University of Strathclyde Glasgow

Can Spacecraft Think? Intelligent Learning Control Onboard Spacecraft

PhD Thesis

Callum James Wilson

Aerospace Centre of Excellence
Department of Mechanical and Aerospace Engineering
University of Strathclyde, Glasgow

September 2025

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom Copyright Acts as qualified by University of Strathclyde Regulation 3.50. Due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Signed: CJ Wilson

Date: 15 September 2025

To my Mum, Sally.

“You could sell basic robots designed for instruction; robots that could be modelled to a job, and then modelled to another, if necessary. Robots would become as versatile as human beings. *Robots could learn!*”

Isaac Asimov, “Lenny”, The Complete Robot.

Acknowledgements

First I wish to thank my supervisor Annalisa Riccardi for her expert guidance throughout the PhD. I am very grateful to her for the initial offer of this position and for the many ways she has supported me in my research.

I also wish to thank Edmondo Minisci for all the advice and digestivi he has provided. Several other members of academic staff within ACE have also supported me in many different ways throughout the PhD. In particular, my thanks go to Jinglang Feng, Christie Maddock, and Massimiliano Vasile. I also thank the MAE Admin Team for all the practical support they have given.

As part of the ICELab, I have had the privilege of working alongside several other excellent researchers. I am grateful for the many interactions I have had with Mateusz Polnik, Audrey Berquand, Deep Bandivadekar, Francesco Marchetti, Cheyenne Powell, Paul Darm, Bryn Jones, Robert Cowlishaw, and Seonaid Rapach.

There are many other current and former PhDs and Postdocs within ACE and the wider department who have made my PhD much more enjoyable. There are too many names to list here, but to all those with whom I have shared lunches, drinks, and trips together I am incredibly grateful.

I am thankful for the financial support from ESA Education to attend the International Astronautical Congress 2022 and to the IMechE for running the Speak Out for Engineering competition—both of which helped me develop as a researcher. At the IAC 2022 I got to meet many other young space enthusiasts who left a lasting impression on me, for which I am very grateful.

Thank you to Madeleine Taylor and her family for all their support over the years and especially during the pandemic.

Acknowledgements

Throughout the PhD and most of my undergraduate I have had the privilege of being part of the Strathclyde University Jazz Orchestra and the Strathclyde University Concert Band. I have gained so much from both groups and made many friends who have supported me during the PhD, to whom I am very grateful. I wish to thank all those who have been responsible for running these bands who put considerable time and effort into making them as enjoyable as they are.

I am fortunate to have had excellent flatmates while completing the PhD and I am very grateful for their company. Thank you to Joey O'Neill, Ellen Stewart, and Gianluca Filippi. Thanks also to Alice and Tim Simons for providing a great place to stay at the start of the PhD.

My friends who know me as Dilly have mostly been around since my undergraduate and I am so grateful for each and every one of them. While completing the PhD, we have had many shared struggles that they helped me through as well as sharing some of the best times. I cannot imagine having done my PhD without them.

I am grateful for the moments of joy shared with Jess and Graeme, Chris and Susie, Jonathon and Emily, and their families.

Thank you to Louie Anderson for always reminding me who I am.

Special thanks go to my family: Dad, Annie, and Jess. They have supported me in countless ways throughout my life and I owe completing the PhD in no small part to them.

Finally, thank you to Ellie for everything. You have given me so much support over the final stretch of the PhD and I look forward to supporting you in completing yours.

Abstract

Spacecraft guidance, navigation, and control systems need to operate under substantial uncertainties. This presents challenges when designing these control systems using conventional methods that require a certain level of knowledge of the system being controlled. Intelligent control systems have emerged as a means of addressing these challenges. These systems combine theories from automatic control, operations research, and artificial intelligence to derive controllers that can deal with different types of uncertainty. Various methods from the field of artificial intelligence can be used to develop intelligent control systems, however these are often computationally expensive which limits their applicability in spacecraft control problems. A key feature of intelligent control is the ability to adapt the control system online, which presents further difficulties when this must be done onboard a spacecraft. This thesis explores the use of reinforcement learning techniques for intelligent control applied to spacecraft powered descent. The proposed approach combines a reinforcement learning agent for handling uncertainties with conventional optimisation methods to improve the agent's performance. In addition, the agent updates its control policy online using a novel update mechanism called Extreme Q-Learning Machine, which allows the control system to operate in a changing environment. To demonstrate the potential for this method to be implemented onboard spacecraft, results are shown from running online updates on flight suitable hardware. This work provides one possible avenue for increasing the level of intelligence of spacecraft control systems.

Contents

Acknowledgements **iv**

Abstract **vi**

List of Figures **x**

List of Tables **xiv**

Abbreviations **xvii**

Nomenclature **xix**

1 Introduction **1**

1.1 Research Objectives 4

1.2 Research Output 5

1.2.1 Peer-reviewed Journal Publications 6

1.2.2 Peer-reviewed Conference Publications 6

1.2.3 Conference Publications 6

1.3 Thesis Organisation 7

2 Background **8**

2.1 Reinforcement Learning for Optimal Control 8

2.1.1 Reinforcement Learning Problems 9

2.1.2 Policies and Value Functions 11

2.1.3 Q-Learning 13

2.2 Reinforcement Learning with Function Approximation 14

2.2.1 Policy and Value Gradient Methods 15
2.2.2 Q-Networks 16
2.3 Extreme Learning Machines 18
2.3.1 Extreme Learning Machine Theory 19
2.3.2 Regularized Extreme Learning Machine 21
2.3.3 Incremental Extreme Learning Machine 23
2.4 Summary 26

