ term in its objective function as well as not including any constraints. Rather, customized clipping in the objective function is used to reduce reasons for the new policy to deviate from the old policy. For more information as well as the pseudocodes for these two approaches, please refer to the original paper by Schulman et al. (2017).

Algorithm 3 TRPO. Adapted from (Schulman et al., 2015).

1: **Hyperparameters**: KL-divergence limit δ, backtracking coefficient α, maximum number of backtracking steps K
2: **Input**: empty replay buffer D_k, initial policy parameters θ_0, initial value-function parameters ϕ_0
3: **for** each episode **do**
4: Collect set of trajectories $D_k = \tau$ by running policy $\pi_k = \pi(\theta_k)$ in the environment
5: Compute rewards-to-go $\hat{r}_t$
6: Compute advantage estimates, A^θ (using any method of advantage estimation) based on the current value-function V_{ϕ_k}
7: Estimate policy gradient g as given by Eq. (2.14)
8: Use the conjugate gradient algorithm to compute

$$\hat{x}_k \approx \hat{H}_k^{-1} g_k$$

9: where $\hat{H}_k$ is the Hessian of the sample average KL-divergence
10: Update the policy by backtracking line search while satisfying the sample KL-divergence constraint
11: Fit value-function by regression on mean-squared Bellman error (MSBE) by

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|D_k|\,T} \sum_{\tau \in D_k} \sum_{t=0}^{T} \left(V_\phi\left(s_t\right) - \hat{r}_t\right)^2$$

12: typically via some gradient descent algorithm
13: **end for**

PPO is a straightforward policy-based, gradient-based method which has been implemented in both discrete and continuous action spaces. However, its online use still needs to be explored. For this reason, it is not deemed feasible to be used in this research as it may not be applicable in the two-stage learning.

Off-policy algorithms

Differently from the algorithms that were just discussed, off-policy methods seek to improve a different policy from the one used to make decisions. A classical off-policy algorithm is **Q-Learning**. It can be seen that the only real difference between Algorithm 2 and Algorithm 4 is line 9, specifically in the second to last term. This is because Q-Learning uses the estimate of the *optimal* future value rather than simply the possible reward received at time $t + 1$.

Algorithm 4 Q-Learning (Watkins and Dayan, 1992).

1: Initialise $Q(s_t, a_t), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$
2: **for** each episode **do**
3: Initialize s_0
4: Select a_0 using policy that is based on Q
5: **for** each step in the current episode **do**
6: Select a_t from s_t using policy that is based on Q
7: r_{t+1}, $s_{t+1} \leftarrow$ env(s_t, a_t)
8: Select a_{t+1} from s_{t+1} using policy that is based on Q
9: $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r(s,a) + \gamma \max_a Q\left(s_{t+1}, a\right) - Q(s,a)\right]$
10: **end for**
11: **end for**

Similarly to SARSA, Q-Learning is applicable only to discrete state and action spaces. It is therefore not viable for this research. However, recent studies implemented neural networks instead of the tabular method used in Q-Learning's as function approximators. The algorithms can then be used in continuous state spaces. This approach lead to a novel type of algorithms named **Deep Q-Network (DQN)** developed by Mnih et al. (2015). The general idea is shown by Fig. 2.5: in DQN, the tabular

method used to approximate the Q-function is replaced by an ANN allowing the system to work with continuous state spaces. Therefore, instead of having one Q-value per iteration, N Q-functions are continuously approximated at each state s_t. DQN is based on modifying Q-Learning by applying two separate ideas, **replay buffer** and **target network**.

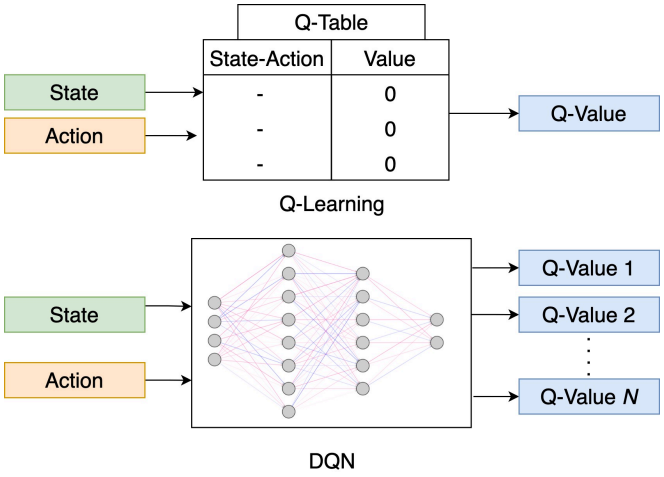


Figure 2.5: DQN framework.

Replay buffer takes inspiration from the natural mechanism of experience replay. At each time step, after a ϵ-greedy action is taken, a sample of the agent's experiences, namely (s_t, a_t, r_t, s_{t+1}) is stored into the replay buffer. The Q-learning update is then applied to find the Q-value-functions. By storing mini-batches of the agent's experience, the Q-function can be learned more easily, which alleviates the computational effort and allows for better data efficiency.

The second concept implemented in DQN is target networks. A separate Q-network is introduced to allow the original Q-network to look up from this secondary target network in order to learn a policy efficiently. This can be seen in line 9 of Algorithm 5. After every C steps, the experience from the original network are copied into the target network. Since the Q-learning target delay is generated with old parameters, the divergence of the algorithm is reduced and the learning occurs in a more stable manner. Line 10 shows the loss function, here given by the mean squared error of the target Q-value minus the predicted Q-value. The success of DQN leads to the generation of different algorithms which lay their foundation in a similar matter. As it can be seen from Fig. 2.2, **Double-DQN** (Van Hasselt et al., 2016) is an offspring of the original DQN developed by Mnih et al. (2015)

When developing Double-DQN the main focus was put into dealing with the problem of overestimation. The Q-learning function shown in line 9 of Algorithm 5 includes a *max* operator: Q is usually noisy and since the expectation of maximum noisy parameters is never less than the maximum expectation of noises, the maximization operator makes it so that the upcoming Q values are often overestimated (Zhang et al., 2019). The solution found by Van Hasselt et al. (2016) is to untie the noises in selection and assessment procedures by employing two distinct networks in these two phases. The results of Double-DQN with respect to DQN are analysed by Wang et al. (2016). When comparing the two algorithms, Double-DQN outperforms DQN in 70.2% (40 out of 57) of Atari games. Although this is a different application than flight control, comparing performance in Atari games gives a baseline about the effectiveness of these two algorithms.

A second mutation of DQN was developed by Wang et al. (2016) and called **Dueling-DQN**. It exploits the notion according to which some actions are not relevant towards learning and therefore should be discarded to improve the learning efficiency. Wang et al. (2016) propose to split the Q-function to differentiate between the state value-function $Q^\pi(s_t)$ and action advantages $A^\pi(s_t, a_t)$. Without having to understand the effect of each action for each state, this separation allows the agent to learn which states are valuable in an efficient way. Wang et al. (2016) compares Dueling-DQN with the classical Deep Q-Network and their performance in Atari games. It was found that Dueling-DQN outperforms

DQN in 25 out of 30 games.

Previously in this paper, it was decided to focus on a dual online/offline approach which could result in a two-stage learning process. If this strategy is pursued, off-policy algorithms would be the best suited methods. This way the agent would be allowed to explore the environment closer to the safe boundary region during the offline phase while finding a safe optimal policy in the online phase. For this reason, on-policy algorithms will not be further analyzed in this research.

Algorithm 5 DQN (Mnih et al., 2015).

Hyperparameters: replay buffer capacity N, reward discount factor γ, delayed steps C for target action-value-function update, ϵ-greedy factor
for each episode **do**
  Initialize environment and get observation s_0
  for each time step t **do**
    Perform ϵ-greedy action selection:

$$a_t = \begin{cases} \underset{a}{\text{random action}}, & \text{with probability } \epsilon \\ \underset{\phi}{\text{argmax}} Q_\phi(s_t, a), & \text{otherwise} \end{cases}$$

    Execute action a_t and observe s_{t+1} and reward r_t
    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer
    Sample minibatch from replay buffer $\{(s_k, a_k, r_k, s_{k+1})\}_{k=1}^{N}$
    Calculate targets y_j :

$$y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} Q_{\phi'}(s_{j+1}, a'), & \text{otherwises} \end{cases}$$

    Calculate the loss:

$$L = \frac{1}{N} \sum_{k=1}^{N} \left(r_k + \gamma \max_{a} Q_{\phi'}(s_{k+1}, a) - Q_\phi(s_k, a_k) \right)^2.$$

    Synchronize the target $\hat{Q}$ every C steps
    Until s is terminal
  end for
end for

2.3. Deep Q-Networks and Actor Critic

One flaw of DQN is its inability to work with continuous action spaces which in modern control tasks is crucial. In the past years, combining AC methods with DQN has resulted in many popular algorithms that allow operation in both continuous state and action spaces without compromising sample efficiency and accuracy. In the upcoming sections, some of these algorithms combining DQN with AC will be discussed.

2.3.1. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) was developed by Lillicrap et al. (2015) as a bridge joining actor-critic algorithms with DQN. By combining AC with DQN, the algorithm can be considered off-policy which gives it a better sample efficiency of its on-policy counterpart. The high sample efficiency, the possibility of working in continuous spaces and their model-free approach make DDPG algorithms particularly interesting for this research. Similarly to AC, DDPG establishes a critic (by means of Q-function) updated in a similar manner as in DQN with TD updates and an actor (through a policy function) updated with a policy gradient approach.

Algorithm 6 DDPG (Lillicrap et al., 2015). Adapted from Dally (2021).

1: Initialize randomly-chosen weights θ and k for policy π_θ and critic $Q_{\bar{k}}$ networks, respectively
2: Initialize weights $\overline{\theta} \leftarrow \theta$ and $\overline{k} \leftarrow k$ for policy $\pi_{\overline{\theta}}$ and critic $Q_{\overline{k}}$ target networks, respectively
3: Initialize initial state s_0, random process N and smoothing factor τ
4: Initialize memory buffer $\mathcal{D}$
5: Sample initial action $a_0 \sim \pi_\theta\left(a_0 \mid s_0\right)$
6: **for** each step t **do**
7: Execute action $a_t = \pi_\theta\left(a_t \mid s_t\right) + N$
8: Sample r_t and $s_{t+1} \sim \mathcal{P}\left\{s_{t+1} \mid s_t, a_t\right\}$
9: Store transition $\left(s_t, a_t, r_t, s_{t+1}\right)$ in $\mathcal{D}$
10: Sample a random mini-batch of n transitions $\left(s_i, s_i, \tilde{r}_i, s_{i+1}\right)$ from $\mathcal{D}$
11: Compute targets: $y_i = r_i + \gamma Q_{\overline{k}}\left(s_i, \pi_\theta\left(s_i\right)\right)$
12: Update critic with one-step gradient descent by minimizing the loss:

$$L = \frac{1}{n} \sum_{i=0}^{n} \left(y_i - Q_k\left(s_i, a_i\right)\right)^2$$

13: Update the actor policy using the sampled policy gradient:

$$\nabla_\theta J \approx \frac{1}{n} \sum_{i=0}^{n} \nabla_a Q_k(s, a)\Big|_{s=s_i, a=a_i} \nabla_\theta \pi_\theta(s)\Big|_{s=s_i}$$

14: Update the target networks weights:

$$\overline{\theta} \leftarrow (1-\tau)\theta + \tau\overline{\theta}$$

$$\overline{k} \leftarrow (1-\tau)k + \tau\overline{k}$$

15: **end for**

As it can be seen from line 1 of Algorithm 6, the policy is denominated as π_θ while the critic is denominated as $Q_{\bar{k}}$. In DDPG, the policy is deterministic. This is due to the fact that the policy is integrated over only the action space rather than both state and action spaces as commonly done within stochastic policies. This is done in order to alleviate computational strain which is a frequent problem in continuous RL methods. Finding an optimal policy is assured by allowing the agent to learn the deterministic policy in an off-policy fashion. If the agent were to learn on-policy, the learning would be way too conservative at the beginning to make a real progress in finding optimal actions. Instead, the exploration of the agent is improved by adding mean-zero Gaussian noise N to its actions while learning (Sanghi, 2021).

From Algorithm 6, many similarities can be seen when comparing it to Algorithm 5. Generally, DDPG carries the same idea of DQN of minimizing a mean-squared Bellman error via the loss function L. Minimizing the error is aided by the concept of reply buffer, which is also introduced in DDPG. Reply buffer, i.e. allowing the agent to learn in batches of experiences gathered by the agent after learning as shown in **line 9-10**, grants the algorithm improved stability while learning and lowered error due to knowledge from past experiences.

Another notion carried from DQN is that of target networks. Although DQN's target networks were implemented solely for $Q_{\phi'}$, DDPG sports target networks for both the actor and the critic as seen in **line 2**. The target is found as from **line 11**, where the algorithm is trying to make the main Q-network as close as possible to the target. Finally, learning occurs by MSBE loss with stochastic gradient descent as shown in **line 12** while the policy is updated via gradient ascent (with respect to policy parameters only) as given by **line 13** of Algorithm 6. In the end, the target networks parameters θ and k are updated by exponential smoothing as shown in **line 14**.

2.3.2. Twin Delayed DDPG

Although DDPG improves the learning performance with respect to DQN, it also shares some of its flaws. A modified version of DDPG, Twin Delayed DDPG (TD3) was developed by Fujimoto et al. (2018) to account for some of the disadvantages of DDPG. As mentioned in Section 2.2.4, DQN suffers the curse of overestimation resulting from the *max* operator in the value-function. This problem is present in DDPG as $Q(s, a)$ is updated in a similar fashion. To improve upon this problem, an approach similar to Double-DQN is implemented where two Q-value-functions are learnt and the minimum between the two is used for the policy update. Overestimation is then avoided. It could be argued that this approach would introduce an underestimation; however, this case is preferred to an overestimation (Zhang et al., 2019). This is due to the fact that underestimated Q-values do not propagate from update to update, resulting is a less severe bias than overestimated values (Fujimoto et al., 2018).

Another method that finds its roots at DDPG is using target networks. It was already discussed in the previous section how target networks are a solid means to achieve a more stable learning process by reducing the error throughout the policy's updates. To exploit this statement, TD3 updates the policy Q-network with reduced frequency with respect to the main Q-network. Therefore, the variance of the update is reduced due to the fewer policy updates. This results in an overall better policy (Fujimoto et al., 2018).

As mentioned, TD3 is an extension of DDPG to improve the overall quality of the algorithm. One issue that the developer of this method addressed regarded the overfitting typical of deterministic policies. If the value-function shows some narrow peaks, the overall function could be quite overfit. This results in an overestimation which is to be avoided as mentioned before. To avoid this, the author suggests implementing noise after every action in order to smooth the Q-values over the learning period.

2.3.3. Soft Actor-Critic

When looking at Fig. 2.2, DDPG has two branches: the first is TD3 discussed above; the second is Soft-Actor-Critic (SAC) (Haarnoja et al., 2018). The main difference between DDPG and SAC is that the latter optimizes a stochastic policy rather than the deterministic policy typical of DDPG methods. Due to the stochasticity of this process, a smoothing approach similar to TD3 is also introduced. Another similarity with TD3 is brought by the double-DQN inspired structure. However, although some underlying themes can be seen, SAC brings a new concept with respect to its competitor, namely *entropy regularization.*

The concept of entropy strictly relates to how random a given variable is. In a mathematical sense, it follows Eq. (2.16):

$$\mathcal{H}(\mathcal{P}) = \underset{x\sim\mathcal{P}}{E}[-\log \mathcal{P}(x)] \tag{2.16}$$

where $\mathcal{P}$ is the entropy term. In the context of RL with stochastic policies, exploration of high-entropy regions is promoted. Therefore, if the agent develops a policy with high entropy, i.e. high randomness, it gets a reward proportional to the entropy of the policy at a given time step t. This concept can be incorporated into a modified Q-function as shown by line 12 of Algorithm 7. Here, η is the temperature hyperparameter which determines the target entropy during exploration. A set of *twin critics* generates the targets with the minimum between the two used to update the policy. The main difference from TD3 is that two function approximators are used rather than one. The policy is then updated via a mean-squared Bellman error as for DDPG. The actor policy however is updated by implementing a *reparametrization trick*. This is due to the expectation dependent on π_θ. To avoid this, π_θ can be reparametrized as an action networks which has s_t and a random noise vector as shown below:

$$a' = f_\theta\left(\xi; s'\right) \tag{2.17}$$

where ξ is the input noise, usually Gaussian (Zhang et al., 2019).

Algorithm 7 SAC (Haarnoja et al., 2018). Adapted from Dally (2021).

Initialize randomly-chosen weights θ and k_1 and k_2 for policy π_θ and twin critic $Q_{\overline{k}_j}$ networks, respectively
Initialize weights $\overline{k_j} \leftarrow k_j$ for twin critic $Q_{\overline{k_j}}$ target networks with $j = 1, 2$
Initialize initial state s_0 and smoothing factor τ
Initialize memory buffer $\mathcal{D}$
Sample initial action $a_0 \sim \pi_\theta\left(a_0 \mid s_0\right)$
for each step t **do**
    Execute action $a_t \sim \pi_\theta\left(a_t \mid s_t\right)$
    Sample r_t and $s_{t+1} \sim P\left\{s_{t+1} \mid s_t, a_t\right\}$
    Store transition $\left(s_t, a_t, r_t, s_{t+1}\right)$ in $\mathcal{D}$
    if *it's time to update* **then**
        Sample a random minibatch of n transitions $\left(s_i, s_i, \tilde{r}_i, s_{i+1}\right)$ from D
        Compute targets for the twin critics:

$$y_i = \tilde{r}_i + \gamma\left(\min_{j=1,2} Q_{\overline{k}_j}\left(s_i, a'\right) - \eta \log \pi_\theta\left(a' \mid s_i\right)\right)$$

        with $a' \sim \pi_\theta\left(a' \mid s_i\right)$
        Update twin critics with one-step gradient descent by minimizing the loss with respect to k_j:

$$L = \frac{1}{n}\sum_{i=0}^{n}\left(y_i - Q_{k_j}\left(s_i, a_i\right)\right)^2 \text{ for } j = 1, 2;$$

        Update the actor policy with:

$$J_\pi(\theta) = \underset{s' \sim P}{E}\left[\underset{\xi \sim N}{E}\left[\log \pi_\theta\left(f_\theta\left(\xi; s'\right) \mid s'\right) - Q_k\left(s', f_\theta\left(\xi; s'\right)\right)\right]\right]$$

        Update the temperature hyperparameter η with one-step gradient ascent
        Update the target networks weights: $\overline{k}_j \leftarrow (1-\tau)k_j + \tau\overline{k}_j$ for $j = 1, 2$
    end if
end for

SAC seems a promising algorithm thanks to its robustness induced by the maximum entropy approach and its sample efficient approach. However, the algorithm appears brittle to the hyperparameter η. Nonetheless, automatic temperature tuning could be a solution to this problem (Chen and Li, 2018).

2.4. Conclusion

In this chapter, the fundamentals of reinforcements learning as well as a taxonomy including the main algorithms used was given. The taxonomy presented gives an answer on *RQ-2.2* as a comprehensive overview of state-of-the-art reinforcement learning methods is presented. Firstly, it was determined whether the learning was going to be performed online or offline. In an online setting, the learning occurs *during flight* allowing the controller to learn a policy in a setting comparable to real life flight. On the other hand, offline learning is done *on the ground* where the agent has no time constraints to follow. The latter can be less computational expensive, however it may be less relevant within the scope of this research as learning offline may result in a inherently less safe policy. Therefore, it was decided to develop a two-stage method where the agent can gather information about the environment offline and use the gain knowledge to avoid risky situations while online.

By using the decision tree shown in Fig. 2.2, different approaches to RL were explored. At the top of the tree, a decision can be made between model-based and model-free approaches. The first implies that the dynamics of the environment such as probability of transitioning from one state to the other, rewards etc... are known to the agent. The second assumes that the agent has no information about the environment and it acquires knowledge by interacting with the latter. Model-based algorithms

have the drawback that the provided information may give the agent an inherently safe environment to learn. Therefore, it was decided to focus on model-free algorithms. Fig. 2.2 shows that model-free algorithms can optimize their policy in two ways, either by following a policy-based or a value-based approach. The first directly optimizes the policy π while the second focuses on Q^{π}. Additionally to these two, Actor-Critic (AC) algorithms are found to be a middle point. The use of actor-critic algorithms mitigate some of the drawbacks of policy and value-based algorithms, such as low sample efficiency and high variance. Therefore, since AC combine the best aspects of these two classes and can be used on continuous action spaces (an attractive feature for flight control tasks), it was chosen as the class of algorithm that will be implemented in this research.

A further classification was made between on and off-policy algorithms. On-policy algorithms improve the policy that the agent follows while performing exploratory actions. On the other hand, off-policy algorithms wish to improve a policy that is different than the one they use to perform actions. AC can be considered to be at a crossing between off-policy and on-policy methods as it is shown in Fig. 2.2 meaning that either approaches could be used. However, earlier on it was decided to focus on a two-stage offline-online technique. It was found that off-policy algorithms are better suited for this technique as the agent would be able to explore unsafe areas of the action space while offline and find a different safe optimal policy while online.

Finally, algorithms that meet all the criteria discussed above were further characterized. These were found to be algorithms that combine Deep Q-Networks with Actor-Critic method. The main algorithm is Deep Deterministic Policy Gradient (DDPG), a off-policy, model-free algorithm that works with continuous action spaces developed by Lillicrap et al. (2015). Two updates of DDPG take advantage of the success of this method while including some characteristics to further extend DDPG. The first is Twin Delayed DDPG (TD3) that deals with the flaw of DDPG of overestimating its value-function. The second method, Soft Actor-Critic, expands DDPG by allowing stochastic policies to be optimized rather than the strictly deterministic ones of DDPG.

3

Safety in Reinforcement Learning

Safety is deemed as the top level focus of this research. In Section 3.1, a classification of different reinforcement learning methods which ensure safety are discussed. Next, in Section 3.2, a novel method which implements the structure of a *shield* in the learning process is reviewed. The contents of this chapter will allow *RQ-2.1* and of *RQ-3.1* to be answered.

3.1. Taxonomy of Safe Reinforcement Learning

Safe Reinforcement Learning (SRL) aims at developing learning policies that maximize the return in circumstances where it is critical to respect safety constraints or achieve specific system performance during learning. Garcıa and Fernández (2015) give a comprehensive overview and classification of the two main approaches in SRL which allow safe learning. A simplified version of the taxonomy provided by Garcıa and Fernández (2015) can be seen in Fig. 3.1. From the given option tree shown, two main branches can be seen, corresponding to the two main ways SRL can be implemented. Since the term safety can be considered ambiguous, it is perhaps easier to relate to the concept of risk, which is interpreted as the inherent uncertainty of the environment. This is directly related to the oftentimes stochasticity of the environment explored by the agent (Coraluppi and Marcus, 1999; Garcıa and Fernández, 2015). This particular definition of risk is central to safety in RL, as safety can be expressed as the condition of protecting the agent from pursuing a dangerous action or reaching unsafe state spaces. Therefore, a RL method is deemed to be safe if it instructs the agent on whether an action leads to a state that could risky or not.