3 A Taxonomy of Intelligent Control **27**
3.1 Path to Intelligent Control 29
3.2 Defining Intelligent Control 34
3.2.1 Methods for Intelligent Control 36
3.2.2 Dimensions of Intelligent Control 38
3.3 Taxonomy 41
3.3.1 Environment Knowledge 41
3.3.2 Controller Knowledge 43
3.3.3 Goal Knowledge 44
3.4 Classification of Relevant Examples 45
3.5 Summary 49

4 Q-Learning for Spacecraft Powered Descent **52**
4.1 Relevant Literature 54
4.2 Powered Descent Problem 57
4.2.1 Environment Dynamics 58
4.2.2 Shaped Reward Function 60
4.2.3 State Space 62
4.2.4 Action Space 63
4.3 Results of DQN Applied to Powered Descent 64
4.3.1 Hyperparameter Study 64
4.3.2 Action Size Selection 68
4.3.3 Training and Testing Agents 71

4.3.4 Training with Raw State Representation 77
4.4 Summary 81

5 Learning with Demonstrations 82
5.1 Relevant Literature 83
5.2 DQN with Optimal Demonstrations 85
5.2.1 Optimal Control Problem 86
5.2.2 Demonstrations from Optimal Trajectories 86
5.2.3 Demonstration Experience Replay 88
5.3 Powered Descent Optimal Control Problem 88
5.3.1 Optimal Control Problem 88
5.3.2 Convexification of the Optimal Control Problem 91
5.4 Results of DQN with Demonstrations Applied to Powered Descent . . . 93
5.4.1 Optimal Trajectories 93
5.4.2 Discretised Optimal Trajectories 95
5.4.3 Training Agents with Demonstrations 98
5.4.4 Testing Agents 100
5.5 Summary 104

6 Online Updates for Continuous Learning 106
6.1 Relevant Literature 108
6.2 DQN with Online EQLM Updates 111
6.2.1 Offline Agent Pretraining 112
6.2.2 Offline EQLM Initialisation 112
6.2.3 Online EQLM Updates 114
6.3 Results of DQN with Online Updates Applied to Powered Descent . . . 115
6.3.1 Offline Pretraining 116
6.3.2 Testing Initialisation Random Seeds 118
6.3.3 Online Updates 121
6.3.4 Online Updates with a Changing Environment 125
6.3.5 Hardware Results 132

6.4 Summary . . . 135

7 Conclusions **136**

7.1 Summary of Objectives . . . 136

7.2 Limitations and Future Work . . . 138

A Optimal Demonstrations for Reward Shaping **141**

A.1 Reward Shaping Method . . . 141

A.2 Simulation Setup . . . 143

A.3 Results . . . 144

Bibliography **146**

List of Figures

2.1 Agent-environment interaction in reinforcement learning. 10
2.2 Diagram of a single-layer feedforward network indicating the relevant parameters defining the transformation. 20

3.1 Open loop control system. 30
3.2 Feedback control system. 31
3.3 Optimal control system. 32
3.4 Stochastic control system. 32
3.5 Adaptive or learning control system. 34
3.6 Intelligent control is the interaction between Artificial Intelligence, Operations Research, and Automatic Control Systems. 35
3.7 Intersections of Artificial Intelligence (AI) techniques commonly used in Intelligent Control (IC) systems. 37
3.8 Parallel coordinates plot showing the number of applications of each IC classification previously surveyed. 50

4.1 Illustration of the lander powered descent frame of reference and glideslope limit. 58
4.2 Initial agent learning curve to establish the number of training episodes. 65
4.3 Parallel plots showing hyperparameter optimisation evaluations for area. Colour indicates loss normalised via a power norm for clarity. 68
4.4 Parallel plots showing hyperparameter optimisation for reward. Colour indicates loss. 69

4.5 Change in performance with varying values of action magnitude for the powered descent problem. Error-bars denote one standard error. 69
4.6 Change in performance with varying mid-points of z-direction action for the powered descent problem. Error-bars denote one standard error. . . 70
4.7 Learning curves averaged over 50 runs for each agent. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 120 episodes for clarity. 71
4.8 Mean and worst performing learning curves for the reward-optimised agent. 72
4.9 Average reward and steps per episode over a training run of the PPO agent. 73
4.10 Average reward and steps per episode over a training run of Deep Q-Networks (DQN) for two different sets of hyperparameters. Uniformly filtered (average) over 120 episodes for clarity. 74
4.11 Scatter and Kernel Density Estimate (KDE) plots showing distributions of terminal landing states for each agent. 75
4.12 Trajectory of the area-optimised agent over a sample episode with commanded thrusts. 77
4.13 Trajectory of the reward-optimised agent over a sample episode with commanded thrusts. 78
4.14 Learning curves for agents trained with a raw state representation. . . . 79
4.15 Scatter and KDE plots showing distributions of terminal landing states for each agent trained with a raw state representation. The z-velocity axis is truncated for clarity. 80

5.1 Histogram of minimum glideslope angle over optimal trajectories starting from different initial conditions. 94
5.2 Example optimal trajectory with constraints on the maximum z-direction thrust. 95
5.3 Example optimal trajectory without constraints on the maximum z-direction thrust. 96
5.4 Histogram of minimum glideslope angle over discretised optimal trajectories starting from different initial conditions. 97