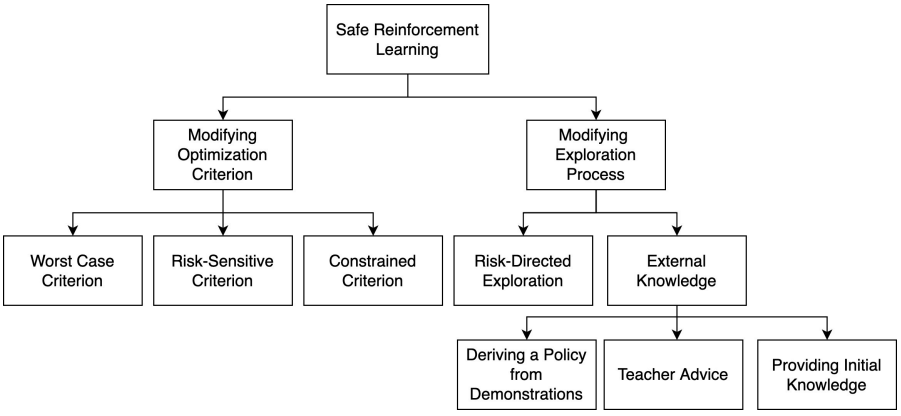


Figure 3.1: Taxonomy of Safe Reinforcement Learning algorithms analyzed in this research. Adapted from Garcıa and Fernández (2015).

In the following sections, the taxonomy showed in Fig. 3.1 will be unfolded and the main characteristics of each methods will be given.

3.1.1. Modifying the Optimization Criterion

The first approach is to modify the optimization criterion during learning. As previously discussed in Section 2.1, the agent needs to optimize a policy in order to maximize the reward. However, the actions taken in order to increase the accumulated reward may not always be safe as no metric of risk is explicitly included in the basic RL approaches. Risk is implemented in the optimization criteria in three ways that follow from the left branch of Fig. 3.1.

The first approach is **Worst Case control** where the objective is find the policy that maximizes the return with respect to the worst case scenario. This approach is particularly suited when there is inherent uncertainty about the environment or the model itself induced for example, by noisy estimations. The main contribution to worst case control method is by Heger (1994) which introduced the criterion into a Q-learning approach. The Q-function is modified into:

$$\hat{Q}\left(s_t, a_t\right) = \min\left(\hat{Q}\left(s_t, a_t\right), r_{t+1} + \gamma \max_{a_{t+1} \in A} \hat{Q}\left(s_{t+1}, a_{t+1}\right)\right) \tag{3.1}$$

where $\hat{Q}$ is a lower bound value of Q as it can be intuited by the minimization operator. This first approach however was found to be too pessimistic as it takes into account severe events which they may never be encountered by the agent. A modification of this approach was developed by Gaskett (2003) named **β-pessimistic Q-learning**. The Bellman equation of this algorithm is shown below

$$Q_\beta\left(s_t, a_t\right) = Q_\beta\left(s_t, a_t\right) + \alpha\left(r_{t+1} + \gamma\left((1-\beta)\max_{a_{t+1} \in A} Q_\beta\left(s_{t+1}, a_{t+1}\right) + \beta \min_{a_{t+1} \in A} Q_\beta\left(s_{t+1}, a_{t+1}\right)\right)\right) \tag{3.2}$$

where β is a parameter between zero and one that allows Eq. (3.2) to balance the influence between $\hat{Q}$ or Q_β. This method however was found to be more effective than the one introduced by Heger (1994) only for few values of β, which result in a still overly pessimistic algorithm (Garcıa and Fernández, 2015). Both Heger (1994) and Gaskett (2003) implement their algorithms within the model-free realm; however, only discrete state-action spaces are considered. Therefore, worst case criterion will not be further discussed.

The second approach is **Risk-Sensitive** RL where a parameter is introduced in the objective function that controls the degree of risk to be handled. The risk sensitivity parameter β can have either positive or negative values. If negative, risk is pursued; if positive, risk is averted. The main contributors to this class of SRL are Borkar (2002) and Mihatsch and Neuneier (2002). Although these methods are used in continuous space, model-free RL, their implementation require explicit values of the transition probabilities and rewards. This is not always possible in RL within flight control, hence this method is not suitable for this research.

The third and final approach considered under this branch of optimization criterion modification is **Constrained Criterion**. In this case, the aim is to maximize the expectation of the return for a policy subjected by certain constraints. Mathematically this is defined by Eq. (3.3)

$$\max_{\pi \in \Pi} E_\pi(R_t) \text{ subject to } c_i \in C, c_i = \{h_i \leq \alpha_i\} \tag{3.3}$$

where h_i is a function related to $R - t$, α_i is a parameter which constraints function h_i and c_i is one of N constraints to be fulfilled by the policy. By applying certain constraints, the set of allowable policies is reduced. This is particularly advantageous for risky domains as the best policy has to be found within the domain of safe policies Γ. Although this approach is more widely used in the field of planning and finance (Di Castro et al., 2012), it gives an intuitive metric for the concept of safety and risk as many applications know to which extent the performance of the system needs to be bounded. However, many of these approaches are not computationally tractable, making then not suitable for RL algorithms (Garcıa and Fernández, 2015).

3.1.2. Modifying the Exploration Process

The right-sided branch of Fig. 3.1 covers approaches which modify the agent's exploration process to ensure safety. In classical RL, the agent visits potentially the whole exploration space in order to develop an optimal policy. However, this method does not delineate any concept of risk, hence the

agent may be choosing dangerous actions. The exploration process can be modified in two different ways, namely risk directed exploration or by incorporating external knowledge in the learning.

Providing External Knowledge
When exploring unsafe environments, the agent will not be able to avert risky situations without being granted a some external knowledge as a risky state needs to be visited before it can be characterized as such. For this reason, providing external knowledge is a advantageous approach to aid the agent take safe actions. External knowledge can be incorporated in three different ways namely by providing initial knowledge, by allowing a teacher to give advice or by deriving a policy from demonstrations.

Initial knowledge can be considered the most elementary way of providing information to the agent. A baseline approach is to record some demonstrations from a human teacher and provide it to the agent. These demonstration allow the agent to bootstrap and develop a partial Q-function that can be used for safe exploration (Driessens and Džeroski, 2004). However, although straightforward, it provides a solid method for avoiding visiting risky states. This approach has been widely used in reinforcement learning as well as in flight control tasks. Xiong (2021) introduces the concept of a Safety Modification Layer (SML) which compares the states reached by a learning agent with known safe states. This knowledge is given to the SML upon initialization of the algorithm as a safe policy Γ as mentioned in Section 3.1.1. Therefore, Xiong (2021) effectively combines both SRL methods shown in Fig. 3.1. Pollack (2019) introduced prior knowledge in a research aimed at improving learning safety and efficiency during navigation for a Unmanned Aerial Vehicle (UAV). Together with methods for improving the learning's efficiency, Pollack (2019) introduces a SRL paradigm which has its base in giving external knowledge to the agent. The agent is given information about the state of the model in a safe state. If for any transition between one state to another, the control policy used to arrive at s_t can be reused to move to s_{t+1}, the process is considered safe. Safety is enhanced by introducing a safety filter developed by Mannucci et al. (2015). SHERPA, or Safety Handling Exploration with Risk Perception Algorithm, reinforces safe learning by overriding inputs that do not satisfy the ergodicity constraint i.e. verifies that the system is bounded to its set safety requirements (Pollack, 2019).

When operating in risky environments, especially when learning is done online, it is important to reduce the complexity of the learning process in order to be able to adjust to unforeseen dangers in a timely matter. One way to tackle this, a teacher figure can be implemented who supports safe learning. **Teacher Advice** allows safe exploration by guiding the agent to safe states while following its own safe strategy and it can also give suggestions on actions to take to prevent reaching risky status. Fig. 3.2 schematized the interactions between the different elements in teacher advice control.

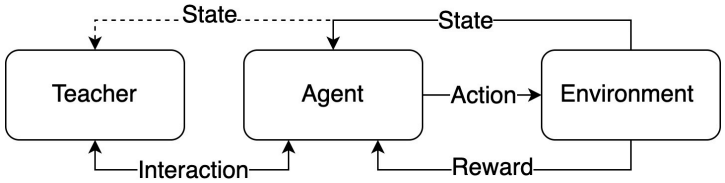


Figure 3.2: Teacher- Learning Agent framework. Adapted from Garcıa and Fernández (2015).

At each time step t, the agent observes its state s_t, decides upon an action and receives a reward after action a_t is completed. The teacher observes the state s_t in the same manner as the agent. Depending on the algorithm, either by initiative of the agent or of the teacher, advice is given to the agent in order to modify its strategy to ensure safe operations.

Teacher-Advice is implemented by Matiisen et al. (2019) as a method that enhances sample efficiency while being inherently safe. As mentioned before, this is an interesting focus due to the complexity and urgency of learning safe policies online. In this paper, Matiisen et al. (2019) proposes a teacher-student set up in which the student attempts to master a complex task. The teacher gives the student advice to increase the reward while performing tasks that have already a high learning curve. This is aimed at performing one task at the best of the agent's abilities before moving on to a less known task. However, this paper addresses the issue of forgetting optimal policies by redirecting the agent focus on tasks

where the learning curve has decreased during learning. One issue with the method proposed by Matiisen et al. (2019) is that the state space is only considered in its discrete form making it quite limiting for complex control tasks.

Geramifard et al. (2013) implements the teacher-advice method on UAV's fuel planning. In this paper, a safe function is defined, which is based on a constrained function. The constrained function S gives information on which states are allowed or not due to the risk of being unsafe. The definition of risk then follows as the probability of stepping in any of states constrained by S (Garcıa and Fernández, 2015). Therefore, the teacher consults the safe function $S \times A$ and gives the agent advice on a strategy to avoid constrained states. One drawback shared with constrained criterion method is that the unsafe states need to be known before a policy can be defined. However, this issue can be mitigated e.g. by implementing prior knowledge. The approach reported by Geramifard et al. (2013) combines teacher-advice with constrained criterion. A similar set up is presented as *shielding*, a concept which will discussed in Section 3.2.

Safety can also be achieved by **deriving a safe policy from demonstrations** which is the last method under the external knowledge umbrella shown in Fig. 3.1. This approach is also known as Learning from Demonstration (LfD) (Argall et al., 2009). The basic concept of LfD is divided in two steps. Firstly, a teacher *demonstrates* how to approach the task and its state-action transitions are documented. Secondly, the student uses the recorded transitions to learn the policy. Ravichandar et al. (2020) propose a survey for LfD paradigms in the context of robotics and automation. It is clear that LfD is widely uses as this comprehensive review discusses many application of LfD algorithms such as ground and underwater vehicles, manufacturing as well as health-care robotics. Aerial vehicles are also discussed with Ross et al. (2013) being the only paper relating to complex flight control tasks. In this research, LfD is applied to a quadrotor navigating a forest. The aim is to implement LfD to train a controller so that the UAV can successfully avoid obstacles such as trees. The agent learns from demonstrations extrapolated from a human pilot and these information are iteratively learned and corrected to boost the learning performance of the agent.

Within learning from demonstration, Inverse Reinforcement Learning (IRL) is an attractive method for many applications where a specific reward function is not known (Garcıa and Fernández, 2015). In IRL, the policy function is known and the agent strives to deduce the reward function associated with that policy. Ng et al. (2000) associates the main motivation of focus on this approaches to its potential for reinforcement learning in behavioral studies. Therefore, it could be productive in studying the behavior of a demonstrator to feed the knowledge to a learning agent. This is precisely done by Yuan (2019) which uses LfD to rebuild the reward function derived by demonstrations to facilitate the agent's learning process in the field of UAV flight control. In the research, Yuan (2019) uses existing assessments to improve on an initial demonstration after which the agent manages to produce a policy which outperforms the initial demonstration. IRL, and LfD as a matter of fact suffer from a major drawback namely that full knowledge of the trajectories from state to action (and vice-versa for IRL) need to be known (Akhtar et al., 2021). This unfortunately is not always possible. Another disadvantage is tied to the quality of the demonstrations provided. If the learner is granted poor demonstrations, an equally poor policy could be learned since the two are highly correlated.

3.1.3. Risk-Directed Exploration

The last method in the right branch of Fig. 3.1 is **Risk-Directed Exploration**. This approach is based on a *risk metric* depending on the variable w i.e. the controllability of a state-action pair. Garcıa and Fernández (2015) defines this metric as a function of the TD error: if a state results in large variability in the TD error, this state is poorly controllable. When exploring the environment, the agent is instructed to seek spaces where there is high controllability.

These methods however are extremely similar to the Constrained Criterion approaches discussed in Section 3.1.1 as controllability can be implemented as a constrain. Therefore, these methods will not be further discussed in this research.

3.2. Shielded Reinforcement Learning

The approached discussed in Section 3.1.1 and Section 3.1.2 can be combined to derive new forms of safe Reinforcement Learning. One interesting method developed from merging teacher-advice and constrained criterion is **Shielding** (Alshiekh et al., 2018). This algorithm, similarly to constrained criterion, enforces specifications that need to be followed by the learning agent. The conditions are enforced by a *shield* that observes the actions taken by the agent and intervenes only if they are deemed unsafe. Shielding can also be closely related to teacher-advice since by providing information when needed, the shield acts as a teacher. Alshiekh et al. (2018) adapts the same definition of safety as Garcıa and Fernández (2015), hence as the concept of not visiting any troublesome state. For this definition to apply, the shield must have some indication on which states are dangerous or not. Therefore, shielding also uses prior knowledge to some extent.

The approach proposed by Alshiekh et al. (2018) adds the shield as an addendum to a classical RL framework to ensure minimum interference with the environment dynamics. Fig. 3.3 and Fig. 3.4 show two classical shielding frameworks. Fig. 3.3 shows *Preemptive* shielding, where the learning loop is modified by removing the unsafe actions from the beginning as the shield is placed before the agent. This approach can be considered more intrusive, as learning is effectively constrained to the policies in the safe space Γ.

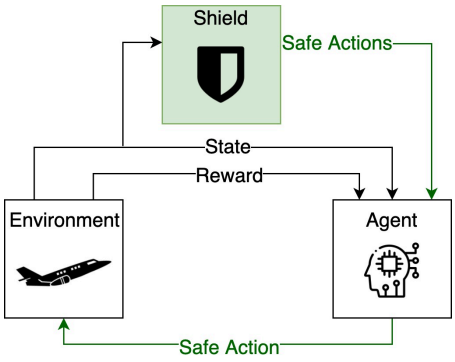


Figure 3.3: Preemptive Shielding framework. Adapted from Zoon (2021).

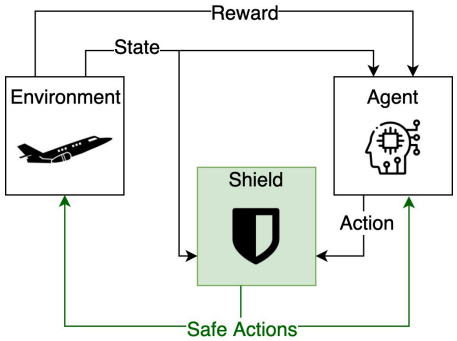


Figure 3.4: Post-Posed Shielding framework. Adapted from Zoon (2021).

The second approach shown in Fig. 3.4 is named *Post-Posed* shielding. In this case, the shield is placed after the learning agent, meaning that the shield only intervenes if unsafe actions are chosen violating the safety specification ϕ^s. As it can be seen, the shield receives both the state s_t and the action chosen by the agent a_t^1. If the actions are deemed safe with respect ϕ^s, the shield will not activate but remain in observation mode. Otherwise, safe actions $a_t \neq a_t^1$ will be suggested to the agent so future risks can be avoided. In order for a_t^1 to be discarded in future learning, a punishment r'_{t+1} can be assigned so that the agent learns that the action violated ϕ^s. Jansen et al. (2018) introduce the concept of a rank_t. The agent provides a ranking of actions from most desirable to least. The shield selects the action that is safe according to the safety specification. If none of the given actions are deemed safe, the shield proposes an action $a_t \notin \mathrm{rank}_t$. This approach is found to be more efficient considered that if the shield chose a new action a_t which does not belong to the ranking, all the actions proposed by the agent can be discarded by assigning a punishment. This effectively speeds up the learning process.

Alshiekh et al. (2018) and Jansen et al. (2018) both apply shielding to game theory and some Atari games. Although not comparable to the flight control tasks which will be studied during this research, results on games give a good indication of the overall performance of a specific method. Jansen et al. (2018) consider the arcade game PAC-MAN where the task is for the agent to eat food in a maze while avoiding ghosts. In this example, the reward is defined as the food eaten. The longer it takes the agent to complete the task (i.e. eat all the food) a penalty is introduced. The agent is also penalized if it is eaten by a ghost. The shield is created via the probabilistic model checker *Storm* (Dehnert et al., 2017). This game is solved via Q-learning first and then by implementing the aforementioned shield to this RL technique. It should be noted that a episode is defined as the time until all the food is eater

or until PAC-MAN is eaten by the ghosts. Fig. 3.5 shows the environment for both the shielded and unshielded case for the same arbitrary instance. On the right two out of three of the potential decision that PAC-MAN could take are indicated as *safe* (the green squares) while on the same instance on the left, the episode is about to end. This shows that for this instance, the shielded version is more successful than the classical RL framework. This is supported by monitoring the average score in the long term as shown in Fig. 3.6. It can be seen that the average reward is consistently higher for the shielded case even on the very first training episode. It can also be noted how the learning curve is steeper for the shielded case, meaning that an optimal policy is reached faster than the unshielded case.

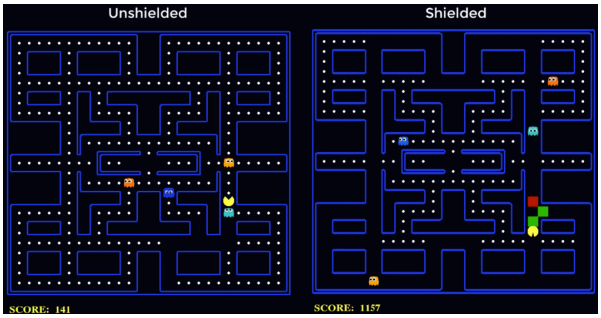


Figure 3.5: Classic PAC-MAN environment. Retrieved from Jansen et al. (2018).

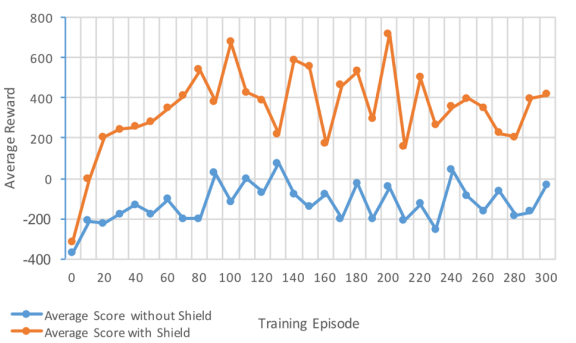


Figure 3.6: Resulting scores for shielded and classical learning. Retrieved by Jansen et al. (2018).

As of 2021, shielding has not been explicitly used in any flight control tasks. However Zoon (2021) proposes shielding for autonomous vehicle control implemented in a Double-DQN. The choice of the base RL algorithm was made by realizing that shielding has been only introduced for discrete action spaces whilst traffic control replies on continuous state spaces. This made Double-DQN a appropriate candidate for this research. Zoon (2021) introduces two shielding approaches both based on post-posed shielding namely Safety Checking Shield (SCS) and the Safe Initial Policy Shield (SIPS). SCS is the baseline of SPS: operations follow exactly as shown previously in Fig. 3.4. SIPS however, implements a different method to overrule an unsafe action. A parameter ρ is introduced which gives an estimate on how much a proposed action deviates from a safe action determined by ϕ^s. Once the agent proposes rank_t, the shield approximates the deviation ρ from a safe action given by prior external knowledge. If the deviation is small, the action is taken; otherwise it is overruled. Both approaches were implemented within the Double-DQN and applied to autonomous driving scenarios such as steering, accelerating and breaking.

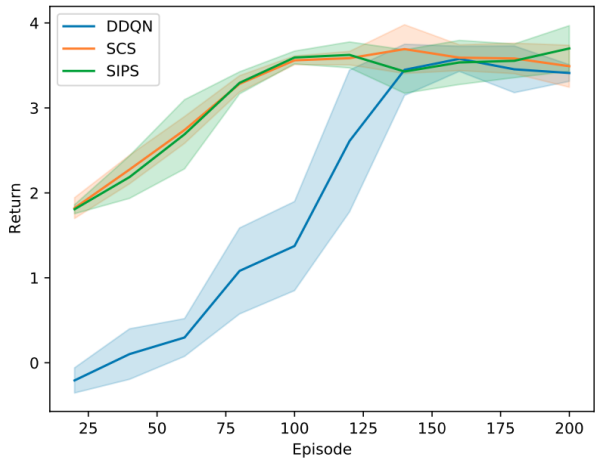


Figure 3.7: Sum of rewards over episodes when training an RL agent using the regular DDQN, the SCS and the SIPS Retrieved from Zoon (2021).

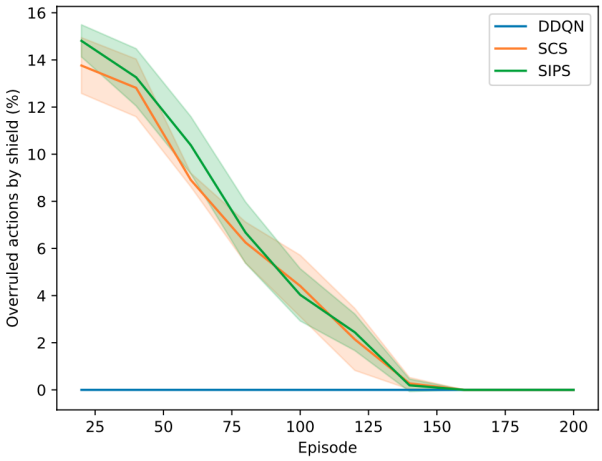


Figure 3.8: Percentage of overruled actions over episodes when train- ing an RL agent using the regular DDQN, the SCS and the SIPS. Retrieved from Zoon (2021).

The results shown in Fig. 3.7 and Fig. 3.8 are for a scenario with one lane, hence no steering is required. Fig. 3.7 shows the return for both shielding methods and for Double-DQN. SIPS and SCS perform quite similarly: this relates to the fact that both shields are overruling the same unsafe actions. A striking improvement can be seen with respect to Double-DQN. Since Double-DQN takes random actions at the beginning, there is more chance of receiving a penalty. This is avoided by the shielded method as a penalized action will not be proposed again. Fig. 3.8 shows the percentage of overruled actions throughout the learning. Of course, Double-DQN will always have a value of zero since there is no architecture that allows decisions to be overruled. Again, both SIPS and SCS perform similarly. It can be seen that both methods converge to a point when no actions are overruled. This follows from the concept just highlighted: since unsafe actions are not re-proposed once overruled, at the end of the learning the shield will only be proposed actions that do not violate ϕ^s.

Zoon (2021) successfully implemented shielding within the field of autonomous vehicles. The results showed improved learning performance and overall *safer* actions taken since by the end of the learning no more unsafe actions are proposed. One interesting suggestion would be to see how the system would react to a representation of the environment which is not fully correct or incomplete. Indeed one of the shortcoming of shielding is that prior knowledge is necessary for the shield to be constructed. However, Alshiekh et al. (2018) argue that this is unavoidable in SRL since safe actions are related to the environment.

3.3. Conclusion

In this chapter, the main Safe Reinforcement Learning (SRL) approaches delineated by Garcıa and Fernández (2015) have been reviewed to answer the research questions presented in Chapter 1. First of all, *RQ-2.1* was answered as safety in RL was defines as the condition of protecting the agent from pursuing a dangerous action or reaching an unsafe state space. Following, the different SRL methods shown in Fig. 3.1 were illustrated. The first class of approaches was based on modifying the optimization criterion. These methods actively introduce the metric of risk so that the policy is modified only to include safe actions. Some methods that implement this are worst case, risk-sensitive and constrained criteria. Although most of these methods do successfully enable the agent to remain within a safe domain, their implementation require explicit values of the transition probabilities and rewards. This however, is not always possible in RL within flight control.

The second class of approaches is based on modifying the exploration process so that safety can be ensured. This can be done in two manners, either by providing the agent with external knowledge or by conducing risk directed exploration. External knowledge might be crucial in SRL as the agent can avoid unsafe actions and states by knowing whether they are effectively such. External knowledge can be given as ad advice presented by e.g. a teacher, by a policy which is derived by a demonstration or by simply providing initial knowledge. These methods are effectively used in SRL and may be an interesting option for this application in flight control.

Another approach not discussed by Garcıa and Fernández (2015) is to combine the two branches of Fig. 3.1 to develop a method that draws all the advantages of the different algorithms. This however is proposed by Alshiekh et al. (2018) as Shielding. This research proposed to combine teacher-advice methods and constrained criterion to implement a shield, assigned to block any unsafe action to be performed by the agent. Although this technique has yet to be used in flight control, research on autonomous driving cars has found this method to be particularly fictitious.

This chapter presented an overview of state-of-the-art RL methods according to which safety can be improved. In this way, *RQ-3.1* has been answered.

4

State-of-the-Art of RL in Flight Control

This chapter will answer *RQ-2.3* by giving insight in state-of-the-art approaches to control fixed wing aerial vehicles using DDPG methods. Section 4.1 will give an overview of the research carried out by Milz and Looye (2020) on a flight controller based on DDPG as well as PID. Right after, in Section 4.2, a summary of a research carried out by Tang and Lai (2020) on the design of a automatic landing controller using DDPG is given. Finally, Section 4.3 presents a research on the use of DDPG to control a fixed wing aircraft created as means of verification for real-life cases.

4.1. Design and evaluation of advanced intelligent flight controllers

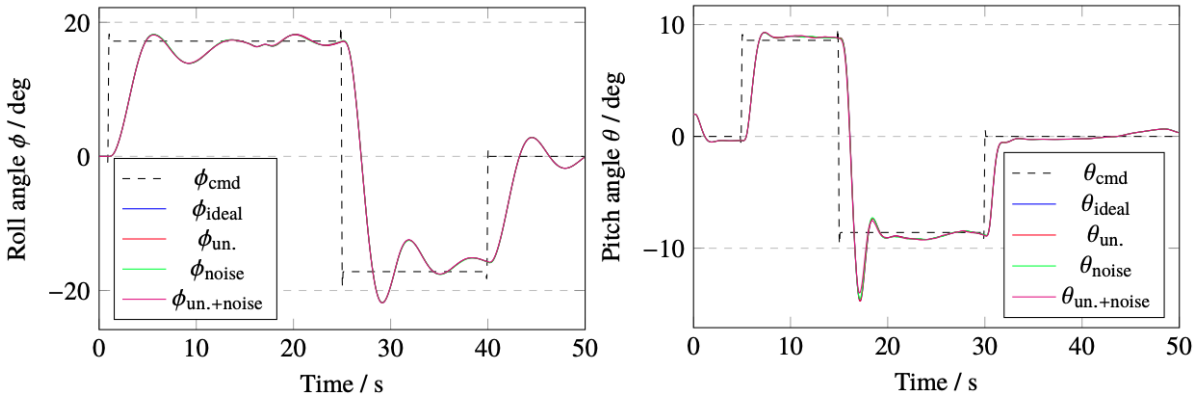
Milz and Looye (2020) propose a research focused on applying RL methods to robust and adaptive flight control tasks. This paper is particularly interesting in regard to this research as adaptive flight control allows safety to be ensured when encountering unexpected situations. As mentioned in Chapter 1, non-learning controllers have been a key element in flight control for decades. Past adaptive controllers were based on the concept of indirect adaptive control. The working principle is based on a dynamic model that is regularly updated using system identification algorithms. Recently, the dependency on this dynamic model has been reduced by implementing Incremental Nonlinear Dynamic Inversion (INDI). This method is made of an online system identification that incrementally updates an inverse model of the system, allowing changes in the dynamics of the environment to be handled in a continuous matter.

In the work of Milz and Looye (2020), an INDI controller is used as a benchmark to compare the efficacy of learning controllers when working on the automatic landing of a large cargo aircraft. The two intelligent controllers are equipped with a DDPG and a PIDNN algorithms. The latter approach is a controller consisting of a NN with three layers each made of proportional, integral and derivative neurons. The addition of a NN to the standard PID controller allows the system to be a multiple-input-multiple-output (MIMO) system. This is necessary as both the tracking error and the output need to be monitored. The DDPG agent implemented follows the description given in Section 2.3.1 and developed by Lillicrap et al. (2015) allows to include the dynamical behavior of the aircraft in the network.

The benchmark case discussed in Milz and Looye (2020) has four scenarios:

1. No disturbances and no noise;
2. Gaussian distributed noise with $\mu = 1$ and $\sigma^2 = 0.05$ applied to the output;
3. Changing lift and pitch moment coefficient with respect to the angle of attack;
4. A combination of case 2 and 3.

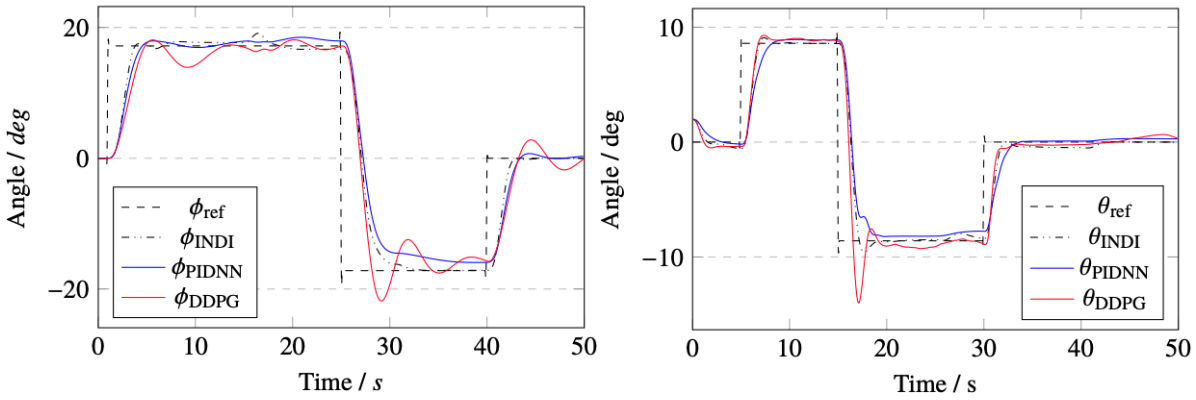
In this work, the roll angle ϕ and the pitch angle θ will be tracked. After these parameters are set, the performance of the DDPG and PIDNN controllers can be analyzed and compared to the INDI controller for all scenarios. An overview of the DDPG controller behavior for the different scenarios is given in Fig. 4.1a and 4.1b. It can be seen that the responses are barely different suggesting that noise and disturbances do not influence the DDPG controller. This is an attractive result as real-life conditions are



(a) Lateral **(b)** Longitudinal

Figure 4.1: Combined simulation results of the DDPG controller on the four different scenarios. Retrieved from Milz and Looye (2020).

far from the idea case used in many simulations. It can be seen that all the results display an oscillatory transient behavior. The authors note that considering the consistency among the different cases, this behavior may be caused by the training of the agent if the optimization does not improve after falling into a local minima. To analyze and compare the different controllers, the authors decided to show only



(a) Lateral. **(b)** Longitudinal.

Figure 4.2: Comparison of all applied controllers. Retrieved from Milz and Looye (2020).

the ideal condition since it was shown, for both DDPG and PIDNN, that the trends are the same for all conditions. In general, both the non-learning INDI controller and the learning PIDNN one show an overall better performance than DDPG. The oscillatory transient behavior described before worsen the performance of the DDPG. Nonetheless, Milz and Looye (2020) suggest that working on the training stage and adjust the networks hyperparameters could be an effective solution to improve performance. Additionally, the ability of DDPG to execute the desired action for different uncertain conditions are considered of great interest for further research in the field of adaptive control. Therefore, although the results are not in favor of DDPG, the authors appear confident in the possible improvements that could be achieved in future research.

4.2. Automatic landing control using DDPG

Approach and landing phases are challenging sections of the flight envelope even when performed by a controller due to many disturbances as well as limitations caused by structural, aerodynamic and aviation regulations. In order to offer an alternative to the conventional PID controllers, Tang and Lai (2020) focus on developing a learning controller for a commercial fixed wing aircraft equipped with a DDPG algorithm.

When landing, the aircraft follows the glide and then the flare path until touchdown. In this research, the altitude and the vertical velocity will be tracked and constrained to follow the limitations set by the aircraft manufacturers and the governing bodies. Wind disturbances modeled with a Dryden model have been introduced to the model to subject the controller to more realistic conditions. The goal of this simulation is of course to land successfully while following the requirements mentioned above. However, this sparse reward strategy is not deemed successful by the authors as the agent may be lured to explore risky policies or not reach convergence. Therefore, a new dense reward function was created to lead the agent along the correct path by increasing the reward by small steps.

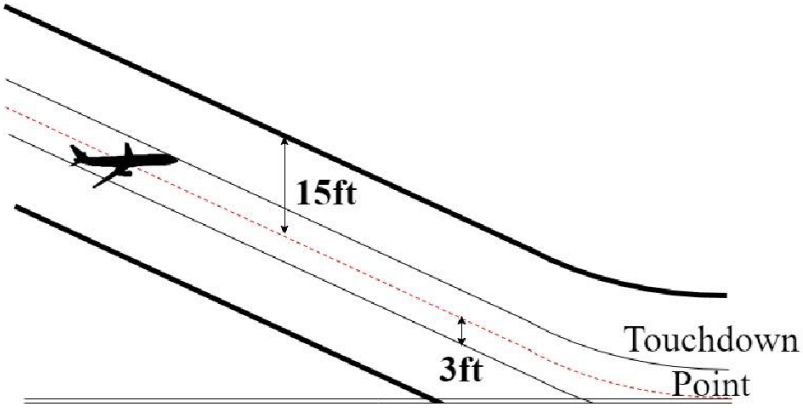
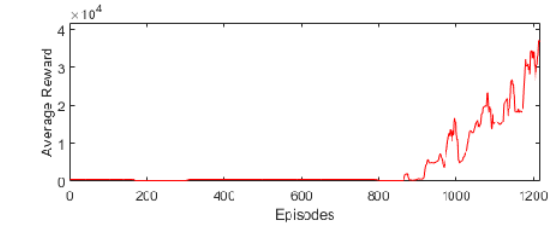
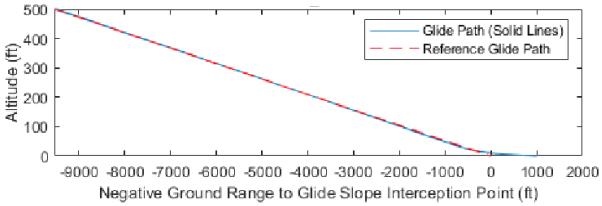


Figure 4.3: Altitude boundaries for Reward R_h. The aircraft can deviate 15 ft above the glideslope and 3 ft below. Retrieved from Tang and Lai (2020).

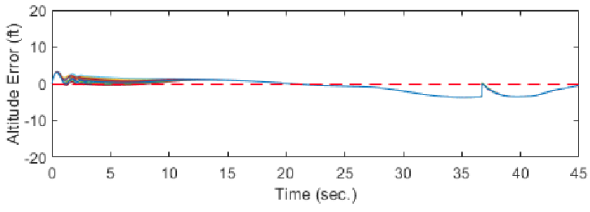
In this new reward function, the total reward R_T depends on deviations from the glideslope, deviations from the vertical velocity limits and deviations from the intended touchdown point. For example, the reward given to the agent for maintaining a correct altitude can be visualized in Fig. 4.3. The agent acquires a positive reward R_h if it follows the glideslope with a maximum deviation of 15 ft above it or 3 ft below it. Tang and Lai (2020) validated the results by simulating the DDPG in more than 1500



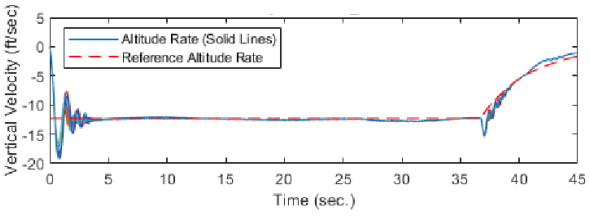
(a) Average reward R_T. Retrieved from Tang and Lai (2020).



(b) Glideslope path. Retrieved from Tang and Lai (2020).



(c) Altitude error. Retrieved from Tang and Lai (2020).



(d) Vertical velocity. Retrieved from Tang and Lai (2020).

Figure 4.4: Results from simulation. Retrieved from Tang and Lai (2020).

different flight conditions each with a different wind profile. Since the authors identified a trend among all DDPG agent trained, they decided that only one case will be considered for the sake of conciseness. The results presented here are for flight conditions with a vertical wind of $20ft/s$. Fig. 4.4a shows the average reward for 1200 episodes. It can be seen that initially, the values remain low until the 800th episode. After this plateau, the learning curve increases and convergence is reached. Interesting to see from Fig. 4.4d that the glideslope is almost perfectly followed from the beginning of the maneuver

until touchdown. Regarding the altitude error, it can be seen from Fig. 4.4c that the error is consistently less than 3ft, following the set constraints.

The work presented by Tang and Lai (2020) is an interesting source as it indicates that a DDPG agent can successfully control an aircraft to land with disturbances such as wind introduced. It is also particularly relevant for this thesis since some states have been chosen as risky and the agent successfully recognized them as such during the simulations. However, similarly to the previous research mentioned, the choice of DDPG hyperparameters significantly affected the results of the simulations. Therefore, the authors advise to direct any future work's attention to fine tuning these values.

4.3. DDPG attitude control for low-cost aircraft

In recent years, intelligent flight control has been a main focus for many researchers. However, before its full scale implementation can be reached, consistent results from real-life trials need to be achieved. One main issue is that traditional aircraft are costly and not readily available for research. Therefore, the lack of validation means delays in the use of learning controllers for commercial use. Wang et al. (2020) individuate this gap and propose to model a reusable fixed wing aircraft which can be handled by learning controllers and provides verification data without incurring high costs.

The aircraft is modeled as a scaled-down medium-size jet that can be built by 3D printing. Since the aircraft is being built ad-hoc, not all performance characteristics can be known. For instance, aerodynamic data can be acquired via wind-tunnel measurements. To avoid these cost, approximations are gathered via aerodynamic software. The uncertainty of the model is a main factor that encourages the authors in working with model-free algorithms. DDPG is chosen as it is model and policy free, and can be implemented in continuous control which is a focal point in the process of implementing learning flight controllers. The DDPG algorithm is made of four NNs: two critic networks and two actor networks.

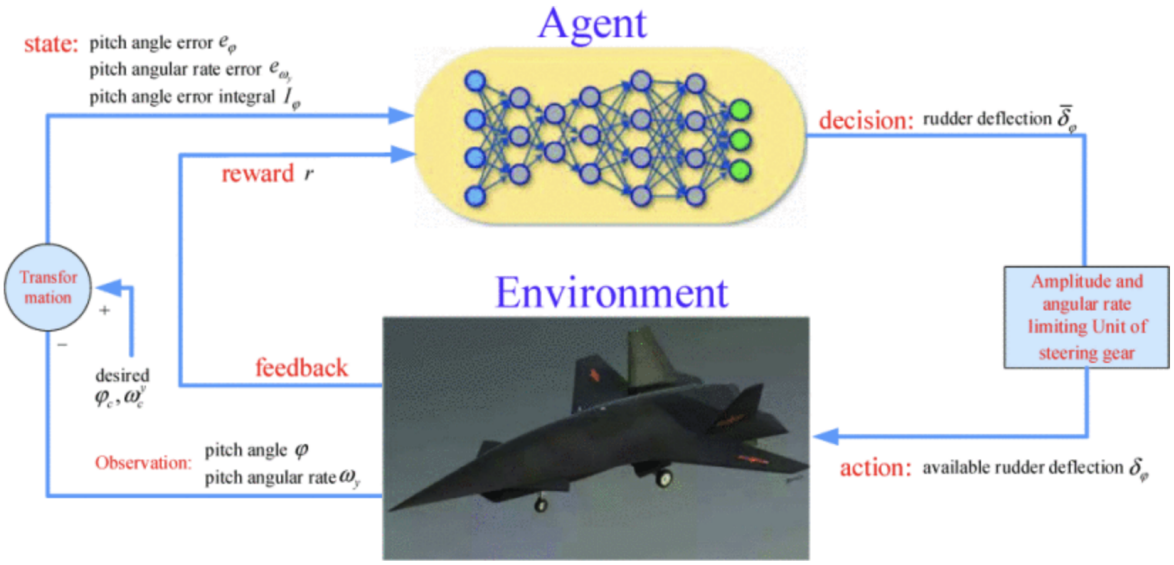


Figure 4.5: Controller architecture. Retrieved from Wang et al. (2020).

From Fig. 4.5, the state and action spaces can be deduced. The states are the pitch angle error e_ϕ, the pitch angular rate error e_{ω_y} and the pitch angle error integral defined as follows:

$$I_\varphi(k) = \sum_{i=0}^{k} e_\varphi(i) \cdot T \tag{4.1}$$

where T is the simulation time. The action is the rudder deflection δ_ϕ. With these information, Wang et al. (2020) define the actor and critic networks which can be seen in Fig. 4.6. This method applies a noise to the output in the form of a normal distribution with $\sigma^2 = 3$ and the μ equal to the output of the online

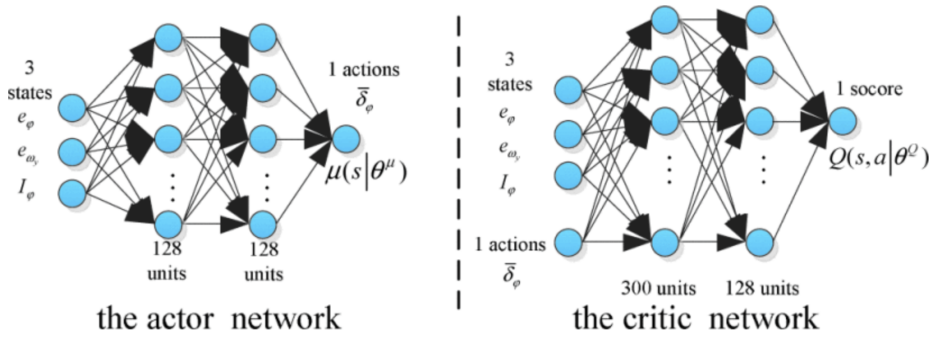
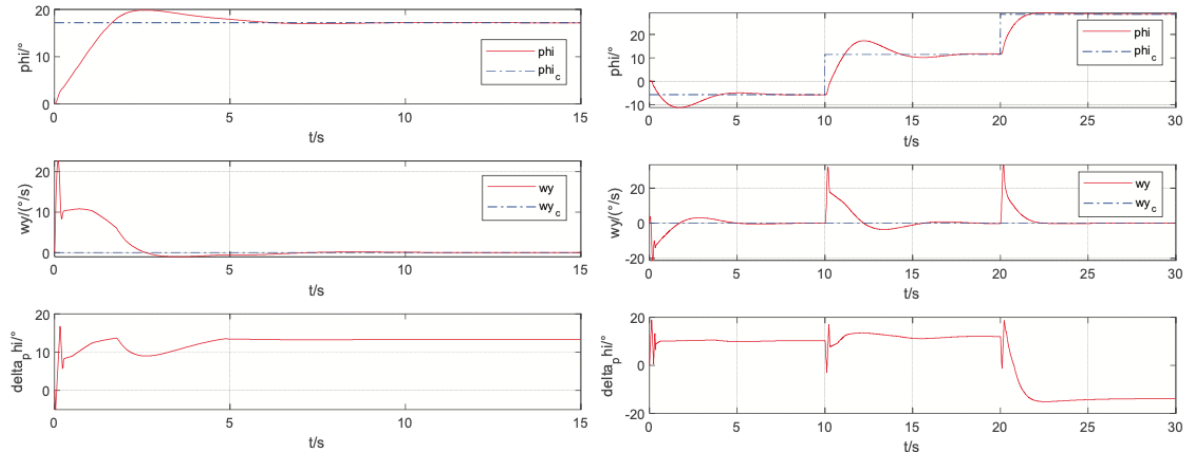


Figure 4.6: Network architecture. Retrieved from Wang et al. (2020).

actor policy network. While the variance starts at 3 when learning begins, it decays with a discount rate of 0.9995 while training so that the output becomes a fully deterministic action. The initial conditions for the simulation are set to zero for both pitch angle and rate while the goal is to reach a pitch angle of 17 degrees and keep the pitch angular rate equal to zero. Additionally, a generalized goal for which the agent was not trained is simulated. This is shown in Fig. 4.7b where the pitch angle increases in three steps, the pitch angular rate is still kept at zero and the rudder deflection switches of 180 degrees. As it can be seen from the top graphs of Fig. 4.7a and 4.7b, the pitch angle smoothly converges to the desired value after a small overshoot. For the pitch angular rate and the rudder deflection, the desired conditions are reached after an overshoot accompanied by a small oscillatory section. Similarly to what seen before in Section 4.1, this could be due to a behavior acquired during training as well as the choice of hyperparameters. When monitoring the errors of the research specific case, Fig. 4.8



(a) Research specific case.

(b) Generalized case.

Figure 4.7: Control results from simulation. Retrieved from Wang et al. (2020).

shows that both e_ϕ and e_{ω_y} are correctly tracked by the agent. Both errors converge to zero in the first third of the simulation, which demonstrate that the algorithm can successfully track errors. Wang et al. (2020) research indeed demonstrates that implementing a learning controller within a low-cost fixed wing aircraft (for which a complete accurate model is not available) gives effective results. The approach can be considered straightforward while still providing exhaustive proof that this beginning to end design can be used to verify further research on RL based controllers. The authors suggest that the reward function should be further studied to provide an improved performance as well as introducing a stochastic noise to the output rather than the current normally distributed one.