5.5 Example optimal trajectory following discretisation of the actions. . . . 98

5.6 Learning curves averaged over 8 runs for different values of k_{demo}. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 150 episodes for clarity. 100

5.7 Average reward and steps per episode over best performing training run for different values of k_{demo}. Uniformly filtered (average) over 150 episodes for clarity. 101

5.8 Box plots of the terminal position and velocity of the trained agents over 5000 test episodes. Coloured line indicates median and whiskers indicate minimum and maximum values. 102

5.9 Scatter and KDE plots showing distributions of terminal landing states for best agents trained with and without demonstrations, where $r_z \leq 0.01m$. 103

5.10 Trajectory of the agent trained with optimal control demonstrations. . . 104

6.1 Schematic illustration of the three main phases in the proposed method of online updates. 111

6.2 Diagram of a deep feedforward network indicating the relevant parameters that are fixed following pretraining and output weights that are updated online. 113

6.3 Learning curves averaged over 8 pretraining runs. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 150 episodes for clarity. 117

6.4 Average reward and steps per episode over an offline pretraining run of Q-learning. Uniformly filtered (average) over 150 episodes for clarity. . . 118

6.5 Scatter and KDE plots showing distributions of terminal landing states for the best agents before and after Extreme Q-Learning Machine (EQLM) updates. 122

6.6 Scatter and KDE plots showing distributions of terminal landing states with the original training environment and environment with new disturbance magnitudes. 127

6.7 Variation in mean terminal states and fuel consumption with number of online EQLM training episodes. Shaded area indicates +/- one standard deviation. 128
6.8 Scatter and KDE plots showing distributions of terminal landing states for agents with and without further online updates in the environment with new disturbance magnitudes. 130
6.9 Trajectory of the agent trained with online updates and the environmental disturbance forces along this trajectory. 131
6.10 Average norm change in output weights over 8 test runs of 100 steps with EQLM updates and gradient based updates. 133
6.11 Histogram of time taken to select actions and update weights over 500 steps on Jetson Nano hardware. 133
6.12 Trend in update times for different network sizes with and without a Graphics Processing Unit (GPU) on edge hardware. 134

A.1 Positions and velocities from optimal trajectories and the polynomial fit target velocities. 142
A.2 Average reward and steps per episode over a training run with a new shaped reward. Uniformly filtered (average) over 150 episodes for clarity. 144
A.3 Scatter and KDE plots showing distributions of terminal states for the best agent trained with new shaped reward. 145
A.4 Trajectory of the agent over a sample episode with commanded thrusts. 146

List of Tables

4.1 Range of initial conditions in the Mars lander powered descent environment. 59

4.2 Values of terms defining the powered descent reward function. 62

4.3 Selected DQN hyperparameters when optimising for area under learning curve and final reward. 67

4.4 Comparison of test results from both agents trained with DQN. Statistics shown for terminal state over 5000 test episodes. 76

4.5 Test results for agent trained with PPO. Statistics shown for terminal state over 5000 test episodes. 76

4.6 Selected DQN hyperparameters when optimising for area under learning curve and final reward with a raw state representation. 79

4.7 Comparison of test results from both agents trained with a raw state representation. Statistics shown for terminal state over 5000 test episodes. 81

5.1 Range of initial conditions for generating optimal trajectories. 94

5.2 Results of discretised optimal trajectories. 97

5.3 Hyperparameters for training DQN with demonstrations. 99

5.4 Comparison of test results from agents trained with and without demonstrations. Statistics shown for terminal state over 5000 test episodes. . . 102

6.1 Hyperparameters for agent pretraining. 117

6.2 Summary statistics of terminal position and velocity for the pareto optimal pretrained weights, initialisations, and EQLM update seeds over 500 test episodes. 119

6.3 Summary statistics of terminal position and velocity for the pareto optimal EQLM update seeds over 500 test episodes. 120
6.4 Comparison of test results from the pretrained agent and the best performing agent with EQLM updates. Statistics shown for terminal state over 5000 test episodes. 121
6.5 Comparison of test results from the agent with EQLM updates + 10m optimal descent and the full trajectory optimisation with disturbances. Statistics shown for terminal state and fuel consumption over 5000 test episodes for the EQLM agent and 100 test episodes for each of the 33 initial states of the optimal trajectories. 124
6.6 Initial conditions and uncertainties for the environment with new disturbances. 126
6.7 Comparison of test results in the environment with disturbance changes for agents with and without further online updates. Statistics shown for terminal state and fuel consumption over 5000 test episodes. 129
6.8 Comparison of test results from the agent with EQLM updates + 10m optimal descent and the full trajectory optimisation with updated disturbances. Statistics shown for terminal state and fuel consumption over 5000 test episodes for the EQLM agent and 100 test episodes for each of the 33 initial states of the optimal trajectories. 132
6.9 Timings for weight updates across different hardware configurations. . . 134

A.1 Polynomial coefficients for target velocity. 143
A.2 Range of initial conditions for training the agent. 143
A.3 Test results from the agent trained with new shaped reward. Statistics shown for terminal state over 5000 test episodes. 146

Abbreviations

AI Artificial Intelligence

EA Evolutionary Algorithm

EC Evolutionary Computing

FL Fuzzy Logic

GA Genetic Algorithm

GP Genetic Programming

KDE Kernel Density Estimate

ML Machine Learning

LLM Large Language Model

ANFIS Adaptive Neuro-Fuzzy Inference System

FDIR Fault Detection, Isolation, and Recovery

IC Intelligent Control

KBS Knowledge Based System

MPC Model Predictive Control

SOC Self-Organising Control

ERM Empirical Risk Minimisation

SRM Structural Risk Minimisation

NLP Non-Linear Programming

TPE Tree of Parsen Estimators

DQN	Deep Q-Networks
EQLM	Extreme Q-Learning Machine
GPI	Generalised Policy Iteration
HER	Hindsight Experience Replay
LSTD	Least Squares Temporal Difference
MDP	Markov Decision Process
PPO	Proximal Policy Optimisation
RL	Reinforcement Learning
RL-CD	Reinforcement Learning with Context Detection
TD	Temporal Difference
DNN	Deep Neural Network
ELM	Extreme Learning Machine
GPU	Graphics Processing Unit
LS-IELM	Least Squares Incremental Extreme Learning Machine
NN	Neural Network
RNN	Recurrent Neural Network
SLFN	Single-Layer Feedforward Network
DOF	Degrees-of-Freedom
GNC	Guidance Navigation and Control
V&V	Verification and Validation
ZEM	Zero Effort Miss
ZEV	Zero Effort Velocity

Nomenclature

Neural Networks and Extreme Learning Machines

$\bar{\gamma}$ regularisation parameter

$\boldsymbol{\beta}$ neural network output weights

$\hat{\boldsymbol{\beta}}$ calculated neural network output weights

λ Lagrangian multiplier

$\mathbf{e}$ error between neural network output and target output

$\mathbf{H}$ matrix of hidden layer outputs

$\mathbf{h}_i$ output of the i_{th} hidden layer

$\mathbf{o}$ neural network output

$\mathbf{T}$ matrix of target outputs

$\mathbf{w}$ hidden node input weights

$\mathbf{x}$ input from training data

$\mathbf{y}$ target output from training data

$\tilde{N}$ number of hidden nodes

b hidden node bias

g activation function

i index of hidden node i

j index of sample in training data

m number of terms in the outputs

N number of samples in training data

n number of terms in the inputs

Powered Descent

α_T fuel consumption rate

$\bar{a}$ normalised action

$\bar{a}^*$ normalised optimal action

Γ slack variable

γ_{gs} glideslope angle

$\hat{\mathbf{n}}$ unit direction vector

$\mathbf{E}_{norm}$ matrix of direction vectors normal to surface

$\mathbf{E}_{tan}$ matrix of direction vectors tangential to surface

$\mathbf{F}_D$ disturbance force vector

$\mathbf{F}_T$ engine thrust vector

$\mathbf{g}$ gravitational field strength

$\mathbf{r}$ position

$\mathbf{U}$ space of allowable control inputs

$\mathbf{v}$ velocity

$\mathbf{X}$ space of allowable states

ϕ thrust angle of spacecraft

$\rho_{1/2}$ lower/upper thrust magnitude

$\sigma, \mathbf{u}$ normalised control variables

F_i disturbance force component

m lander mass

T_i thrust component

$t_{min/max}$ minimum/maximum time-of-flight

z natural log of mass

Reinforcement Learning

α learning rate

$\boldsymbol{\theta}$ parameter vector of agent's policy

$\boldsymbol{\theta}^-$ parameter vector of agent's target policy

δ temporal difference error

ϵ exploration probability

ϵ_0 initial exploration probability

ϵ_f final exploration probability

γ discount factor parameter

$\mathbf{R}$ vector of cumulative rewards over a number of episodes

$\mathcal{D}$ replay memory

π policy

$\tilde{N}_l$ number of hidden nodes in layer l

a	action
a^*	action taken using an open-loop optimised policy
C	number of timesteps between target parameter updates
G_t	cumulative reward received following timestep t
k	minibatch size
k_{demo}	minibatch size of demonstration data
L	loss function
N_ϵ	number of episodes to decrease ϵ
N_{ep}	total number of training episodes
Q	estimate of the action-value function
q_*	optimal action-value function
q_π	action-value function when following policy π
Q_θ	estimate of the action-value function given parameters $\boldsymbol{\theta}$
r	reward
R^*	optimal cumulative reward over one episode
R^{ep}	cumulative reward over one episode
s	state
s'	observed state following taking an action in a state
s_0	initial state
s_T	terminal state
t	discrete timestep

V estimate of the value function

v_* optimal value function

v_π value function when following policy π

Chapter 1

Introduction

Over the past century, more advanced methods have become necessary to handle an increase in the complexity of control problems. We now require control systems to operate autonomously in very challenging environments where the consequences of failure can be significant. Perhaps no other field embodies this better than spacecraft Guidance Navigation and Control (GNC). These control systems must deal with substantial uncertainties that are inherent to space missions and often without any possibility of human intervention. As humanity's ambitions in space exploration grow bigger, there will be a need for new approaches to GNC that can safely operate in previously unexplored environments.

Despite the relatively short history of spaceflight, there are still many notable success stories to date. In terms of missions beyond geocentric orbits, humanity has successfully landed multiple spacecraft on the planets Venus and Mars [1], landed and returned humans from Earth's Moon [2], visited and returned samples from multiple extraterrestrial bodies [3], captured images of the Sun's poles [4], and sent probes beyond the solar system into interstellar space [5], among other achievements. Alongside the successes of spaceflight, there are failures from which lessons must be learnt. The ExoMars Schiaparelli lander is one such example where the spacecraft GNC failed during the mission's entry, descent, and landing phase [6]. An investigation into this incident concluded that larger than anticipated angular accelerations resulted in erroneous measurements from a sensor, which caused large errors in the altitude estimation that ultimately led to

the hard landing of the spacecraft. While a human operator might have noticed and managed to account for these errors, the lander's control system could not and instead executed its planned sequence of events based on unrealistic data. This case represents just one example of the difficulties in designing effective autonomous control systems for space missions.