4.4. Conclusion

In the previous section, an overview of recent advancements of RL in flight control were presented. Section 4.1 described an interesting paper that compares DDPG controllers to a PIDNN. Both learning

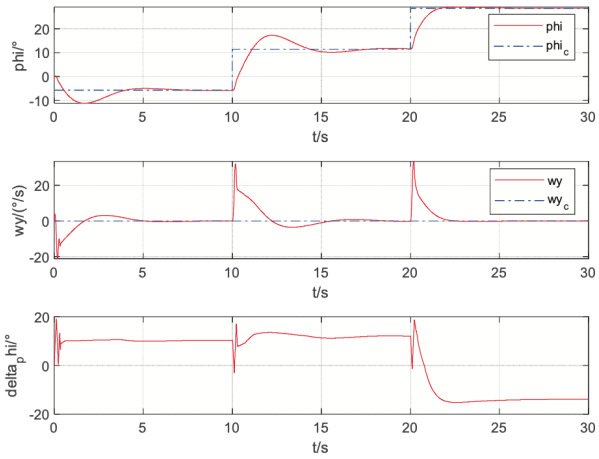


Figure 4.8: Control errors from simulation. Retrieved from Wang et al. (2020).

controllers were also compared with a non-learning one based on the concept of INDI. In the end, both learning controllers successfully completed the required tasks with PIDNN doing a better job since DDPG shows an oscillatory transient behavior. However, Milz and Looye (2020) suggest that this may be due to the chosen DDPG hyperparameter and recommend to perform further work on them.

Section 4.2 proposed a DDPG based controller for landing of a fixed wing aircraft. This paper was especially relevant as some of the actions that could be taken by the agent were constrained to allow landing to happen safely according to structural or regulatory constraints. Although the controller was successful in landing the aircraft, the authors individuate a similar trend to the research before regarding the hyperparameters.

Finally, Section 4.3 proposed a research for a DDPG attitude controller implemented on a fixed wing aircraft of which not all necessary modeling parameters are known. This was done to improve the existing means of verification so that the implementation of learning controllers in real-life could be accelerated. The research successfully designed a DDPG controller that is able to learn a policy for which it was not trained before. However, since the noise introduced is simply normally distributed, the authors suggest that future work should be spent on implementing a stochastic noise to the output. By presenting these papers on state-of-the-art of reinforcement learning in flight control, *RQ-2.3* can be considered as answered.

5

Preliminary Analysis

In this chapter, a Q-learning agent will learn how to navigate the Grid World environment when a shield is implemented in the loop. The aim is to assess whether a RL algorithm with similar characteristics to DDPG is able to successfully complete a task while a simplified version of a shield is applied. Not only this exercise is aimed at giving confidence about the validity of the methods described before, but it is aimed at gathering some practical knowledge about the approaches that will later be applied to the control of the Cessna Citation. The chapter begins with a description of the environment in Section 5.1. The Q-learning agent is discussed in Section 5.2 followed by an overview in Section 5.3 of the shielding techniques used in this analysis. Results will be presented in Section 5.4 while concluding statements will be given in Section 5.5

5.1. Environment Description

In this analysis, the Matlab Grid World environment is used to test the workings of some of the algorithms discussed in previous chapters. The environment is composed of a 10-by-10 grid as shown in Fig. 5.1, where an agent can move in four different directions namely North, South, East and West. The agent begins from the top right cell `[1,10]` shown in yellow. The goal is to arrive to the blue at `[8,4]` while avoiding the obstacles represented by the black cells.

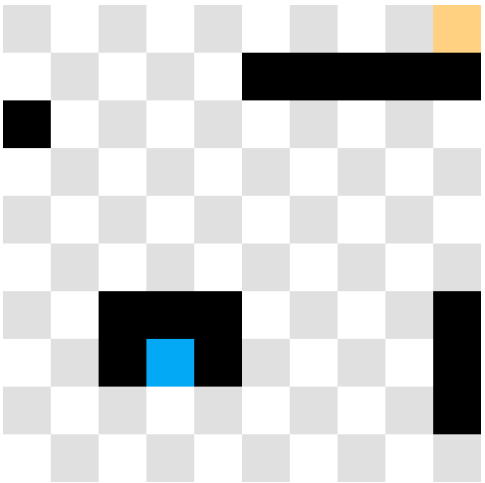


Figure 5.1: 10-by-10 Grid World environment. The black cells represent the obstacles, the yellow cell is the initial position of the agent while the blue cell is the target.

As the aim is to reach the target state in the least amount of steps possible, the agent receives a reward of -1 for each action taken. Once it reaches the target, a reward of +50 is received. It should be noted that all transition probabilities from one state to another are set to one, while the transition to any of the obstacles states are always zero.

5.2. Q-Learning Implementation

In this simulation, a Q-learning agent that follows the algorithm described by Algorithm 4 is implemented. Although it was decided to work with a DDPG agent in previous chapters, the choice of Grid World does not allow such an algorithm to be used. The action space of DDPG is required to be continuous, while Grid World's actions are four discrete steps. Additionally, the implementation of neural network is not deemed necessary due to the simple environment, making DQN algorithms superfluous. Since Q-learning shares the main characteristics of DDPG i.e. being model-free and off-policy, it is chosen to carry out this analysis.

This experiment is similar to one carried out by Alshiekh et al. (2018) where a Q-learning agent navigates a 9-by-9 environment with obstacles and sub-goals. The hyperparameters were kept as the ones given in the aforementioned paper and are reported in Table 5.1.

Table 5.1: Q-learning agent hyperparameters. Values from Alshiekh et al. (2018).

Hyperparameter	Value
Learning rate λ	1
Discount factor γ	0.99
Exploration probability ϵ	0.04

The learning occurs over 400 episodes each lasting a maximum of 50 steps. While the 400 episodes is an arbitrary choice, the 50 steps are a default option of the Matlab environment that has been kept unchanged.

5.3. Shielding Implementation

In Chapter 3, it was decided to implement shielding as the safe Reinforcement Learning method. In the analysis presented here, a simplified version of the shielding technique applied by Alshiekh et al. (2018) is proposed. Some of the steps taken here do not follow the method described in Section 3.2 but rather propose a solution of the same essence. The approach begins with synthetizing a reactive system such that a safety specification ϕ^s is followed. This reactive system can be placed at different points in the control loop, as shown in Fig. 3.3 and 3.4. In the case of a preemtive shield, the unsafe actions are directly removed so that the agent does not have the ability to choose such actions. This case can be considered as the case with obstacles states in the grid. Since the transition probability of going from state s to an unsafe state (represented by the obstacle) is zero, the agent will never be able to select an action that leads to danger.

The second method is a post-posed shield, where the actions are monitored and corrected only if an action that leads to an unsafe state is selected. One of the simplifications mentioned before is applied in this case. In contrary with the method of Alshiekh et al. (2018), the shield is not a reactive system that itself intervenes during learning. An agent is free of taking any action, but it will receive a punishment if an unsafe state is reached. Hence, actions that go against ϕ^s are not substituted but instead they are learned by the agent to be disadvantageous. Although this approach does not avoid unsafe actions to be excluded from the policy, this also cannot be avoided in the original algorithm as mentioned by Alshiekh et al. (2018).

Fig. 5.1 shows the adjusted grid environment with unsafe states used to simulate a post-posed shield. These can be seen in orange and they are always located adjacent to the obstacle states in black. If an agent chooses an action that leads to an unsafe state, a penalty of -10 is allocated. This penalty makes unsafe states highly unattractive and allow an optimal and safe policy to be learned. Another approach discussed in Section 3.2 is to let the agent follow actions which are ranked from most to least desirable. In this analysis, not all steps taken by the agent follow a ranking. Two states are selected, namely `[3,3]` and `[5,7]` as they are located at the two possible corridors that the agent may follow to arrive at the target destination. Jansen et al. (2018) states that not all possible actions need to be ranked, therefore selecting only two states was deemed a viable solution. The ranking is translated into a reward scheme, with the best action earning a reward of +2 and the worst a penalty of -5. An action that results in being adjacent to an unsafe state results in a penalty, otherwise no penalty is given. If the

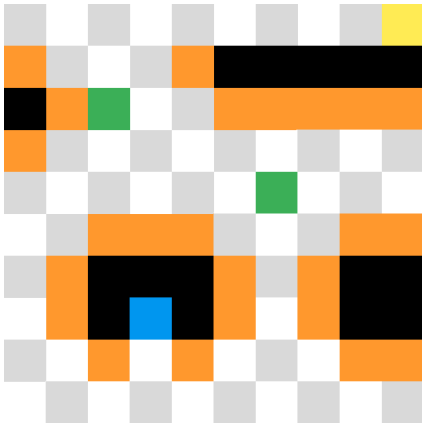
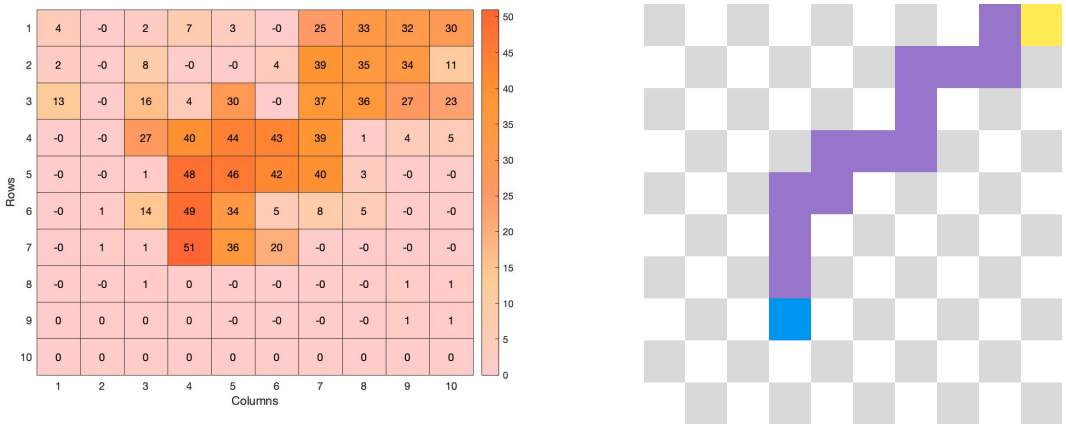


Figure 5.2: Modification of 10-by-10 Grid World environment. Unsafe states indicated in orange and states with ranked actions are in green.

action results in arriving at a state that is not adjacent to an unsafe one and positively contributes to reaching the goal, a reward is given. For example, if the agent is at state `[3,3]`, going East or South will result in a reward while going West is penalized. The action of going North is not ranked as it does not positively contribute i.e. the path to the target becomes longer hence not optimal.

5.4. Results

In the following section, results for different scenarios are presented. The results should give an indication of the performance and the safety of the generated policy as well as the balance between these two factors. *Scenario 1* is the simplest one that can be done with Grid World where no obstacles, no unsafe states and no ranked action states are present. This was done simply to make sure that



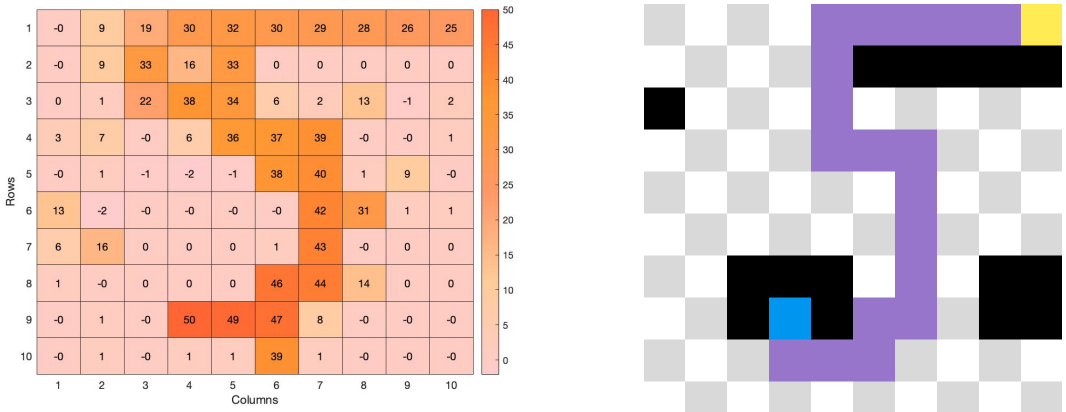
(a) Policy visualization.

(b) Optimal path found by agent.

Figure 5.3: Policy and path visualization for Scenario 1: no obstacles, no unsafe states and no ranked action states.

the Q-learning agent and the hyperparameters chosen are working correctly. It can be seen from Fig. 5.3b that the agent successfully reaches the target state with the shortest path possible. Fig. 5.3a give information about the exploratory pattern of the agent. It can be deduced that most of the states explored lay on the upper right side of the grid: since the agent has no notion of obstacles and unsafe states, the exploration occurs in a unorganized manner. *Scenario 2* is represented by a grid with only obstacles states. The agent knows that obstacles states may not be reached, hence it tries to find the shortest path while avoiding the obstacles. It can be seen from Fig. 5.4b that the shortest path is indeed found with a reduced amount of exploration of the environment. It can be seen from Fig. 5.4a that although some exploration is done, at the end of the episodes the agent chooses a policy which is not the shortest possible. This could be due to the choice of a low exploration probability factor as the

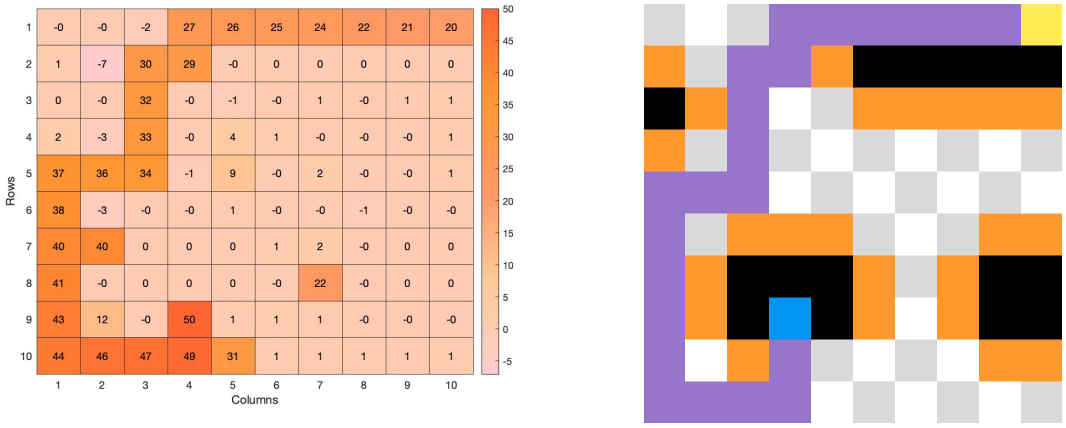
agent tends to exploit the current reward rather than keep exploring possible better options. This case represents the preemptive shield presented by Alshiekh et al. (2018) since all states that are considered unsafe are unavailable to the agent.



(a) Policy visualization. **(b)** Optimal path found by agent.

Figure 5.4: Policy and path visualization for Scenario 2: obstacles but no unsafe states and no ranked action states.

Fig. 5.5 show the policy and path visualization for *Scenario 3* where the states adjacent to obstacles are considered unsafe. The agent is able to visit these states, but a penalty is awarded for doing so. From Fig. 5.5b it can be seen that at the end of the episodes, the agent does find the optimal path while avoiding the unsafe states. However, from Fig. 5.5a is can be observed that the optimal policy

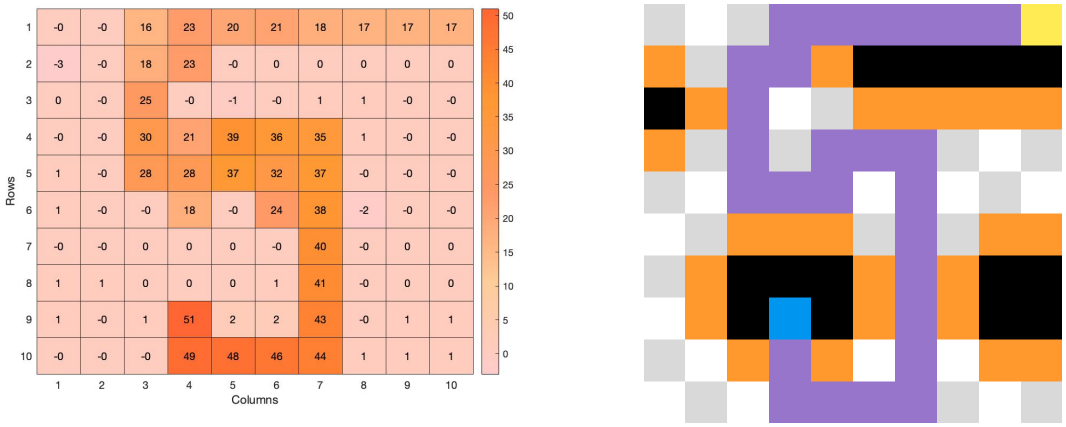


(a) Policy visualization. **(b)** Optimal path found by agent.

Figure 5.5: Policy and path visualization for Scenario 3: obstacles and unsafe states present but no ranked action states.

is reached by exploring unsafe states which is to be expected. As mentioned in Section 5.3, unsafe actions may be part of the learned policy since the agent learns to avoid them by effectively exploring these unsafe states. Finally, Fig. 5.6 shows the results for *Scenario 4* where the obstacles and the unsafe states kept the same as *Scenario 3*, but some states with ranked actions are added. These follow the description given in Section 5.3. From the path visualization in Fig. 5.6b, it can be seen that the agent successfully avoids the unsafe states and uses the ranking of actions provided correctly. In fact, it moves South at both `[3,3]` and `[6,7]` gaining the reward for performing the highest ranked action. However, it can be seen that the generated path is not the shortest one that can be achieved in this environment. The reason could lay behind the hyperparameters as mentioned in the results of *Scenario 2*. From Fig. 5.6b it can be seen that the unsafe states are successfully avoided and barely explored. This shows that applying this method does in fact help prevent visiting unsafe states in the environment. It should be noted that since there is no guarantee that unsafe actions are not part of

the policy, the shield should be implemented in the stages beyond the learning phase (Alshiekh et al., 2018).



(a) Policy visualization. **(b)** Optimal path found by agent.

Figure 5.6: Policy and path visualization for Scenario 4: obstacles, unsafe states and ranked action states present.

It is necessary to also look into the performance of the algorithm for the different scenarios. A summary of the conditions for each scenario is given in Table 5.2 while Fig. 5.7 shows the average reward gained by the agent through the simulation episodes. Unsurprisingly, *Scenario 1* converges the fastest of all since the agent does not need to learn to avoid certain states. It can be seen from the figure that agent accumulates the maximum average reward in *Scenario 1*; this is due to the fact that the reward scheme does not include additional penalties other than the ones related to moving from a state to the next and

Table 5.2: Summary of different scenarios for the Grid World preliminary study.

Scenario #	Obstacles	Unsafe states	Ranked actions
1	✗	✗	✗
2	✓	✗	✗
3	✓	✓	✗
4	✓	✓	✓

that the path taken by the agent is the shortest between all cases. In terms of average reward, *Scenario 2* and *3* converge to a similar value. Since *Scenario 3* optimal policy successfully avoided unsafe states, the agent was not penalized other than for taking steps to reach the target states. Therefore, it is not surprising that the two algorithms received a similar average reward upon convergence. However, it can be clearly seen that at the beginning of the training the agent from *Scenario 3* does receive much more negative rewards. This is a byproduct of exploration and it can not be avoided. In order reduce the amount of negative reward, the unsafe actions have to be removed. This is indeed the case for *Scenario 2* where a preemptive shield is implemented and the average reward stays more or less constant until convergence is reached. Regarding the convergence, all algorithms show a similar learning curve. However, *Scenario 1* and *Scenario 4* show faster convergence than *Scenario 2* and *3*, even if by little. In fact, although *Scenario 4* shows a similar behavior to *Scenario 3* before convergence, this agent reaches a optimal policy before. This can be attributed to the fact that for certain states the agent is not required to explore different options but can simply follow the best ranked action. In this way, exploration time is reduced and convergence can be promptly reached. Although the difference from the cases with shielding but no ranked actions is minimal, this study only provided ranking for actions of two states. Therefore, the learning curve could be further improved by providing a ranking for more actions.

5.5. Conclusion

This section concludes the preliminary analysis done on the implementation of a Q-learning agent and a shield focused on solving the Grid World environment. In Section 5.1, the 10-by-10 grid was described where an agent has to reach a terminal state at `[8,4]` when starting from `[1,10]`. In the simple most

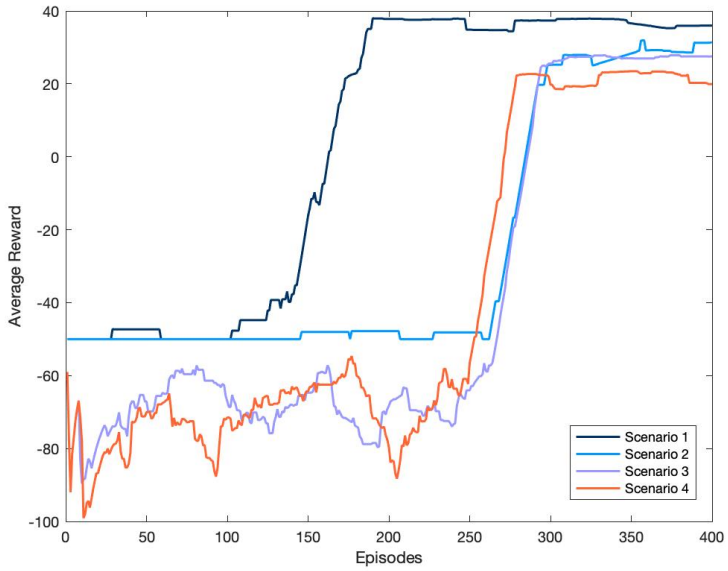


Figure 5.7: Average reward for the four scenarios in the Grid World.

case, *Scenario 1*, the agent has to reach the terminal state without worrying of obstacles or else. In Section 5.2, the Q-learning agent and its characteristics were described. These mainly follow from the analysis carried out from Section 2.2.4. The shielding approach was described in Section 5.3. This is a simplified version of the shielding method described by Alshiekh et al. (2018). Obstacles are introduced to remove the possibility to visit unsafe states. This is the case of a simplified preemptive shield and is labeled as *Scenario 2*. Adjacent to the obstacle states, some unsafe states that the agent is able to visit have been implemented. If the agent does take an action that leads to such state, a penalty is given. This allows the agent to learn not to visit these states rather than completely removing them from the environment. This is defined as *Scenario 3*. Finally, the last and fourth scenario is created where certain states are equipped with a ranking of actions that may be taken. If the agent reaches one of these states, the actions that can be performed to reach a successive state are ranked from most to least desirable. This allows the agent to learn a safe policy in a faster manner.