Most spacecraft GNC systems are designed based on what will be referred to here as "conventional" control theories. These have a strong grounding in the mathematical analysis of dynamical systems that can guarantee a desired level of performance from a control system under certain conditions. However, when operating in uncertain environments, many conventional control approaches require limiting assumptions on the uncertainties to hold true. When discussing uncertainties in control systems, these are typically divided into epistemic and aleatoric uncertainties. Epistemic uncertainties are due to a lack of system knowledge and these uncertainties can be reduced by gaining more information about the system. Aleatoric uncertainties come from inherent stochasticity in the system. In conventional control approaches, the aleatoric uncertainties are often assumed to be bounded or to come from a fixed, known distribution. In situations outside of these, they are more likely to fail. One class of methods that has emerged to deal with these scenarios is Intelligent Control.

Intelligent Control (IC) combines theories from artificial intelligence, operations research, and automatic control systems to achieve control systems that can handle substantial uncertainties [7]. Automatic control refers to the study of control systems that operate with minimal human intervention. Operations research in the context of IC is mainly concerned with activity planning for highly autonomous systems that decide their own tasks. The field of Artificial Intelligence (AI) has been an area of active research since the mid 20th century, but still the term "AI" lacks a single agreed upon definition [8]. Nevertheless, as it pertains to IC, AI can be understood as a collection of methods that are often referred to as "AI" methods. At the time of writing, AI is most often used to refer specifically to Large Language Models (LLMs) that generate synthetic text and images [9]. However, these models only represent a subset of the broader research field of AI.

A key marker of human intelligence is the ability to learn from experience. This is the fundamental idea behind Reinforcement Learning (RL), which refers to a group of problems and associated AI methods for solving them. These methods learn how to control a system through interaction and optimise their behaviour with respect to a reward function specified by the designer [10]. In the 2010s, RL methods achieved significant breakthroughs in learning to play games [11–13]. Following this, RL has been applied to a variety of challenging control problems such as navigating stratospheric balloons [14], protein engineering [15], control of quantum information processors [16], and controlling nuclear fusion processes [17, 18]. This accordingly generated research interest in RL, alongside other AI methods, for spacecraft GNC problems [19]. Although RL can be applied to many different control problems in aerospace, spacecraft powered descent emerged as a popular test case for applying these methods [20–25]. This is due partly to the problem being well-suited to the RL paradigm of learning through repeated interactions. Furthermore, it requires the control system to operate autonomously in an uncertain environment, which also motivates the use of IC methods more broadly. For these reasons, the research presented here focusses on applying IC, via RL methods, to spacecraft powered descent, but the same methods are applicable in many other GNC problems.

The title of this thesis refers to the question posed by Alan Turing in his seminal work in which he proposed a game that came to be known as the "Turing Test" [26]. He begins the paper by asking the question "can machines think?", but later admits he believes this question to be "too meaningless to deserve discussion". Instead, he changes the question to ask if a "thinking machine" can win his proposed imitation game—the rules of which are well known. The goal then is not to create a machine that can think, but one that can *imitate* a human sufficiently well that it can deceive a human interrogator. Similarly, the question "can spacecraft think?" can be changed to consider if a spacecraft can imitate some aspect of human intelligence and, crucially, could this be useful? This is the question that will be explored in this thesis.

1.1 Research Objectives

The primary objective of this research was to investigate the potential for Intelligent Control (IC) methods to be used onboard spacecraft. This first required a firm grasp on the definition of IC, which is not a trivial task due to the extensive use and abuse of the term in literature. As a result, this motivated additional research into how control systems are classed as "intelligent" control with the aim of creating a formal system of classification. This highlighted the importance of the adaptive and learning capabilities of IC systems as well as their ability to deal with uncertainties.

Although RL methods have been applied to many control problems including spacecraft GNC, most of these applications do not incorporate online learning. This is an important characteristic of IC systems that fits well within the framework of RL problems. Therefore, the scope of this research was narrowed by focussing on new RL approaches to IC. Within the field of spacecraft GNC there are also many conventional (as opposed to intelligent) control approaches that have been well studied. RL methods can learn to control a system without any prior knowledge, but some methods can also incorporate knowledge, for example, from conventional control approaches. It is therefore sensible to try to leverage the capabilities of conventional approaches when training IC systems via RL.

As a final barrier to demonstrating the feasibility of onboard IC, it was necessary to consider the computational requirements of spacecraft hardware. Any IC method that should run onboard a spacecraft needs to be sufficiently small in terms of memory size and run quickly enough to allow online updates. This can be demonstrated by running hardware-in-the-loop simulations of the controller with online updates.

To summarise, given the primary objective stated above, the following specific objectives to achieve this were defined:

1. Develop a taxonomy for classifying intelligent control approaches that is agnostic to the specific control method used and quantifies how intelligent the control system is.

2. Develop a method to exploit conventional optimal control approaches in reinforcement learning when training a control system.

3. Develop a method for online reinforcement learning updates of an intelligent control system that can learn in uncertain environments.

4. Demonstrate online reinforcement learning updates of a control system using flight-suitable hardware.

With respect to each of these objectives, the main contributions of this thesis are as follows:

1. A novel taxonomy of intelligent control that identifies the three main dimensions of intelligence in these control systems and quantifies the level of intelligence in each dimension.