It was found that all scenarios reach the target state within the 400 episodes. *Scenario 1* converges the faster and with the highest average reward. This follows the environment, since no penalty are awarded and the agent does not need to learn a safe policy. *Scenarios 2* and *3* show a similar behavior, however the latter has a much more negative average reward up until it starts converging. This is due to the penalties given for exploring unsafe states. Although unsafe states may not be part of the final optimal policy (which is the case for the third scenario in this analysis), they may still be visited and be part of the policy. This is a result of the exploratory behavior of the agent which is an integral part of reinforcement learning. Finally, although *Scenario 4* shows a very similar trend as *Scenario 3*, it reaches convergence slightly before. This is mostly due to the rank of actions which allow the agent to skip the exploration of certain areas of the grid.

In the end, shielding does provide a final policy which generally avoids unsafe states although the agent may have to deal with the presence of the latter in the policy. This preliminary analysis provided interesting knowledge on how even a simplified version of shielding can be used to enhance safety in a straightforward environment such as grid world. Although shielding requires some external knowledge, it can be argued that any safe RL technique needs to define what is deemed risky and what not. Therefore, in the following stage of this thesis, a complete version of shielding will be applied to the control system of the Cessna Citation to investigate its performance and whether safety of a learning controller can be enhanced.

Part III

Additional Results

6

Turbulence Study

The Cessna Citation model used in this research is only equipped with longitudinal motion controllers namely a learning flight path controller and a pitch rate controller designed as a PID. Although the lateral states are quasi-null in the nominal conditions, their response may vary when subjected to severe turbulence. Additionally, although the patchy turbulence model implemented in this research describes a symmetric field, the non-zero gust velocities u_g and w_g will still affect the roll and yaw moments (De Prins, 2010). For this reason, it should be investigated whether lateral controllers are needed

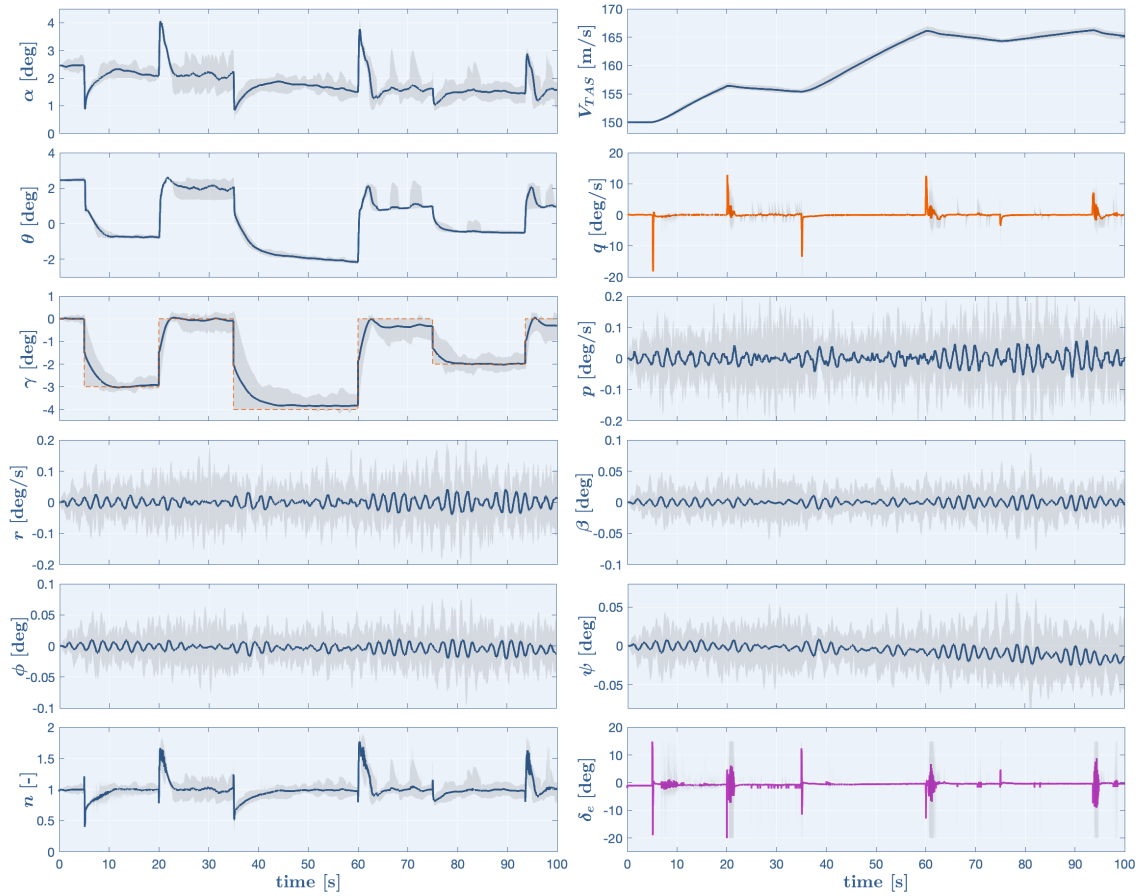


Figure 6.1: Flight path tracking of DDPG agent with shield iterated 15 times. The blue and orange lines represent the mean of the iterated responses and references. The gray areas represent the variation of the responses over the iterated simulation.

when a non-ideal scenario is simulated. The DDPG controller with implemented shield in nominal flight

profile is simulated 15 times in turbulent conditions. The results of the iterative simulation can be seen in Fig. 6.1. The results show no significant discrepancy with respect to the nominal shielded DDPG results in the longitudinal direction. In the lateral and vertical direction, small oscillations can be seen. Both β, ϕ and ψ stay consistently smaller than ± 0.1 degrees while the yawning and rolling moment coefficients remain below ± 0.3 degrees per second. Therefore, the yawning and rolling motions remain small enough for the Cessna Citation 500 simulation used in this research not to need any additional controllers.

7

Robustness Analysis to Different Reference Signals

In this section, a robustness analysis to different reference signals will be carried. Table 7.1 summarizes the tests performed in the sections below and the performance of the controllers for each reference signal. Section 7.1 will analyze the responses of the DDPG and SIP controllers while Section 7.2 will discuss the performance of the shield.

Table 7.1: Robustness analysis to varying reference flight path angle. All simulations were run on the nominal flight conditions.

Reference Signal	Amplitude [deg]	Frequency [Hz]	nMAE%	$\mathcal{S}_{\text{risk}}\%$	$\mathcal{S}_{\text{unsafe}}\%$
DDPG Controller					
Sinusoidal	5	0.05	33.6%	15.7%	1.3%
Sawtooth	3	0.05	30.9%	1.6%	5.9%
SIP Controller					
Sinusoidal	5	0.05	55.1%	6.0%	0.6%
Sawtooth	3	0.05	71.3%	6.0%	0.6%
DDPG Controller with Shield					
Sinusoidal	5	0.05	37.2%	10.5%	0%
Sawtooth	3	0.05	47.9%	7.3%	0%

7.1. Robustness Analysis of DDPG and SIP Controllers

The performance of the DDPG and SIP controllers to a sinusoidal and sawtooth reference γ will be presented in Section 7.1.1 and Section 7.1.2 respectively.

7.1.1. Sinusoidal Reference Flight Path Angle

To assess the DDPG and SIP models' robustness to different γ_{ref}, the response of the agents to a sinusoidal reference is tested. The wave has a frequency of 0.05 Hz and amplitude of ± 5 degrees. Fig. 7.1 shows the responses for the DDPG and SIP controller in blue and lime green respectively. As it can be seen, both controllers are able to track the sinusoidal reference although the SIP controller is not able to reach the same amplitude as the γ_{ref}. The conservative policy of the SIP does not support steep changes in pitch as many of the responses of with this controller are characterized by softer tracking. It can be seen that the SIP agent struggles with most of the pitch up maneuvers as the reference pitching rate is extremely noisy during those times. Consequently, the elevator deflection suffers long, high-frequency disturbances that translate to the load factor. In terms of safety, the SIP controller only remains 0.6% of simulation time in $\mathcal{S}_{\text{unsafe}}$ which is less than half than the DDPG controller. From Fig. 7.1 it is clear that the amplitude of the α and n responses is lower for the SIP than the DDPG. Although this is favorable in terms of safety, it is not for the controller performance. It is clear that in terms of performance the DDPG controller is superior, as it is able to track the sinusoidal signal without introducing significant disturbances in the responses. The nMAE% for the two agent reflects the

responses, with the DDPG and SIP controller nMAE% accounting to 33.6% and 55.1% respectively. The DDPG controller can be considered successfully robust to this reference signal. The SIP shows a high error and many undesirable disturbances; however, since tracking accuracy represents one third of the reward function and the agent successfully respects the safety limits imposed, the controller can be considered robust to this γ_{ref}.

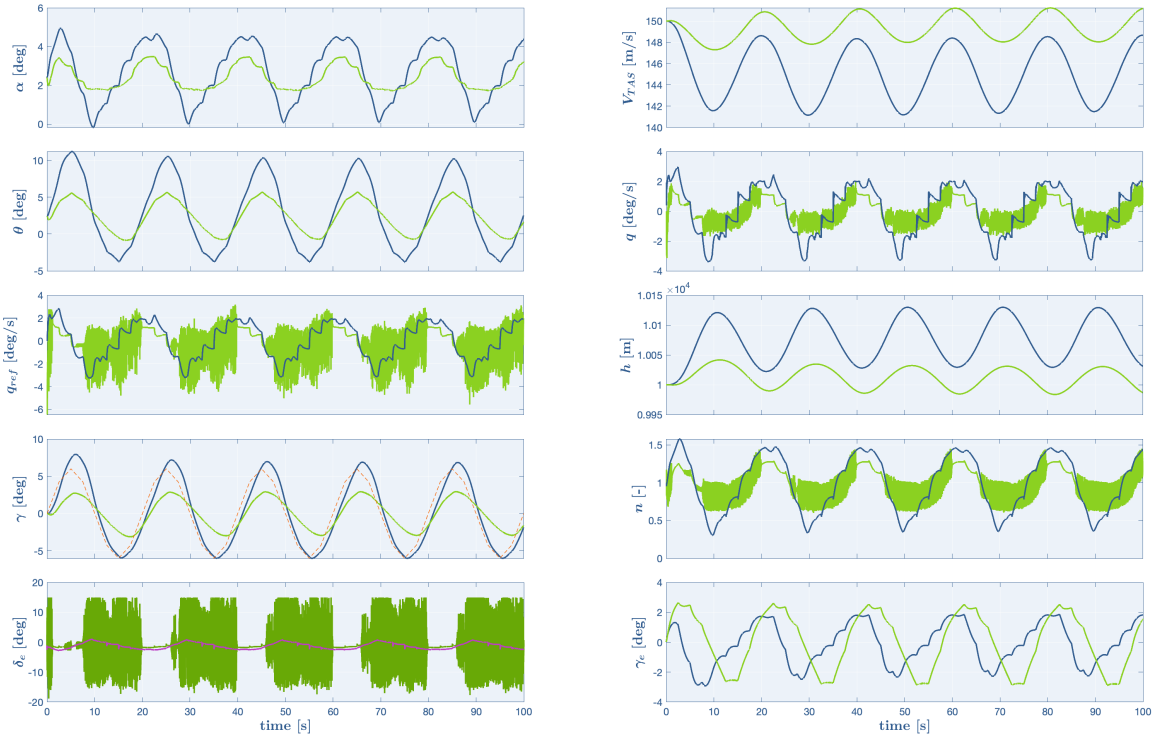


Figure 7.1: Flight path tracking with DDPG controller (in blue) and SIP controller (in lime green) for a reference flight path angle as a sine wave with frequency of 0.05 Hz and amplitude of ± 5 degrees. The orange dashed lines represent the reference signals.

7.1.2. Sawtooth Reference Flight Path Angle

Another γ_{ref} tested is shown in Fig. 7.2. The sawtooth reference signal has an amplitude of ± 3 degrees and a frequency of 0.05 Hertz. In terms of performance, both controllers perform similarly to what discussed in Section 7.1.1. The DDPG follows the γ_{ref} more aggressively and is able to maintain the tracking error low throughout the maneuver. The SIP controller is not fully able to follow the sharp flight path angle and hence resorts to a smoother, softer tracking. The reference pitching rate, the elevator deflection and the load factor are noisy during pitch up sections of the simulation, possibly due to the lower altitude reducing the elevator effectiveness. However, the SIP performs well in terms of safety as both α and n are maintained within the same limits. The behaviour of the SIP to this reference signal is analogous to the one discussed above. The controller remains only 0.6% of times in $\mathcal{S}_{\text{unsafe}}$, which is 5.3% less than the DDPG controller. The SIP controller remains more time in $\mathcal{S}_{\text{risk}_H}$ compared to the DDPG. This can be seen from Fig. 7.2 in the load factor response as it is close to 0.5 once the aircraft pitches down. In terms of performance, Table 7.1 shows that the DDPG's nMAE% is 30.9% and the SIP's is 71.31%. The conclusion about robustness is the same drawn in Section 7.1.1. The DDPG is clearly robust to the sawtooth γ_{ref}; the SIP would normally not be considered robust due to its high tracking error. However, due to the nature of the controller, it can be considered robust as it achieves the goals set by the designer in terms of safety.

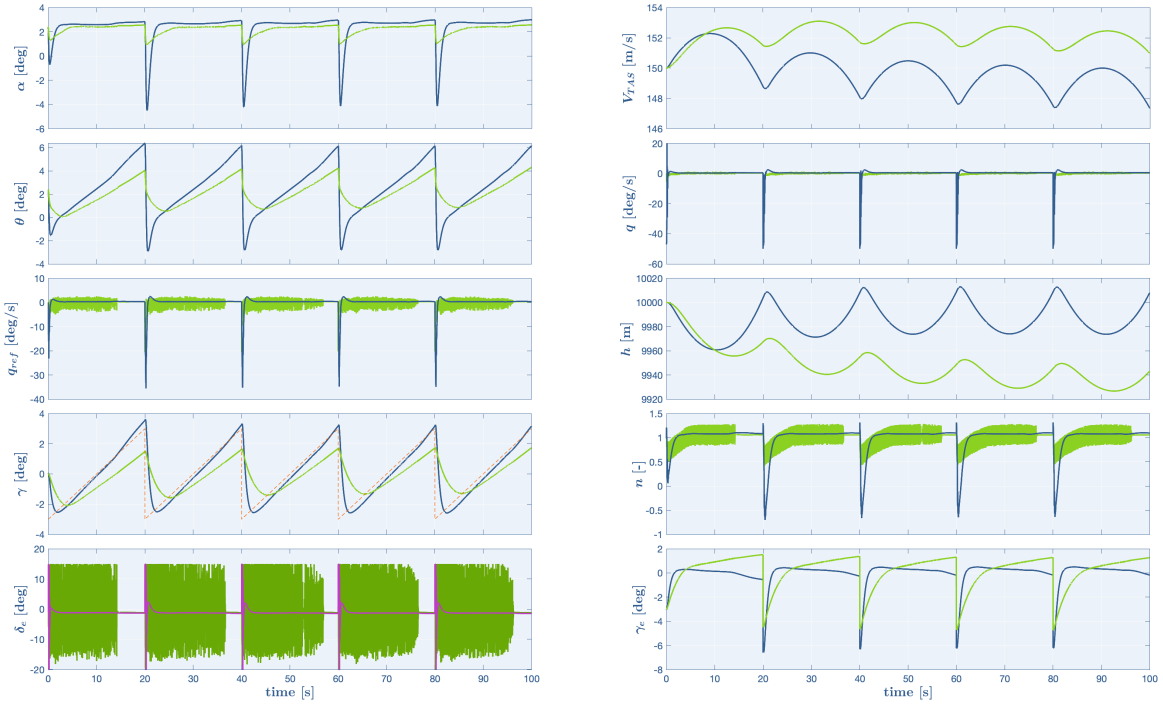


Figure 7.2: Flight path tracking of DDPG agent with shield with reference flight path angle as a sawtooth wave with frequency of 0.01 Hertz and amplitude of ± 5 degrees. The blue and orange lines represent the mean of the iterated responses and references. The gray areas represent the variation of the responses over the iterated simulation.

7.2. Robustness Analysis of Shielded DDPG Controller

After assessing the performance of the DDPG and SIP controllers, the behaviour of the shielded DDPG controller will be analyses. The response to a sinusoidal reference will be discussed in Section 7.2.1 while the response to a sawtooth signal will be presented in Section 7.2.2.

7.2.1. Sinusoidal Reference Flight Path Angle

The shielded controller is tested on a sinusoidal reference γ with frequency of 0.05 Hertz and amplitude of 5 degrees. The responses can be seen in Fig. 7.3. The shield is on periodically during pitch down sections of the maneuver as it is triggered first by M_{SR_n} and following by M_{SR_α}. This allows both angle of attack and load factors to remain within the safe limits prescribed by the safety range model. It can be seen on Table 4 in Part I that the shield allows the agent to remain outside of $\mathcal{S}_{\text{unsafe}}$ for the duration of the simulation. In order to track the signal efficiently without violating safety constraints, the agent stays for 10.5% of times in $\mathcal{S}_{\text{risk}_H}$. This however, reduces the tracking accuracy of the controller as the SIP actions are used, scoring 37.2% in nMAE%, a value slightly higher than its unshielded counterpart. The oscillation between q_{ref} and $k \cdot q_{\text{ref}}$ in theM_{SR_α} introduce some disturbances in the reference pitch rate and consequentially in the elevator deflection and load factor. However, these are small enough and do not particularly effect the tracking of γ_{ref}. Therefore, thanks to the shield, the agent is maintained safe and the controller is able to track the new reference successfully.

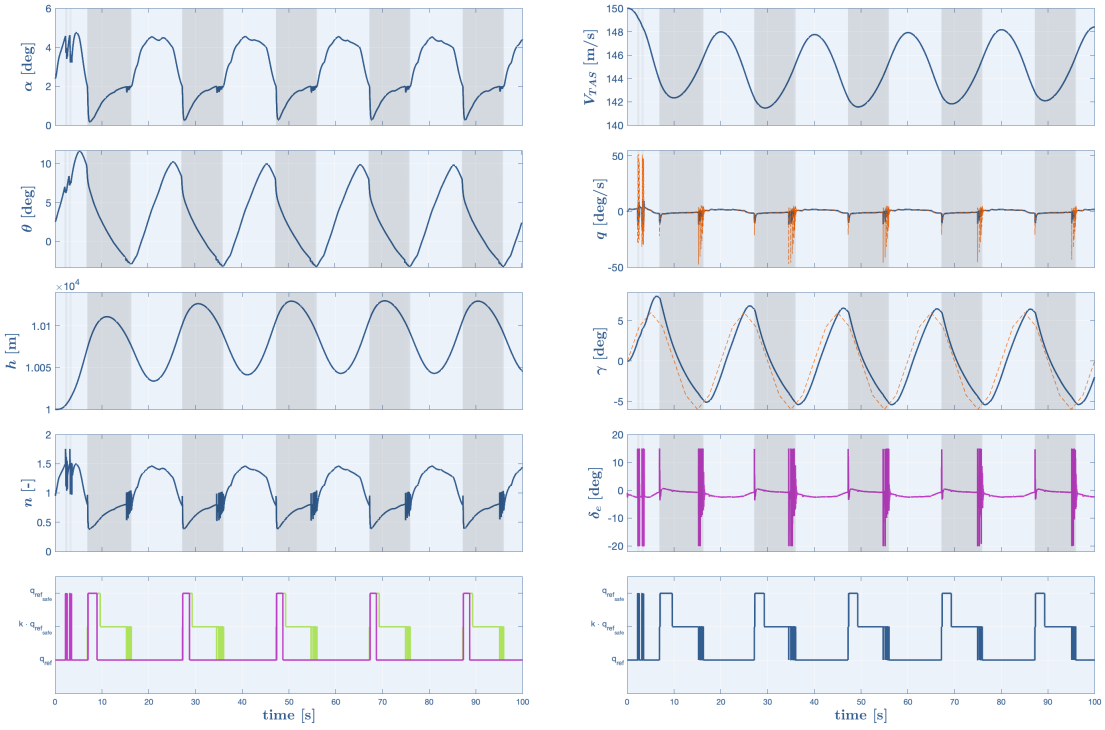


Figure 7.3: Flight path tracking of DDPG agent with shield with reference flight path angle as a sine wave with frequency of 0.05 Hertz and amplitude of ± 5 degrees. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

7.2.2. Sawtooth Reference Flight Path Angle

This reference is chosen as the vertical drop in γ_{ref} is a good candidate for triggering both M_{SR_α} and M_{SR_n} and assessing the shield's robustness. Unsurprisingly, the shield is triggered immediately once the γ_{ref} changes sign. Thanks to the activation of the shield, both the angle of attack and the load factors are kept within the safe limits as $\mathcal{S}_{unsafe}$ remains zero. From Fig. 7.2 it can be seen that the shield allows α to remain above 0 degrees, differently than the unshielded controller where α reaches -4.1 degrees. The load factor is also maintained above 0.35, while the DDPG controller with no shield

pushes n as low as -0.7. With a value of 47.9%, the normalized mean absolute error is higher than the unshielded counterpart, but from Fig. 7.4 it can be seen that error is only substantially high once the shield, and hence the SIP controller, is in use. Therefore, as the shield maintains the agent in a safe state space and is able to track the new reference γ, it is considered robust to this new reference.

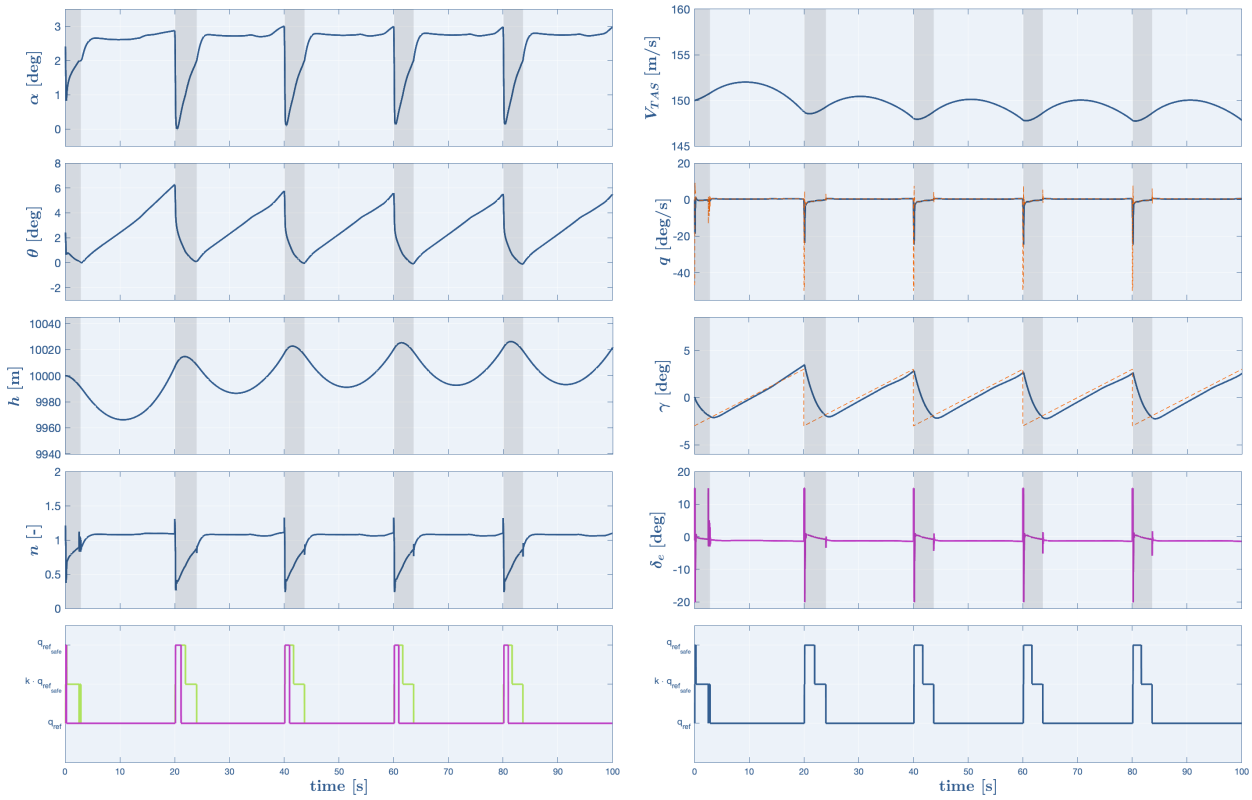


Figure 7.4: Flight path tracking of DDPG agent with shield with reference flight path angle as a sawtooth wave with frequency of 0.05 Hertz and amplitude of ± 5 degrees. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

8

Robustness Analysis to Different Initial Flight Conditions

In this chapter, a robustness analysis for the DDPG, SIP and shielded DDPG controllers to different initial flight conditions will be carried out. Four different flight conditions were investigated and compared to assess the behaviour of the controllers in un-trained situations. An overview of the results can be seen on Table 4 in Part I. Section 8.1 will evaluate the performance of the DDPG and SIP agents while Section 8.2 will discuss the responses of the shielded DDPG controller.