2. A framework for incorporating demonstrations in an off-policy reinforcement learning algorithm. Demonstration data can be provided by conventional optimal control approaches to improve the learned policy while also learning to handle uncertainties.

3. A novel method for online reinforcement learning updates based on theories from Extreme Learning Machines (ELMs) that can deal with online changes in the environment.

4. Results of online reinforcement learning updates run on Jetson Nano hardware that demonstrate the potential for this approach to run onboard spacecraft.

1.2 Research Output

Much of the work presented in this thesis is taken directly or adapted from previous publications produced as part of the PhD research. These publications are listed below along with the relevant chapters in which their content is used. Where applicable, the author's contributions are noted.

1.2.1 Peer-reviewed Journal Publications

- C. Wilson, F. Marchetti, M. Di Carlo, A. Riccardi, and E. Minisci, "*Classifying Intelligence in Machines: A Taxonomy of Intelligent Control*," Robotics, 2020. [27] (Chapter 3) **CW:** Conceptualisation, Methodology, Investigation, Writing - Original draft, Writing - Review and editing, Visualisation.
- C. Wilson and A. Riccardi, "*Enabling intelligent onboard guidance, navigation, and control using reinforcement learning on near-term flight hardware*," Acta Astronautica, 2022. [28] (Chapter 6) **CW:** Conceptualisation, Methodology, Software, Validation, Formal analysis, Writing - Original draft, Writing - Review and editing, Visualisation.
- C. Wilson and A. Riccardi, "*Improving the efficiency of reinforcement learning for a spacecraft powered descent with Q-learning*," Optimization and Engineering, 2023. [29] (Chapter 4) **CW:** Conceptualisation, Methodology, Software, Validation, Formal analysis, Writing - Original draft, Writing - Review and editing, Visualisation.

1.2.2 Peer-reviewed Conference Publications

- C. Wilson, F. Marchetti, M. D. Carlo, A. Riccardi, and E. Minisci, "*Intelligent Control: A Taxonomy*," in 8th International Conference on Systems and Control (ICSC). IEEE, 2019. [30]
- C. Wilson, A. Riccardi, and E. Minisci, "*A Novel Update Mechanism for Q-Networks Based On Extreme Learning Machines*," in 2020 International Joint Conference on Neural Networks (IJCNN). IEEE, 2020. [31]

1.2.3 Conference Publications

- C. Wilson and A. Riccardi, "*Leveraging Optimal Control Demonstrations in Reinforcement Learning for Powered Descent*," in 8th International Conference on Astrodynamics Tools and Techniques (ICATT), 2021. [32]
- C. Wilson and A. Riccardi, "*Enabling intelligent onboard guidance, navigation, and control using near-term flight hardware*," in 72nd International Astronautical Congress, 2021. [33]

1.3 Thesis Organisation

Chapter 2 gives the main theoretical background that is not novel but foundational for the work presented here. The primary areas covered are RL and Extreme Learning Machines (ELMs). In addition to the background presented in this chapter, it is recommended for readers of this thesis to have some knowledge of control systems and particularly optimal control. Chapter 3 briefly describes the evolution of modern control systems and defines IC based on literature. Given this context in the development of IC and its definition, a taxonomy is devised with the goal of classifying levels of intelligence in control systems. Results are shown for the application of this taxonomy to various IC systems.

Chapter 4 introduces the spacecraft powered descent problem that is used as a test case for all the methods presented in this thesis. This chapter shows how this problem is formulated as a RL problem that can be solved using existing state-of-the-art methods. Compared to previous works, the method used in this chapter aims to reduce the necessary training time to solve the problem. The results show that depending on how the state representation is defined in this problem, the learning process may fail to converge on a desirable policy. Chapter 5 proposes an approach for mitigating this behaviour using demonstrations. Data for these demonstrations come from running conventional optimal control approaches under some nominal conditions in the environment. The results illustrate that combining the optimal control demonstrations with RL can help to avoid learning undesirable policies. Chapter 6 adds online learning capabilities to the methods introduced previously such that they can handle more substantial uncertainties. Results are shown for cases where the environmental disturbances are different to those used in training, which demonstrates the effectiveness of online learning in adapting to such changes. Furthermore, results show that incorporating the optimal control approaches described previously for part of the descent greatly improves the soft-landing performance of the method. Chapter 7 concludes with recommendations for future work.

Chapter 2

Background

This chapter describes the relevant theoretical background for the content covered in the remainder of the thesis. Section 2.1 introduces the underlying theory of RL for optimal control. Following this, Section 2.2 describe more modern approaches to RL that incorporate function approximators. Finally, Section 2.3 presents an update mechanism for Neural Networks (NNs) called Extreme Learning Machine (ELM) that represents an alternative to gradient descent methods.

2.1 Reinforcement Learning for Optimal Control

Thanks to several major advancements in the 2010s, RL received a great deal of renewed interest. However, this subset of machine learning has a longer history dating back to the mid-20th century. The field of RL is considered to begin with Bellman's work on optimal control via dynamic programming [34]. This gave rise to the Bellman equation for optimality in sequential decision problems, which can be solved via RL. In 1998, Sutton and Barto published the seminal work "Reinforcement Learning: An Introduction" which describes in detail the theory behind RL [10]. This section provides the relevant background to RL and the approaches used to apply RL in this work.

2.1.1 Reinforcement Learning Problems

Reinforcement Learning can refer to certain types of problems, methods used to solve such problems, and the broader field of research into these problems. Here, RL shall refer to problems that are solved in a certain manner. RL problems consist of an agent that interacts with an environment with the goal of maximising its total reward. “Agent” refers to the machine that can take control actions and is analogous to a conventional “controller”. As with many conventional control systems, “environment” refers to that which is being controlled.

The three key components of a RL problem are:

- *State*: what the agent observes from the environment, e.g. sensor measurements, images, or extracted features such as positions and velocities.
- *Action*: how the agent affects the environment. Similarly to conventional control problems, examples of actions include applied forces, movement directions, or binary decisions.
- *Reward*: what the agent aims to maximise. Certain problems have clear rewards, such as a win or loss in a game, but others can be more challenging to define the reward.

The state space refers to all possible environment states and the action space refers to all possible actions the agent can take in the environment. In most RL problems, the agent observes states and takes actions in a sequential manner at discrete timesteps, denoted t. The reward function defines the reward received by the agent at a given timestep. Figure 2.1 illustrates this agent-environment interaction. At each timestep, the agent takes an action a_t based on the current state s_t and then receives a reward r_{t+1} and observes the new state s_{t+1}.

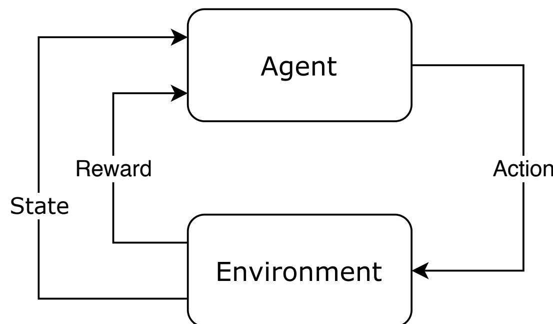


Figure 2.1: Agent-environment interaction in reinforcement learning. The agent observes a state and reward signal from the environment and uses this information to select an action that affects the environment.

The agent's goal is to maximise its return, or specifically its total (cumulative) discounted reward G_t following time-step t, where

$$\begin{aligned} G_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \end{aligned} \tag{2.1}$$

The discount factor, γ controls the priority given to longer term rewards and has a value $0 \leq \gamma \leq 1$. As $\gamma \rightarrow 1$, longer term rewards have more effect on the return and when $\gamma = 0$ only the reward at the following timestep influences the return.

In some environments, there are terminal states that indicate success or failure of a task. For example, games that can end in either a win or loss for the agent. Problems in such environments are referred to as episodic tasks. These episodes run from timestep $t = 0$ to $t = T$, at which point the environment may reset to an initial state, like starting a new game. The terminal state is denoted s_T. Other problems that do not have defined terminal states are called continuing tasks. In this case, the return in Equation 2.1 is finite only when $\gamma < 1$ and the reward is bounded. All the environments investigated here are considered episodic and have defined terminal states.

As stated above, RL problems typically involve sequential decision making processes where actions influence the reward received and the following state. In a Markov Decision Process (MDP), the probability that the next state s_{t+1} following state s_t is state s and the next reward r_{t+1} is r depends only on the current state s_t and the

action a_t taken by the agent. This is called the Markov property and can be expressed symbolically as

$$Pr\left\{s_{t+1} = s, r_{t+1} = r | s_t, a_t\right\}. \tag{2.2}$$

The assumption that this property holds underlies much of the fundamental analysis of RL algorithms. However, when using RL with function approximators, as will be introduced later, the Markov property is not necessarily required [10].

2.1.2 Policies and Value Functions

A policy, denoted π, defines the agent's mapping from states to actions. As the agent interacts with the environment, it updates its policy depending on the observations from the environment. In a MDP, the agent's actions affect the immediately observed reward and can also affect the availability of future rewards. As stated above, the agent's goal is to maximise cumulative reward and not only individual rewards. Therefore, assessing how desirable a certain state is to the agent requires determining the expected return when in this state. This concept is captured in the value function, which is the expected return in a given state from following a policy. The state value function v_π of policy π is defined as

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\left[G_t | s_t = s\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right], \end{aligned} \tag{2.3}$$

where $\mathbb{E}_\pi$ denotes the expected value when following policy π. Similarly, the action-value function is the expected value of taking an action in a state. This is denoted q_π and defined as

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi\left[G_t | s_t = s, a_t = a\right] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right]. \end{aligned} \tag{2.4}$$

A policy that maximises return is an optimal policy, denoted π^*. By definition, the value function of an optimal policy is greater than or equal to that of any other policy

for all states. The optimal policy is not unique for a given RL problem and there may be several optimal policies for one problem. On the other hand, the optimal state-value function v_* is unique for a given problem and is the value function of an optimal policy. The optimal state-value function can be expressed as

$$v_*(s) = \max_\pi v_\pi(s). \tag{2.5}$$

Likewise, the optimal action-value function q_* is unique for a given problem and can be expressed as

$$q_*(s, a) = \max_\pi q_\pi(s, a). \tag{2.6}$$

Agents store estimates of the value function or action-value function that are denoted $V(s)$ and $Q(s, a)$ respectively. If the optimal action-value function is known, i.e. $Q(s, a) = q_*(s, a)$, the optimal policy can be found by always taking actions that maximise the action-value function for any state. This is referred to as taking "greedy" actions.