8.1. Robustness Analysis of DDPG and SIP Controllers

In the upcoming sections, the response of the DDPG and SIP controller to four different initial flight conditions will be discussed.

8.1.1. IFC 1

The first flight condition to be investigated is one with the same V_{TAS} is the same as the nominal case but with slightly lower altitude of 7000 meters. Fig. 8.1 shows the responses for the DDPG controller (in blue) and the control inputs(in purple) together with the responses of the SIP controller (in lime green) and its respective control inputs (in dark green). Both controllers exhibit satisfactory tracking performance. The DDPG controller's γ response shows a more pronounced steady state error, especially within the first 5 seconds. However, in an attempt to maintain both the angle of attack and the load factor within safety limits, the SIP controller follows γ_{ref} avoiding a quick pitch down movement. It can be seen from the γ_{e} that the SIP controller has a more pronounced error for this reason.

In terms of safety, it is clear that the DDPG agent has no notion of safety as it greatly exceeds the limits set my the M_{SR} and remains in $\mathcal{S}_{\text{unsafe}}$ for 3.9% of the simulation. The SIP on the contrary, is able to maintain safe operation throughout the same maneuver, shown by $\mathcal{S}_{\text{unsafe}}$ being 0%. With a nMAE% of 29.3% and 34.3% for the DDPG and SIP controllers respectively, both agents are considered to be robust to the IFC 1.

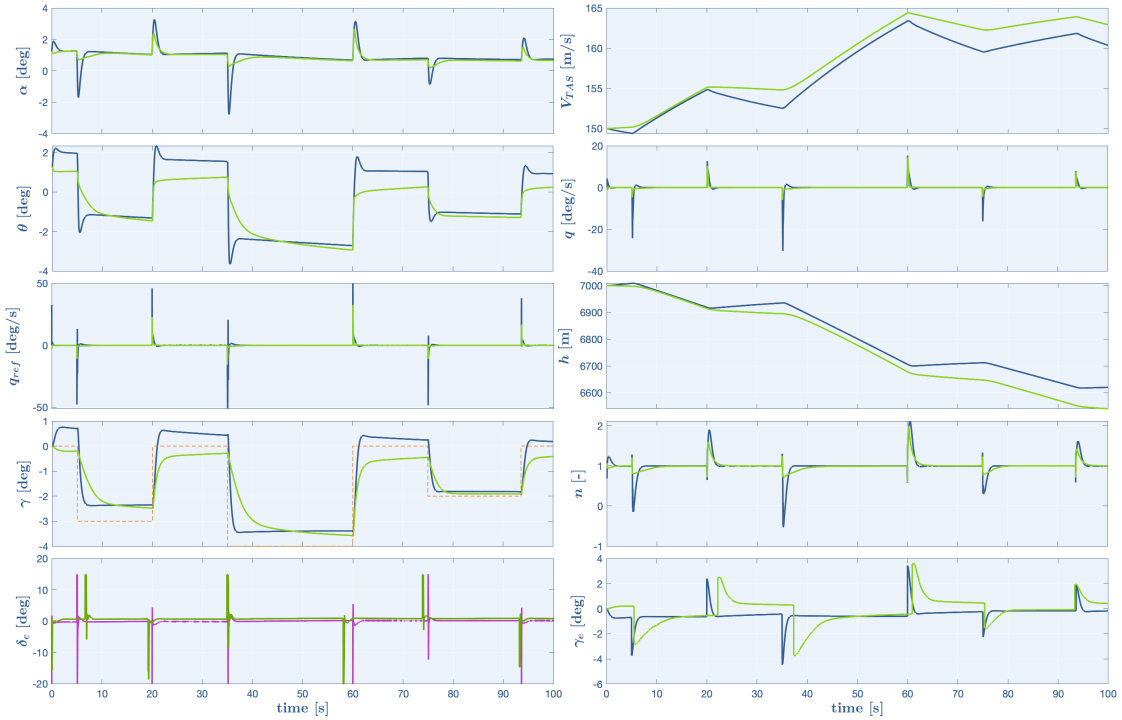


Figure 8.1: Flight path tracking with DDPG controller (in blue) and SIP controller (in lime green) in IFC 1. The control inputs of the DDPG controller are shown in purple while the control inputs of the SIP in dark green.

8.1.2. IFC 2

The second flight condition to be examined is with the same altitude as IFC 1 but with lower velocity V_{TAS}. The responses for this case can be seen in Fig. 8.2. The tracking of the flight path angle begins for both controller with a large steady state error the first 20 seconds, before stabilizing to a smaller error, especially for the DDPG controller. The lower velocity will decrease the dynamic pressure which will in turn decrease the effectiveness of the control surfaces. This occurs as a larger deflection will be needed to generate the same pitching moment at lower speed. From Fig. 8.2 it can be seen that the elevator requires larger deflections than in the IFC 1 case discussed in Section 8.1.1. Although the altitude is the same as IFC 1, the decreased velocity requires a larger angle of attack for the same maneuver. This allows the SIP agent to maintain the angle of attack well above the limits as well as the load factor. The SIP agent never visits $\mathcal{S}_{\text{unsafe}}$.

The same could be said for the DDPG controller, however n reaches 0 at $t = 35s$. This aligns with the -4 degrees γ dive where the DDPG controller generates a strong reference pitch signal. The elevator deflection has a similar behaviour as for the elevator in Fig. 8.1 although the deflection has to be applied to a longer period of time to allow for good enough tracking. The agent remains in $\mathcal{S}_{\text{unsafe}}$ throughout 1.2% of the simulation. Overall, although both controllers performance is subpar with nMAE% of 40.0% and 45.1% for the DDPG and SIP models respectively, both normalized mean absolute errors are below 50% hence the agents can be considered robust to the IFC 2.

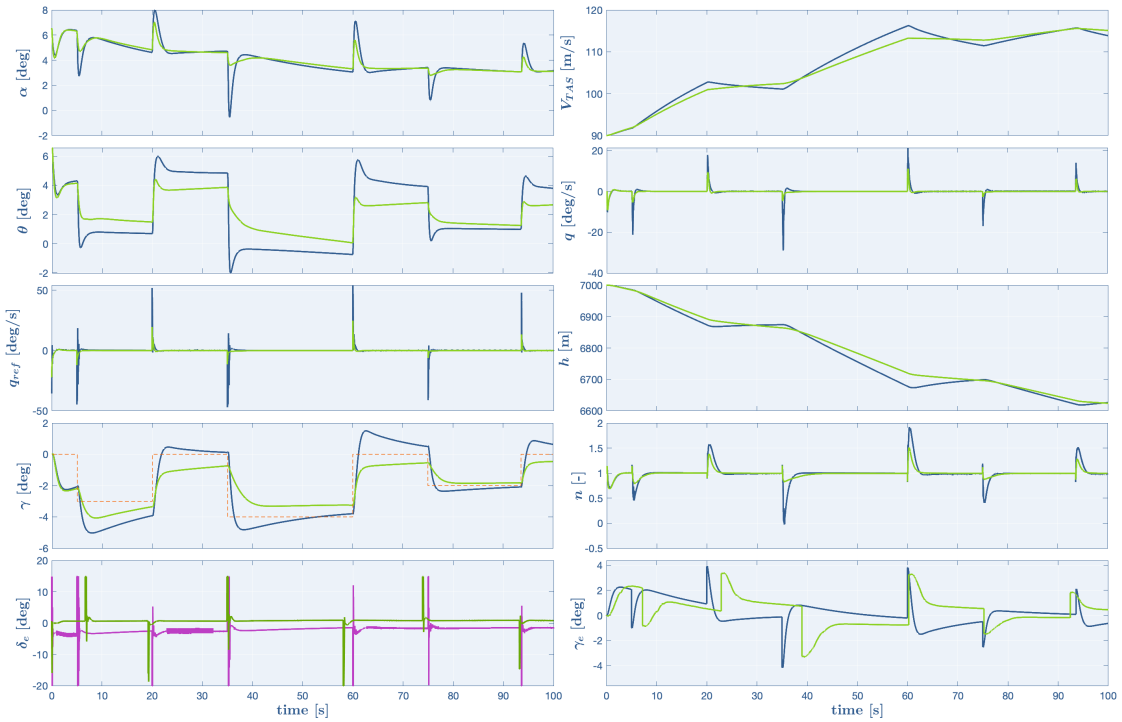


Figure 8.2: Flight path tracking with DDPG controller (in blue) and SIP controller (in lime green) in IFC 2. The control inputs of the DDPG controller are shown in purple while the control inputs of the SIP in dark green.

8.1.3. IFC 3

The third IFC is lower in altitude than IFC 1 and IFC 2 but has the same velocity as the nominal case. In this research, a conventional step down approach at top of descent is simulated. An altitude 8000 meters lower than the norm was chosen to ensure that the agent could effectively track the reference flight path angle during different stages of approach. The results for the simulation are shown in Fig. 8.3. The γ response for both DDPG and SIP controller exhibits a steady state error of almost 1 degree up until $t = 35s$. After that, the error begins to decrease, but still remains noticeable throughout the maneuver.

In terms of safety, the SIP shows some flaws in design. Similarly to the flight case covered in Section 8.1.2, higher velocity requires a substantially smaller angle of attack for an identical pitching maneuver. Due to this, the angle ot attack in this flight condition reaches values lower than the safe limit set at 0 degrees. Although the SIP agent is taught to keep away from particular values of α, the reward function also acknowledges accurate γ_{ref} tracking. Therefore, the SIP is not able to successfully solve the trade-off between safety and performance with respect to α. However, since the SIP agent knows to retain an α between 0 degrees and 12 degrees, the agent will try to fly the aircraft with an angle of attack as close as possible to the aforementioned range. When looking at the load factor however, it can be seen that the SIP agent outperforms the DDPG by keeping n within safety limits.

In terms of tracking performance, both controller perform worst than in the nominal condition. With a nMAE% of 41.4% for the DDPG controller and 35.9% for the SIP, the performance in these flight conditions is not exemplary. Both controllers show the worst behaviour when looking at the simulation time spent in $\mathcal{S}_{\text{unsafe}}$. Both agents visit $\mathcal{S}_{\text{unsafe}}$ for more than 60% of the simulation time. However, these controllers are still considered robust to IFC 3 as both nMAE% are below 50% and the overall deviation from the safe limits is considered small.

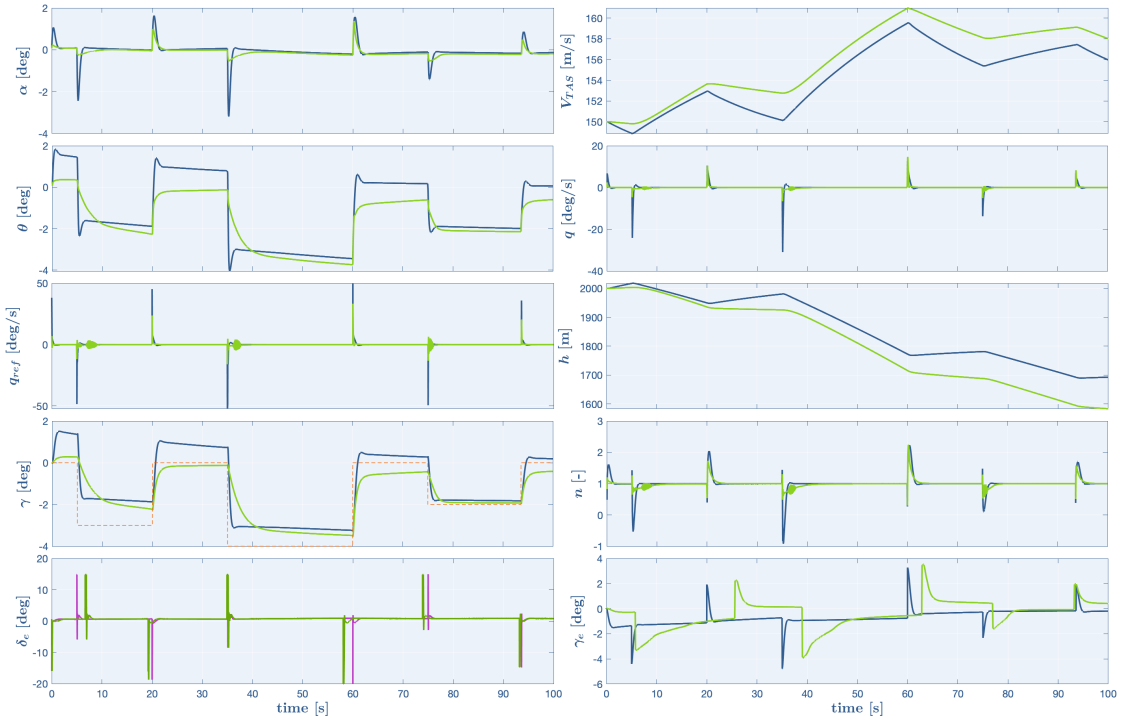


Figure 8.3: Flight path tracking with DDPG controller (in blue) and SIP controller (in lime green) in IFC 3. The control inputs of the DDPG controller are shown in purple while the control inputs of the SIP in dark green.

8.1.4. IFC 4

The fourth initial flight conditions has the lowest altitude and velocity. The responses of the controller to the IFC 4 can be seen in Fig. 8.4. The tracking of the γ_{ref} is performed by the DDPG agent quite well as the error is maintained low and the controller is able to follow the reference throughout the step. The shield controller is also able to track γ_{ref} although a larger error can be seen.

This softer maneuver allows the safety of α and n to remain within limits as the agent never visits $\mathcal{S}_{\text{unsafe}}$. To follow γ_{ref}, the elevator is required to deflect more and for longer: this is to be expected as with a decrease of both velocity and altitude comes a decrease in dynamic pressure. A reduction in the latter results in lower control effectiveness. As already mentioned, both α and n remain in a safe state space with the SIP controller. The same can not be said for the responses of the DDPG, where α reaches a minimum of -1.2 degrees and n of -0.3. The agent is in an unsafe state space for 1.5% of the simulation time. Overall, both controllers perform better than in the nominal conditions with a nMAE% of 15.7% and 33.3% for the DDPG and SIP controllers respectively. Therefore, the controllers in IFC 4 can be considered robust.

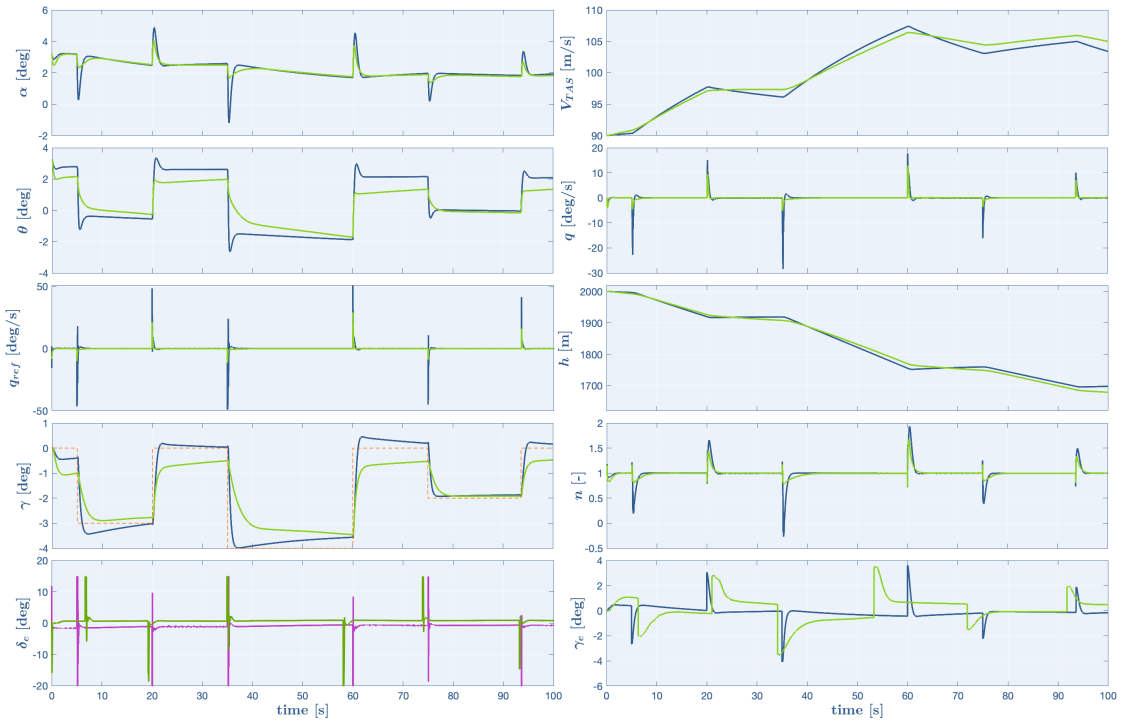


Figure 8.4: Flight path tracking with DDPG controller (in blue) and SIP controller (in lime green) in IFC 4. The control inputs of the DDPG controller are shown in purple while the control inputs of the SIP in dark green.

8.2. Robustness Analysis of Shielded DDPG Controller

The performance of the shielded DDPG controller to four different initial flight conditions will be evaluated in the sections below.

8.2.1. IFC 1

The first IFC case corresponds to the same V_{TAS} as the nominal flight conditions but with decreased altitude. From Fig. 8.5, it can be seen that the shield is on for 98% of the maneuver. This is mostly due to the angle of attack: as the altitude decrease, the air density increases meaning that a smaller angle of attack is needed for the same maneuver. However, a too low α will trigger the shield according to the M_{SR}. This flight condition will therefore be mostly dictated by the action proposed by the SIP agent as it can be checked with the lower figures in Fig. 8.5. With the lower altitude, the higher density improves the aileron effectiveness making the responses overall stable.

Both the angle of attack and the load factor are maintained with safety limits and the agent never visits $\mathcal{S}_{\text{unsafe}}$. Although α is close to the lower bound defined by M_{SR_α}, the controller is able to maintain α above zero degrees. The load factor is kept below 2 and above 0.6, and therefore never reaching an unsafe state space. Therefore, the shield performs successfully in terms of safety with altered initial flight conditions.

From Table 4 in Part I, it can be seen that the nMAE% for this flight condition stands at 34.4%. The result is higher than the nMAE% for the analogous unshielded case and 0.1% higher than the SIP controlled case. This is mostly due to the disturbance at $t = 20s$ where the M_{SR} oscillates between different actions. However, the system's response shows an overall low error and is considered stable when the shield is implemented. Therefore, the shield is robust to IFC 1.

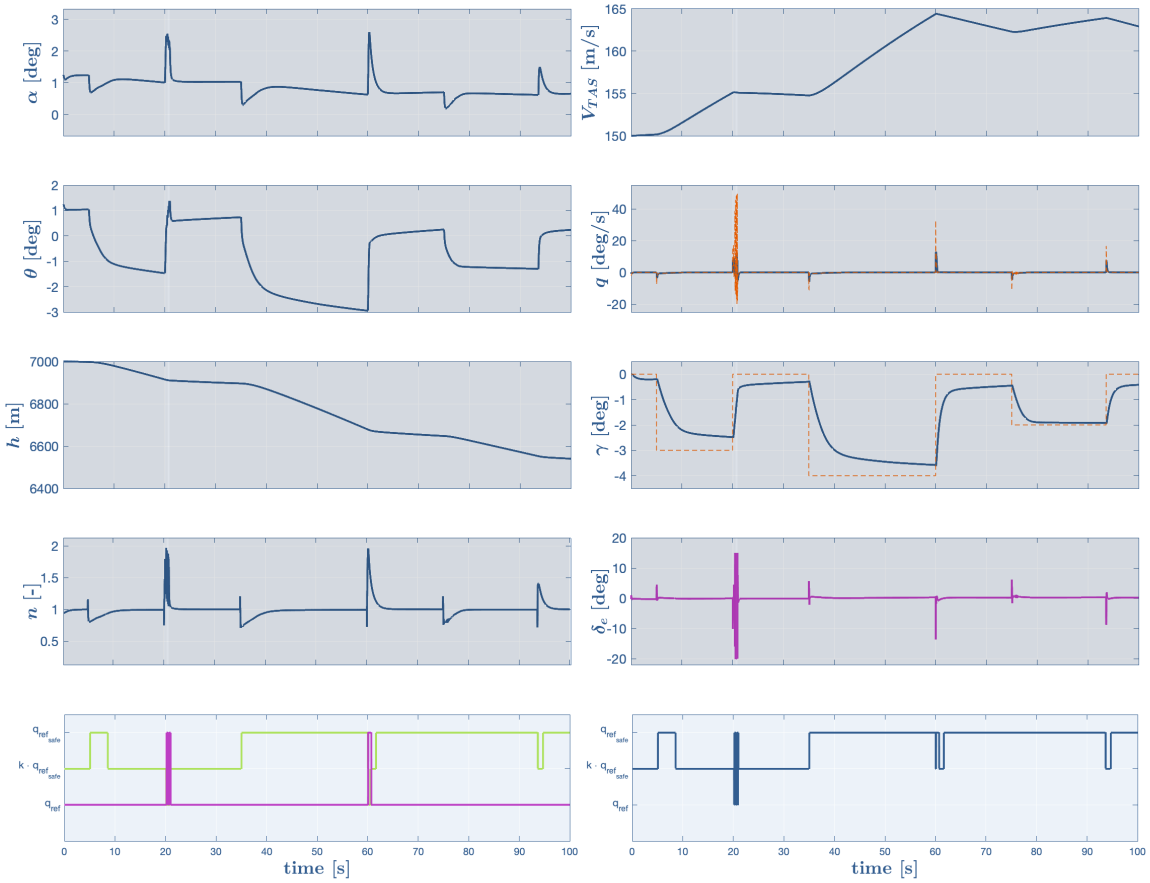


Figure 8.5: Flight path tracking with shielded DDPG controller in IFC 1. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

8.2.2. IFC 2

The second IFC case is with slightly decreased altitude and decreased V_{TAS} with respect to the nominal case. The responses for this case can be seen in Fig. 8.6. Although the altitude is the same as IFC 1, the decreased velocity requires a larger angle of attack for the same maneuver. Therefore, the shield is only briefly on when the aircraft steps down for the required γ_{ref} meaning most of the simulation is controlled by the DDPG agent. The lower velocity will decrease the dynamic pressure which will in turn decrease the effectiveness of the control surfaces. This occurs as a larger deflection will be needed to generate the same pitching moment at lower speed. From Fig. 8.6 it can be seen that the elevator requires larger deflections than in the IFC 1 case discussed in Section 8.2.1. The elevator introduces some disturbances, most clearly seen in q_{ref}; however, the response is overall stable.

In terms of safety, the M_{SR} is mostly activated due to the load factor. By activating the shield, the load factor remains at safe values with maximum at 1.8 and minimum and 0.4. For the short amounts the M_{SR_α} is on, the angle of attack is kept within the safe lower limits defined in the safety range model while the upper limits do not reach an unsafe state space. Therefore, the agent never visits $\mathcal{S}_{\text{unsafe}}$.

The nMAE% for IFC 2 is the third highest in this research among the shielded controller results, reaching 40.7%. It is clear that the error is mostly concentrated in the first 20 seconds of the simulation and slowly decreases as the simulation goes on. This result is coherent with the nMAE% for the same IFC when the DDPG and SIP agents are tested. Although the controller's performance is lacking, the flight path angle is still appropriately tracked and the shield correctly avoids unsafe state spaces.

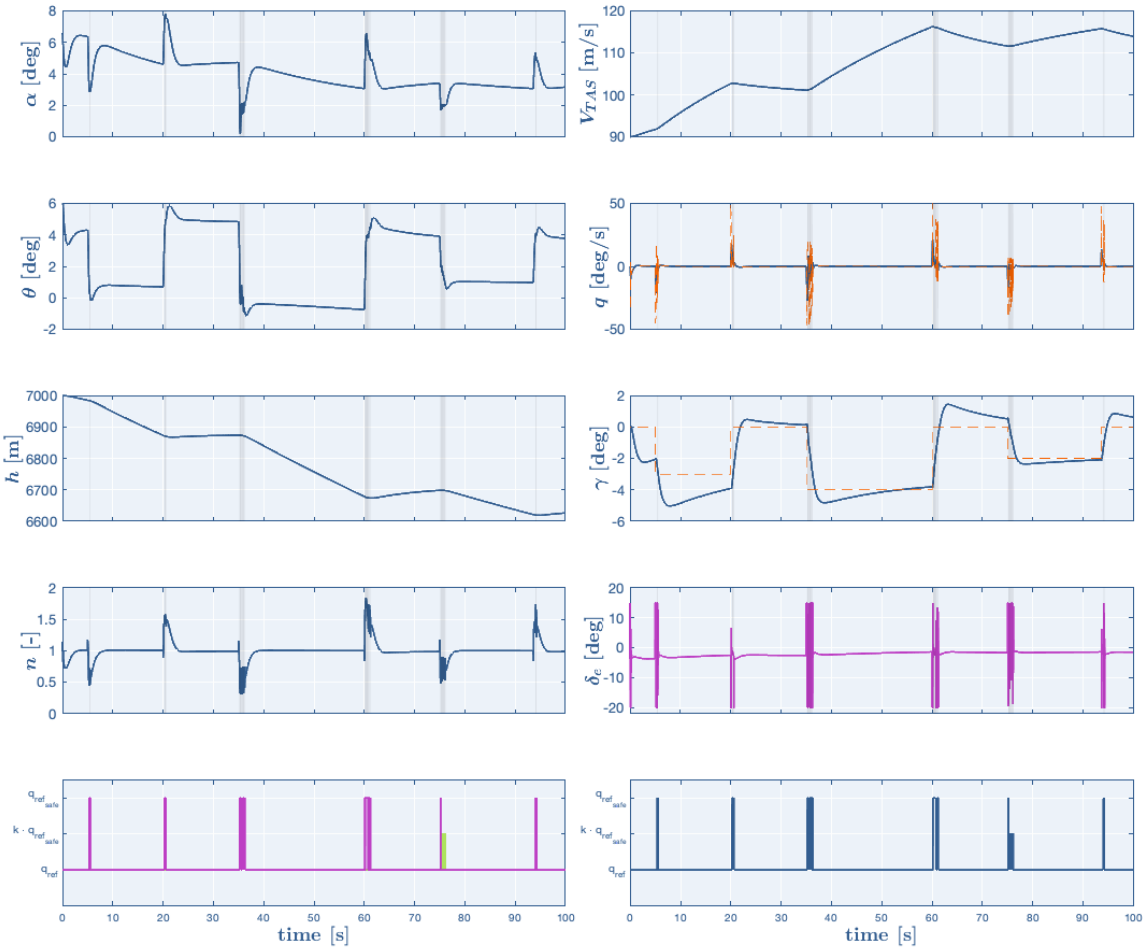


Figure 8.6: Flight path tracking with shielded DDPG controller in IFC 2. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

8.2.3. IFC 3

The third IFC case has the same velocity as the nominal case but an altitude lower than IFC 1 and 2. The results are shown in Fig. 8.7. This flight condition has been chosen due to the maneuver investigated in this research. The nominal case demonstrates a conventional step down approach at top of descent. To make sure the agent would be able to successfully track the reference flight path angle throughout the descent, an altitude 8000 meters lower than the nominal was chosen.

In this case, the shield is on 100% of the time. Similarly to IFC 1 discussed in Section 8.2.1, the angle of attack is significantly lower for higher speed when the same pitching maneuver is required. Here however, the α is mostly lower than the safe limit of 0 degrees. In this case, although the shield is always on, the controller is not able to maintain the angle of attack in a safe state space. This has mostly to do with the design of the SIP controller. Although the SIP agent is instructed to avoid certain values of α and n, the reward function also awards correct tracking of the γ_{ref}. In this case, the flight conditions lead the α to drop, and unable to avoid this, the SIP keeps controlling the aircraft so that γ_{ref} can be tracked and n can be kept within the safe limits. It should be noted that even if α reaches unsafe limits, the agent maintains a small negative angle, as during training it receives a penalty for negative angles of attack. This drawback is directly linked the trade-off between safety and tracking performance embedded the reward function of the agent. If this case highlights one inherent issue with the design of the SIP, it also showcases the flexibility of the M_{SR}. By adjusting the safety range model

limits post-learning, the shield can be modified for different flight conditions so that the DDPG agent can control the aircraft until the SIP is needed, triggered by a new M_{SR}. Since the SIP agent knows to retain an α between 0 degrees and 12 degrees, the SIP agent will try to fly the aircraft with an angle of attack as close as possible to the aforementioned range. Overall, the agent is in an unsafe state space 85.8% of times,the highest across all controllers. The controller is still able to track γ_{ref}, however the

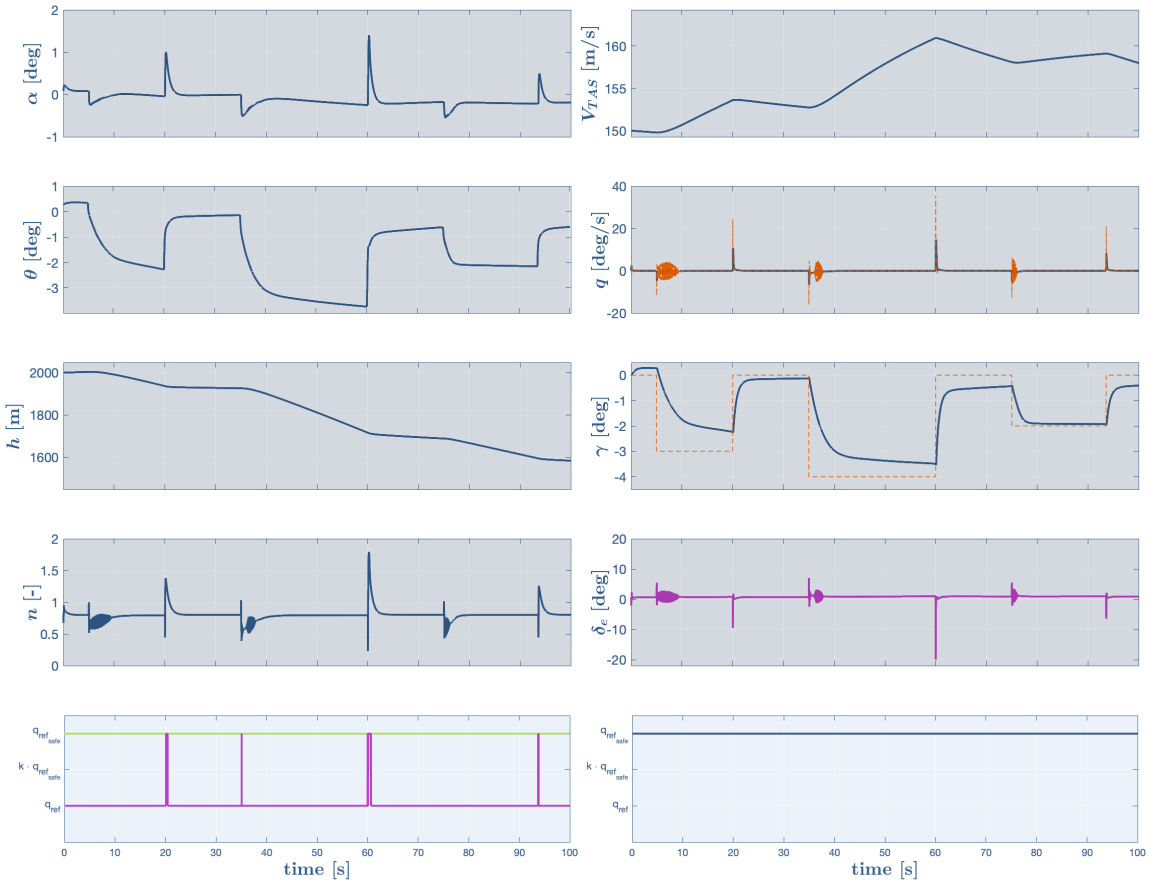


Figure 8.7: Flight path tracking with shielded DDPG controller in IFC 3. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

nMAE% is 35.9%. The nMAE% unsurprisingly matches the nMAE% of the SIP for the same IFC as the shielded case utilizes only $q_{\text{ref}_{\text{safe}}}$. As shown on Table 4 in Part I, the nMAE% for the unshielded case is equal to 41.4%, the second overall highest. Although the performance of the shielded case is worsened by the sole use of the SIP, it is clear that the low altitude, high speed combination deteriorates the performance in all controllers presented on Table 4 in Part I. All and all, in terms of tracking performance, the controller can be considered robust in these flight conditions.

8.2.4. IFC 4

The fourth IFC has an altitude of 2000 meters and V_{TAS} of 90 meters per second. In this case, the shield is on 51% of the time as shown in Fig. 8.8. Similarly to Section 8.2.2, the lower velocity allows for a higher angle of attack with respect to the α shown in Fig. 8.7 for the same maneuver.

Although the α is higher than IFC 3 for the same altitude, at $t = 50s$ it begins to lower, triggering the M_{SR_α}. The switch between q_{ref} and $q_{\text{ref}_{\text{safe}}}$ can be clearly seen at $t = 53s$. The shield allows the angle of attack to remain within safe limits with a minimum value of 0.8 degrees and a maximum of 4.5 degrees. The M_{SR_n} is also turned on at almost every step up or down of γ_{ref}. The load factor is

also kept within the safe limits of defined in the safety range model. The maximum value is found at 1.9 while the minimum is 0.4. Therefore, the shield never lets the agent visit $\mathcal{S}_{\text{unsafe}}$. Although safety is maintained, the load factor shows some disturbed behaviour. By looking at the shaded areas, it can be seen that this behaviour is mostly caused by the oscillation of the M_{SR_n} between two safe states, hence the action fed to the pitch rate varies at each time step. A similar behaviour has been discussed before. This disturbance can also be easily recognized in the elevator deflection and consequentially in the pitch rate. It should also be noted that, although control effectiveness increases with a decrease of altitude, the lower velocity has a much greater influence on the control surfaces efficiency. Therefore, the responses may suffer from disturbances due to the M_{SR}, but the flight conditions do amplify the noisy behaviour.

In terms of performance, the nMAE% for this simulation is 21.9%. This is higher than the unshielded controller with the same flight conditions and can be related to the shield being deployed half the simulation time. Therefore, as the safety of the agent is well within limits and the tracking performance is satisfactory, the shield can be considered robust within IFC 4.

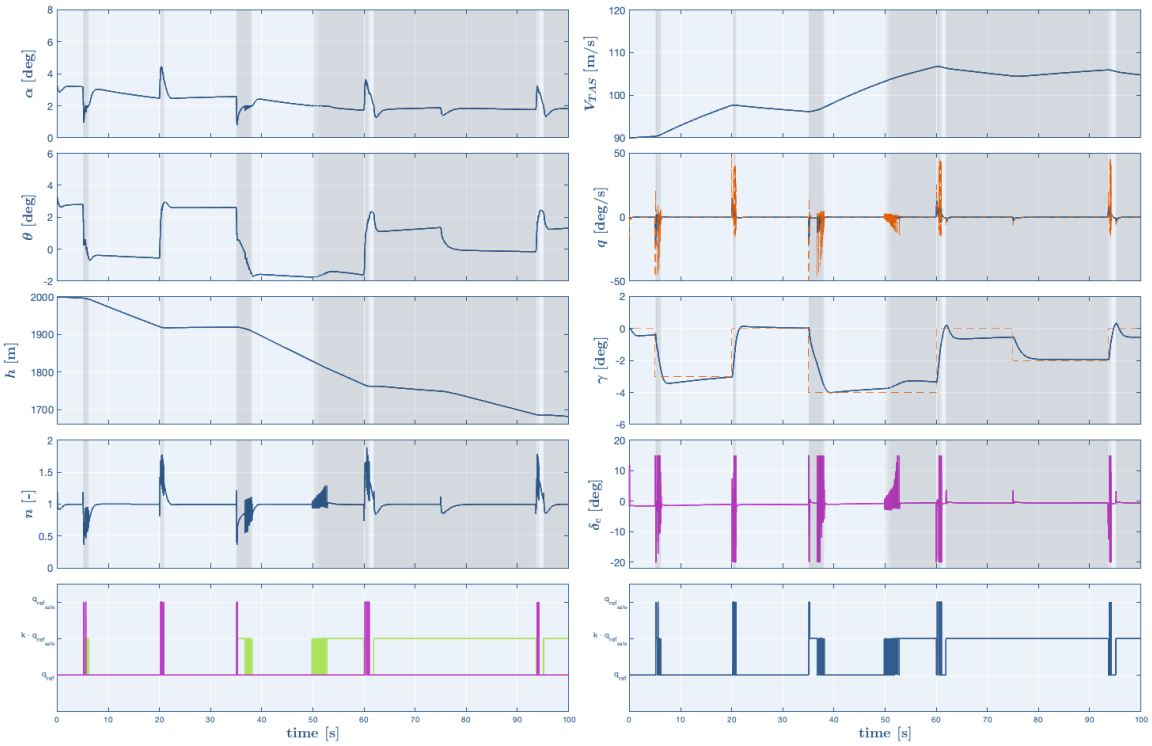


Figure 8.8: Flight path tracking with shielded DDPG controller in IFC 4. The response of the agent is given in blue, the reference signals are shown by the dashed orange lines. The control inputs are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR_α} is also shown in blue. The grey areas represent the times at which the shield is actively on.

9

Verification and Validation

9.1. Verification

In the sections below, the tools used in this research will be verified to endure their correct implementation.

9.1.1. Cessna Citation Model Verification

Delft University of Technology Aircraft Simulation Model and Analysis Tool (DASMAT) is a simulation model and tool developed by Delft University of Technology Van Der Linden (1998). The aim was to delevop a high fidelity model of the PH-LAB, a Cessna Citation 500 used as research aircraft. The model is build on Simulink. The workings of the DASMAT can be verified by performing a sanity check based on general flight dynamics. In Fig. 9.1, the states and control inputs of the model for trimmed nominal flight conditions are given. The aircraft is subjected to a negative elevator deflection and a positive aileron deflection. The applied δ_{e} should result in a positive pitch rate q and in turn increase the pitch angle θ. This phenomenon can be clearly seen in Fig. 9.1 as both q and θ increase once the elevator deflection is decreased from 0° to -5° at $t = 2s$. At $t = 5s$, a positive deflection is applied to the ailerons. This results in a negative roll rate p and a negative bank angle ϕ. Both these effects can be seen in Fig. 9.1. Finally, the zero deflection of the rudder δ_{r} results in a small, quasi-null yaw rate r as well as a small sideslip β. It should be noted that neither of these values are zero as the roll and pitch movements result in small changes in yaw due to coupling effects.

9.1.2. DDPG Model Verification

The DDPG algorithm was reproduced from Lillicrap et al. (2015). The correct workings of the algorithm was checked by analyzing the learning curve of the agent during training. Since the training curve shows a positive learning trend, it can be concluded that the agent correctly learns by interacting with the environment hence the algorithm is accurately implemented.

9.1.3. Shielding Method Verification

The shield can be verified by running a simulation with the intent of triggering the safety range model M_{SR}. In Fig. 9.2, the aircraft is aiming at following a step down of -3 degrees. It can be seen that at around $t = 2s$ both the M_{SR_n} and M_{SR_α} correctly detect the load factor and the angle of attack reaching a risky state space. Right after $t = 2s$, α becomes less than 1 degree before remaining between 1 and 2 degrees up until $t = 6.5s$. After that, it settles at around 2.2 degrees. From the lower-left plot, it is clear that the M_{SR_α} is triggered at $t = 2.1s$ where $q'_{\text{ref}} = q_{\text{ref}_{\text{safe}}}$ before switching to $q'_{\text{ref}} = k \cdot q_{\text{ref}_{\text{safe}}}$ at $t = 2.3s$. At $t = 5.2s$, the M_{SR_α} oscillates as the angle of attack varies between 2 degrees and 1.95 degrees. After some oscillation, the M_{SR_α} finally returns to $q'_{\text{ref}} = q_{\text{ref}}$. As for the M_{SR_n}, the model correctly detects the load factor increasing at $t = 2.1s$ and switches to $q'_{\text{ref}} = q_{\text{ref}_{\text{safe}}}$. After that, the load factor is within the safety limits, hence M_{SR_n} returns to $q'_{\text{ref}} = q_{\text{ref}}$. Finally, from the lower-right, it can be deduced that the M_{SR} correctly chooses the action which represents the *safest option*. Therefore, the shielding method implemented in this research can be considered verified.

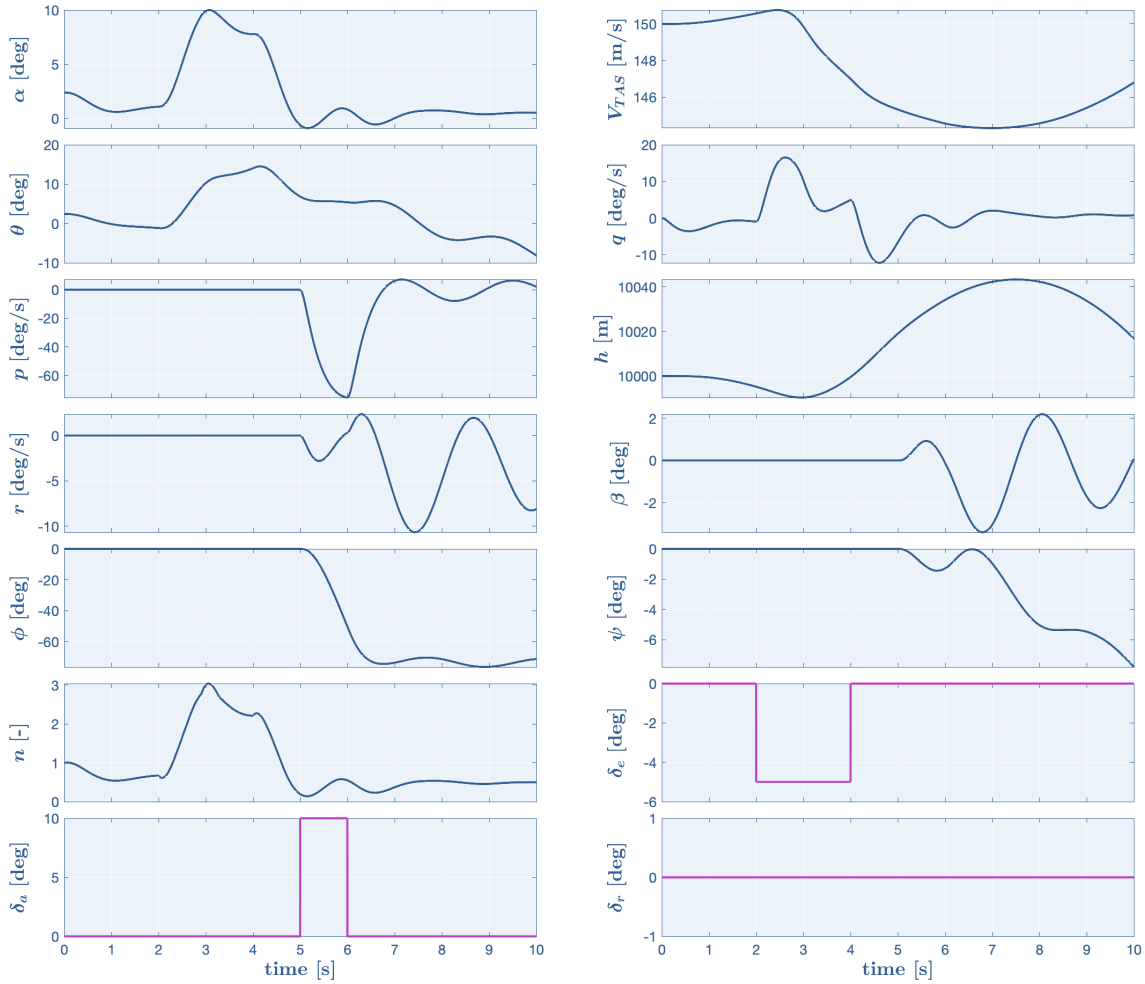


Figure 9.1: DASMAT simulink model response to a elevator and aileron step control. The responses are shown in blue, while the control inputs in purple.

9.2. Validation

The validation of the tools used in this research will be carried below.

9.2.1. Cessna Citation Model Validation

The correct functioning of the DASMAT simulation model can be validated by comparing data acquired by the Cessna Citation 550 PH-LAB research aircraft and data generated by the simulation. Research conducted by Van den Hoek et al. (2018) compares data of the Citation in pre-stall envelope. It was found that the RMSE was respectively 8.38% and 12.65% for the longitudinal force and moment coefficients while the RMSE for the lateral counterparts to be 7.34% and 8.58%. With these values largely lower than 20%, the DASMAT model can be considered a adequate representation of the PH-LAB aircraft.

9.2.2. DDPG Model Validation

The DDPG controller can be validated by determining that its performance, without the implementation of the shield, meets the predicted requirements in tracking accuracy. Table 7.1 gives the nMAE% for different scenarios. The nMAE% was found for both nominal flight conditions, turbulent conditions with added biased sensor noise, different reference signals and initial flight conditions different from the training sample. For all these cases, the nMAE% remained well below 45%. Therefore, the DDPG controller can be considered robust in both different flight conditions as well as when applied to real-world designed systems.

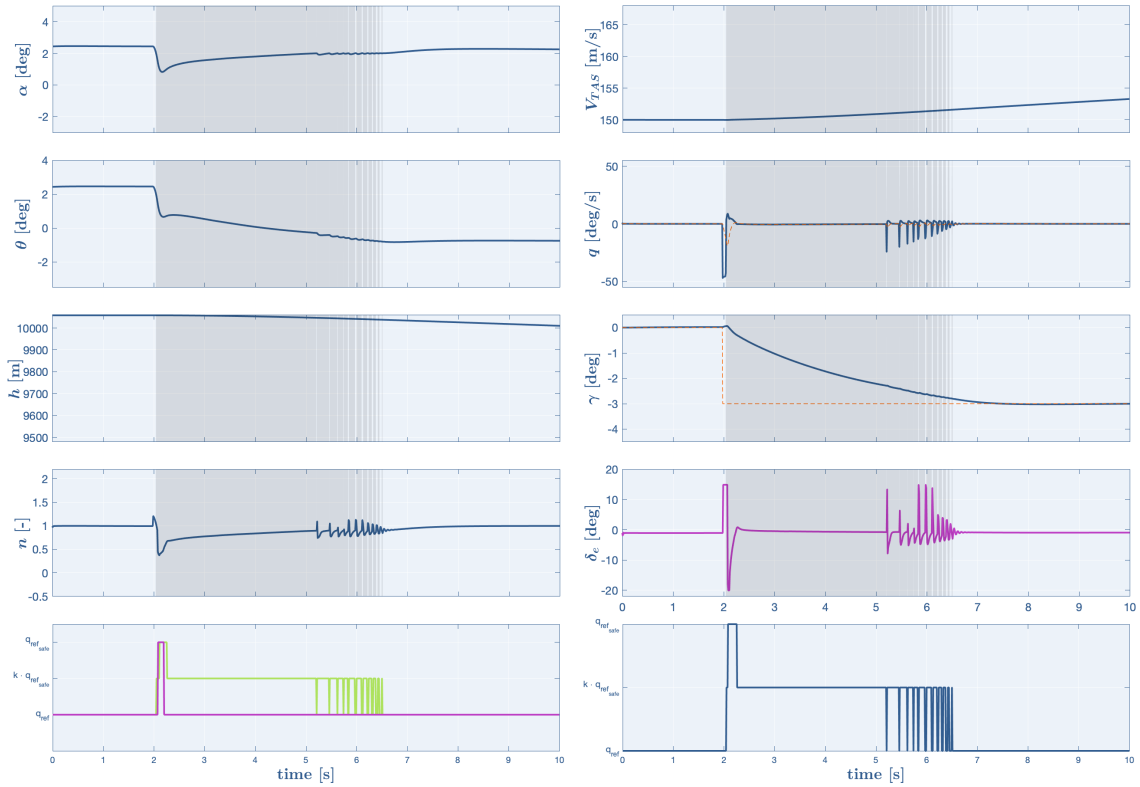


Figure 9.2: Verification of shield model. The aircraft responses are shown in blue, the reference signals in dashed orange lines. The actuators are shown in purple. The M_{SR_α} is shown in lime green while the M_{SR_n} in purple. The final M_{SR} is also shown in blue. The grey areas represent the times at which the shield is actively on.

9.2.3. Shielding Method Validation

The validation of the shielding model has been carried by Zoon (2021). In his research, a shield was implemented in an autonomous vehicle (AV) control system and validated it by using the model in CARLA, a state-of-the-art driving simulator. CARLA is heavily used in AV research, especially RL driven one. Zoon (2021) shows that the shield allows the vehicle in the CARLA simulator to drive in two different scenarios: one where the car needs to drive along a road without going off-course and one where the car needs to drive safely among other vehicles. The first scenario aims to show the performance of the shield model when no safety issues are of concerns, while the second aims to give context to the works of the shield when safety is of concern. In both scenarios, the shield performs as expected, allowing the car not only to drive consistently on the road, but also providing the agent a policy that keeps the agent in a safe state space. Considering that the CARLA simulator has been successfully validated by Dosovitskiy et al. (2017), the shielding method is validated as it performs in its intended matter on a validated driving simulator.

Part IV

Closure

10

Conclusion

Airborne fatalities are highly connected to in-flight loss of control. The latter usually occurs when it is not possible to return to a safe flight condition once entered stall. To avoid this, it is necessary to develop flight controllers able not only to recover from loss of control, but to prevent it. By developing intelligent flight controllers, aviation safety can be greatly improved. This research presents the development of an offline, model-free Deep Deterministic Policy Gradient (DDPG) controller equipped with a shield, a safety enhancing model, used for the safe control of a Cessna Citation 500.

The first research question is showed below.

RQ-1 **What are the main challenges in flight control?**

- *RQ-1.1* How is safety in flight control defined and assessed?
- *RQ-1.2* What is the state-of-the-art of flight control?

In the context of this research, safety has been defined as the ability to maintain an aircraft in flight conditions that do not lead to unsafe state space, as for example stall. Safety can be assessed by defining specific safety limits for the flight envelope that the aircraft shall not exceed. This definitions, specified in Chapter 3, allow research question *RQ-1.1* to be answered. In order design an safe version of a flight controller, it is important to investigate the current solutions. It is determined that linear uncoupled gain-scheduled controllers are considered the state-of-the-art approach to flight control. These are based look-up tables with parameters of the system's dynamics hence do not provide flexibility for handling unexpected conditions. Therefore, developing controllers that can adapt to different environments is deemed the central point of this thesis project. This provides an answer to *RQ-1.2*. With both sub-questions addressed, ***RQ-1*** is fully answered.

RQ-2 **Why is reinforcement learning being introduced in flight control?**

- *RQ-2.1* How is safety in reinforcement learning defined and assessed?
- *RQ-2.2* What is the state-of-the-art in reinforcement learning?
- *RQ-2.3* What is the state-of-the-art of reinforcement learning in flight control?
- *RQ-2.4* Which flight control task is more relevant to approach with reinforcement learning and why?

Chapter 3 gives an overview of the main safe reinforcement learning methods discussed in literature. In the context of this work, safe RL methods are defined as approaches that protect the agent from pursuing dangerous actions and do not lead to unsafe state spaces. This research quantifies safety by finding the amount of time the agent visits an unsafe state space $\mathcal{S}_{\text{unsafe}}$. If the agent never visits $\mathcal{S}_{\text{unsafe}}$, the simulation is considered safe. This definition satisfies *RQ-2.1*. As mentioned before, the aim of the research is to improve safety in flight control. It is known that the majority of fatalities are due to in-flight loss of control during climb and approach procedures. It was decided that, following the definition of safety given in Chapter 1, the control task central to this work will be flight path control. Not

only this task is essential during climb, descent and landing, but it allows parameters that quantify stall to be tracked. Additionally, according to EASA (2021), a crucial step in stall prevention is to be able to maintain a defined flight path. This choice allows for *RQ-2.4* to be answered. Once it was established that the learning controller would be developed with a reinforcement learning method, a taxonomy of the main classes was given in Chapter 2 to answer *RQ-2.2*. The most recent researches in RL identify Deep Reinforcement Learning (DRL) algorithms as the most promising techniques. By using deep network as Q-function approximators, these algorithms are able to provide stable and robust learning with improved generated policies. By identifying DRL as the state-of-the-art of RL, *RQ-2.2* can be answered.
Since reinforcement learning in flight control is not entirely recent, Chapter 4 gives an overview of three state-of-the-art controllers based on reinforcement learning algorithms. The papers discussed are specifically chosen as they implement a DDPG algorithm for airborne flight control. By presenting these papers, *RQ-2.3* is answered. By discussing these sub-questions, an overview on why reinforcement learning is introduced in flight control is given. This allows ***RQ-2*** to be answered.

RQ-3 **How can safety in flight control be improved using reinforcement learning techniques?**

RQ-3.1 What RL methods can be implemented and what are their characteristics in order to improve safety of a flight control system of a fixed wing aircraft?

RQ-3.2 What are the necessary requirements to be set on the RL method and how do they relate to the requirements of the flight control system?

In order to answer *RQ-3.1*, a detailed taxonomy of Safe RL methods is given in Chapter 3. Garcıa and Fernández (2015) define SRL in two classes namely algorithms that modify the optimization criterion and algorithms that modify the exploration process. The first class aims at introducing a metric for risk so that only safe actions are allowed. One downside of these methods is that explicit values for transition probabilities and rewards are needed, hence they are not applicable in all domains. The second class modifies the way the agent explores, mainly by providing external knowledge. Although information about the model are required, these methods have been effectively used in flight control research. One novel method is developed by combining the two classes of algorithms. Shielding is an safe RL algorithm that combines constrained criterion, an algorithm from the first class, together with teacher-advice, an algorithm from the second class. Although not yet implemented in flight control, it is currently popular in Atari games and autonomous driving. In order for a safe RL algorithm to be used in flight control, different characteristics have to be defined. To maintain the aircraft in a safe flight envelope, unsafe states should not be reached. For example, stall is obviously discouraged, hence the flight controller should be able to keep the angle of attack away from α_{stall}. To do so, the RL algorithm should have some knowledge about unsafe states. This can be done by setting the reward function in a matter that penalizes the agent when unsafe states are visited during training. Additionally, flight control tasks have continuous states and actions, hence the RL algorithm should be able to support continuous state-action spaces. To avoid providing the agent with an inherently safe environment, the RL algorithm should be model-free. Therefore, an algorithm that matches the requirements of a flight controller is found in DDPG as it is a model-free, off-policy algorithm that supports continuous state-action spaces. With this analysis, *RQ-3.2* is answered. With both sub-questions addressed, ***RQ-3*** is fully answered.

RQ-4 **What reinforcement learning algorithm can be combined with a safety enhancing technique to improve safety?**

Although many different RL algorithms can be combined with safe RL approaches, this research presents the development of an offline, model-free Deep Deterministic Policy Gradient (DDPG) controller equipped with a shield, a safety enhancing technique, used for the safe control of a Cessna Citation 500. The shield is composed of a Safe Initial Policy (SIP) agent and a Safety Range M_{SR} model. The former is a trained DDPG agent with knowledge about state space safety able to suggest safe actions to the main DDPG agent, while the latter is a rule based model in charge of overruling actions that would lead to unsafe state spaces. The SIP model is a clear example of modifying the optimization process as the agent is penalized when unsafe states are visited. The overall controller can be considered an example of teacher-advice, as the shield provides the DDPG agent counsel on which actions should be avoided. Therefore, this research combines DDPG algorithms with both classes of safe RL discussed in

Chapter 3. This description allows **RQ-4** to be answered.

The research objective of this thesis works is the following.

> *To improve the safety of a State-of-the-Art learning flight control system for a fixed wing aircraft by implementing the most promising reinforcement learning algorithm.*

In this research, a shielded DDPG controller for controlling the flight path angle of a Cessna Citation 500 at top of descent is developed. In the context of this research, to states are monitored to assess safety namely the angle of attack and the load factor. These two states are considered safe if they remain within certain limits defined by governing authorities and defined by the controller designer. The shield is able to maintain the agent within safe limits during nominal operations, meaning that $\mathcal{S}_{\text{unsafe}}$ is never visited, and is able to track the reference flight path angle with a normalized Mean Absolute Error Percentage (nMAE%) of 24.0%. The controller was tested in different initial flight conditions to show that it is robust to many stages of the descending envelope. To test the controller in more realistic condition, atmospheric disturbances and biased sensor noise were introduced in the environment. After these tests, it was shown that the shielded controller was able to maintain the agent in the safe flight envelope while successfully tracking the flight path angle. Therefore, with the development of a successful State-of-the-Art safe RL model equipped with a shield for the control of the Cessna Citation 500 aircraft, the research objective for this thesis can be considered met. Although initially developed for Atari games by Alshiekh et al. (2018) and subsequently applied by Zoon (2021) on autonomous vehicles, this research shows shielded RL to be effective in the field of flight control. The flexibility of this technique makes it appealing for many applications, not only within aviation, but any high complexity control task that can benefit from improved safety.

11

Recommendations

Several recommendations can be made after the research has concluded. Following, an overview of the most relevant.

- The hyperparameters, the network structures and the Ornstein-Uhlenbeck noise parameters chosen for the DDPG and SIP agents have been taken from Lillicrap et al. (2015) and Ba et al. (2016). During this research, none of these variables were investigated and no sensitivity analysis was conducted to optimize the learning. Considering the long training times, it is suggested to research the effect of each parameter to shorten training and reaching a better learning stability.

- DDPG has been selected among other model-free, off-policy algorithms for continuous state-actions spaces as it has been consistently used in flight control researches. However, DDPG shows flaws such as the overestimation of the Q-function. To reduce this issue, it is recommended to investigate two updates of DDPG algorithms, namely Soft Actor-Critic (SAC) and Twin Delayed DDPG (TD3).

- The flight envelope analyzed in this research goes from top of descent until an altitude where the ILS could be intercepted. Although robust to these conditions, it is suggested to analyze higher altitudes and different configurations.

- The SIP agent's reward function is composed of three distinct contribution. R_t gives a reward when when the flight path angle is successfully tracked while R_α and R_n provide a reward if the angle of attack and the load factor are kept within safe limits. Although the trained SIP agent successfully provides the agent with safe actions that keep α and n within safe spaces, it is not always able to avoid unsafe spaces when the flight conditions greatly change. The reason behind this can be linked to the reward function. Since the latter is made of three distinct parts, at each time t, the SIP agent needs to perform a trade-off on whether to avoid unsafe spaces or reduce the γ_{ref} tracking accuracy. It is recommended to create an hierarchy within the reward function to allow the SIP controller always to chose an action that keeps the agent within safety limits, even at the expense of tracking performance.

- The M_{SR} is triggered individually in case α or n exceed the safety limits and the safest action between what M_{SR_α} and M_{SR_n} recommend is chosen. However, it can occur that the chosen $\hat{q}_{ref}$ leads one of the two states that at a time t is safe, to a risky state at $t+1$. It is therefore recommended to create a feedback loop from one M_{SR} to the other and vice-versa to avoid such an issue to arise.

- This research investigates the working of the shield post-learning, meaning that the DDPG controller learns without the shield in the loop. It would be interesting to investigate whether the addition of the shield would allow the agent to learn a safe policy.

As the DDPG agent would get $\hat{q}_{ref}$ as observation, it would be able to learn which of the actions proposed are actually fed to the environment or not. Adding the load factor as observation and introducing a penalty in the reward function once the shield is activated, it would be possible for the agent to learn why certain actions are overruled and eventually learn to avoid such unsafe actions. Experiments with this set up were carried during this research but unfortunately with no success.

Bibliography

Akhtar, S. A., Kolarijani, A. S., and Esfahani, P. M. (2021). Learning for control: An inverse optimization approach. *IEEE Control Systems Letters*.

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. (2018). Safe reinforcement learning via shielding. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Babuška, R. and Kober, J. (2010). Knowledge-based control systems - lecture notes. *Delft University of Technology*.

Borkar, V. S. (2002). Q-learning for risk-sensitive control. *Mathematics of operations research*, 27(2):294–311.

Chen, S. and Li, Y. (2020). An overview of robust reinforcement learning. In *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6. IEEE.

Chen, Z. and Li, M. (2018). Soft Actor-Critic. University of Toronto - Lecture Notes.

Clarke, S. G. and Hwang, I. (2020). Deep reinforcement learning control for aerobatic maneuvering of agile fixed-wing aircraft. In *AIAA Scitech 2020 Forum*, page 0136.

Cook, M. V. (2012). *Flight dynamics principles: a linear systems approach to aircraft stability and control*. Butterworth-Heinemann.

Coraluppi, S. P. and Marcus, S. I. (1999). Risk-sensitive and minimax control of discrete-time, finite-state Markov decision processes. *Automatica*, 35(2):301–309.

Dally, K. (2021). Deep Reinforcement Learning for Flight Control: Fault-Tolerant Control for the PH-LAB. *MSc thesis, Delft University of Technology.* MSc thesis, Delft University of Technology, 2021.

De Prins, J. (2010). Stochastic aerospace systems - lecture notes. *Delft University of Technology*.

Dehnert, C., Junges, S., Katoen, J.-P., and Volk, M. (2017). A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer.

Di Castro, D., Tamar, A., and Mannor, S. (2012). Policy gradients with variance related risk criteria. *arXiv preprint arXiv:1206.6404*.

Dongare, A. D., Kharde, R. R., Kachare, A. D., et al. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, 2(1):189–194.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR.

Driessens, K. and Džeroski, S. (2004). Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304.

EASA (2021). Equivalent safety finding, enhanced stall protection. ESF-B25.103-01(2).

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2021). First return, then explore. *Nature*, 590(7847):580–586.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. Proceedings of Machine Learning Research.

Garcıa, J. and Fernández, F. (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480.

Gaskett, C. (2003). Reinforcement learning under circumstances beyond its control.

Geramifard, A., Redding, J., and How, J. P. (2013). Intelligent cooperative control architecture: a framework for performance improvement using safe learning. *Journal of Intelligent & Robotic Systems*, 72(1):83–103.

Ha, S., Kim, J., and Yamane, K. (2018). Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot. In *2018 15th International Conference on Ubiquitous Robots (UR)*, pages 348–354.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Heger, M. (1994). Consideration of risk in reinforcement learning. In *Machine Learning Proceedings 1994*, pages 105–111. Elsevier.

Heyer, S., Kroezen, D., and Van Kampen, E.-J. (2020). Online Adaptive Incremental Reinforcement Learning Flight Control for a CS-25 Class Aircraft. In *AIAA Scitech 2020 Forum*.

Hofer, L. and Gimbert, H. (2016). Online reinforcement learning for real-time exploration in continuous state and action markov decision processes. *arXiv preprint arXiv:1612.03780*.

Huys, Q., Cruickshank, A., and Series, P. (2014). Reward-based learning, model-based and model-free. In *Encyclopedia of Computational Neuroscience*, pages 1–10. Springer New York.

IATA (2015). Loss of control in-flight accident analysis report 2010-2014. *Montreal-Geneva: International Air Transport Association*.

Jansen, N., Könighofer, B., Junges, S., and Bloem, R. (2018). Shielded decision-making in MDPs. *arXiv preprint arXiv:1807.06096*.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Louis, R. and Yu, D. (2019). A study of the exploration/exploitation trade-off in reinforcement learning: Applied to autonomous driving.

Mannucci, T., van Kampen, E.-J., de Visser, C. C., and Chu, Q. P. (2015). SHERPA: a safe exploration algorithm for Reinforcement Learning controllers. In *AIAA Guidance, Navigation, and Control Conference*, page 1757.

Matiisen, T., Oliver, A., Cohen, T., and Schulman, J. (2019). Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740.

Mazareanu, E. (2022). Number of flights performed by the global airline industry from 2004 to 2022. *International Air Transport Association*.

Mihatsch, O. and Neuneier, R. (2002). Risk-sensitive reinforcement learning. *Machine learning*, 49(2):267–290.

Milz, D. M. and Looye, G. (2020). Design and evaluation of advanced intelligent flight controllers. In *AIAA Scitech 2020 Forum*, page 1846.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. volume 1, page 2.

Ni, Z., He, H., Zhong, X., and Prokhorov, D. V. (2015). Model-free dual heuristic dynamic programming. *IEEE transactions on neural networks and learning systems*, 26(8):1834–1839.

Pollack, T. (2019). Safe Curriculum Learning for Primary Flight Control. *MSc thesis, Delft University of Technology.* MSc thesis, Delft University of Technology.

Powell, W. B. (2009). What you should know about approximate dynamic programming. *Naval Research Logistics (NRL)*, 56(3):239–249.

Ravichandar, H., Polydoros, A. S., Chernova, S., and Billard, A. (2020). Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:297–330.

Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., and Hebert, M. (2013). Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. Citeseer.

Sanghi, N. (2021). Combining Policy Gradient and Q-Learning. In *Deep Reinforcement Learning with Python*, pages 251–303. Springer.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. Proceedings of Machine Learning Research.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. P.

Sonneveldt, L., Chu, Q. P., and Mulder, J. A. (2007). Nonlinear flight control design using constrained adaptive backstepping. *Journal of Guidance, Control, and Dynamics*, 30(2):322–336.

Steinberg, M. L. (2001). Comparison of intelligent, adaptive, and nonlinear flight control laws. *Journal of Guidance, Control, and Dynamics*, 24(4):693–699.

Sun, B. and van Kampen, E.-J. (2020). Incremental model-based global dual heuristic programming with explicit analytical calculations applied to flight control. *Engineering Applications of Artificial Intelligence*, 89:103425.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Tang, C. and Lai, Y.-C. (2020). Deep Reinforcement Learning Automatic Landing Control of Fixed-Wing Aircraft Using Deep Deterministic Policy Gradient. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1–9.

Van den Hoek, M., de Visser, C., and Pool, D. (2018). Identification of a cessna citation ii model based on flight test data. In *Advances in Aerospace Guidance, Navigation and Control: Selected Papers of the Fourth CEAS Specialist Conference on Guidance, Navigation and Control Held in Warsaw, Poland, April 2017*, pages 259–277. Springer.

Van Der Linden, C. (1998). Dasmat-delft university aircraft simulation model and analysis tool: A matlab/simulink environment for flight dynamics and control analysis. *Series 03: Control and Simulation 03*.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Wang, Z., Luo, W., Gong, Q., Cui, Y., Tao, R., Wang, Q., Liang, Q., and Wang, S. (2020). Attitude Controller Design based on Deep Reinforcement Learning for Low-cost Aircraft. In *2020 Chinese Automation Congress (CAC)*, pages 463–467.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. Proceedings of Machine Learning Research.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Weng, L. (2018). Policy Gradient Algorithms. *lilianweng.github.io/lil-log*.

Wiering, M. A. and Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12(3).

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Xiong, J. (2021). Safe Hierarchical Reinforcement Learning for Fixed-Wing Flight Control. *MSc thesis, Delft University of Technology.* MSc thesis, Delft University of Technology, 2019.

Yuan, H. (2019). Self-corrective Apprenticeship Learning for Quadrotor Control. *MSc thesis, Delft University of Technology.* MSc thesis, Delft University of Technology, 2019.

Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *Association of Computing Machinery Surveys (CSUR)*, 52(1):1–38.

Zoon, J. (2021). Safe Reinforcement Learning by Shielding for Autonomous Vehicles. *MSc thesis, Delft University of Technology.*