While the agent is still learning and does not know the optimal action-value function, it is beneficial to experiment with different actions to see which gain more reward. However, this conflicts with the agent's overall goal of maximising its cumulative reward, which motivates taking more greedy actions. This conflict is commonly referred to as the problem of exploration versus exploitation. Exploration is where an agent takes random, exploratory actions to find better policies and exploitation is taking greedy actions to maximise reward. One class of policies that can be used to balance exploration and exploitation is ϵ-greedy policies. These use a parameter ϵ, where $0 \leq \epsilon \leq 1$, to control the amount of exploration. At each timestep the agent will take a random action with probability ϵ, otherwise it takes a greedy action. Usually, random actions are sampled with equal probability from the action space, but this can be adjusted in cases where random actions could be detrimental to the agent.

Most RL approaches to solve a MDP involve two main processes: policy evaluation and policy improvement. Policy evaluation refers to an agent updating its value function to be consistent with the true value function of the current policy. Policy improvement

is how an agent updates its policy towards the optimal policy via greedy actions with respect to the value function. The use of these two processes in an alternating fashion is referred to as Generalised Policy Iteration (GPI). To some extent, like with exploration versus exploitation, these compete with each other; updating the policy creates a moving target for value function updates. Nevertheless, this is a key concept underpinning most algorithms in RL.

Approaches to GPI fall into two broad classes of on-policy and off-policy. In an on-policy algorithm the agent updates estimated value functions (policy evaluation) directly for the policy being followed. This has the drawback that, while the agent is learning, the policy takes exploratory actions that are not optimal. As a result, the policy evaluation will always occur for a sub-optimal policy. Instead, policy evaluation for off-policy algorithms can use any arbitrary policy. In practice, this can also be the agent's current policy as with on-policy. However, off-policy methods lend themselves well to incorporating demonstrations from other policies when training.

2.1.3 Q-Learning

One of the significant early developments in RL research was the Q-learning algorithm developed by Watkins [35]. This is a type of Temporal Difference (TD)-learning algorithm—a class of methods that are well suited to RL problems. In TD-learning, the agent updates estimates of the value function, $V(s)$ or $Q(s,a)$, based on experience, i.e. the sequences of states, actions, and rewards observed by the agent. The Q-learning algorithm directly approximates the optimal action-value function q_* in the policy evaluation.

In its original form, Q-learning updates action-value estimates for the immediately observed state transition consisting of the state s_t, action a_t, observed state s_{t+1}, and observed reward r_{t+1}. This is called one-step Q-learning. The update rule for one-step Q-learning can be expressed as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]. \tag{2.7}$$

In this equation, α is a step size parameter that controls the rate of updates to Q. This is necessary since the "max" operator can lead to over-estimation of action-values. The expression in square brackets is referred to as the Temporal Difference (TD)-error. The TD-error at timestep t is denoted δ_t and defined for Q-learning as

$$\delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t), \tag{2.8}$$

where $Q(s_T, a) = 0$ for all a. Through the process of GPI, the agent aims to reduce the TD-error to zero over the course of training. In practice, TD-learning methods can store estimates of action values in a table with an entry for each state or state-action pair. Tabular Q-learning updates the action-value estimates in its table via Equation 2.7. Work by Watkins and Dayan proved that the action-value estimate $Q(s, a)$ in Q-learning converges to the optimal action-value q_* provided every state-action pair updates throughout training [36]. The policy used for training can therefore be any policy derived from Q that allows exploration, such as an ϵ-greedy policy. Another important feature of Q-learning is that it is an off-policy method. This means that the policy evaluation (Equation 2.7) is independent from the agent's own policy and can therefore use an arbitrary policy.

2.2 Reinforcement Learning with Function Approximation

Tabular methods of solving RL problems described above have several limitations. Since they require a discrete state space, the state space size increases dramatically with the dimensionality of the state. This is known as the curse of dimensionality. Problems with high dimensional states therefore require large tables for storing values or action-values. This also has implications for the agent's exploration, since in such problems there are many states to explore and it is difficult to visit all of them regularly. The main approach used to overcome these limitations is the use of function approximators [37]. These can work with continuous state spaces thereby reducing the need for large tables. Moreover, this allows the agent to exploit similarities in regions of the state space to learn more efficienctly. The main drawback of using function approximators is that these do not

always possess the same guarantee of convergence as with tabular methods. In addition, they have many additional hyperparameters that must be carefully selected to obtain desired performance. Nevertheless, approaches to RL with function approximation have proven to be very powerful in many challenging environments.

2.2.1 Policy and Value Gradient Methods

A function approximator has a parameter vector denoted $\boldsymbol{\theta}$ that determines the mapping from inputs to outputs. Instead of storing value estimates in a table, function approximators can express the relationship between states and values in terms of its parameters. Q_θ denotes the estimated action-value function for parameters $\boldsymbol{\theta}$. This function can also be expressed as

$$Q_\theta(s, a) = \hat{q}(s, a, \boldsymbol{\theta}), \tag{2.9}$$

where $\hat{q}$ is a function in terms of the state, action, and parameters $\boldsymbol{\theta}$. Value gradient methods aim to minimise the TD-error (Equation 2.8 for Q-learning) by taking the gradient of the parameters $\boldsymbol{\theta}$ with respect to the TD-error, or some function of this, and performing gradient descent on the parameters. Gradient descent refers to updating the parameters by moving them in a direction with negative gradient. This simple idea is very effective for training function approximators in supervised learning and has also found utility for RL.

An alternative to parameterising the value function is to use function approximation to express the policy π in terms of the parameters $\boldsymbol{\theta}$. In this case, as with general approaches to RL, the policy aims to maximise its return and so performs gradient ascent in terms of the parameters $\boldsymbol{\theta}$. These methods are referred to as policy gradient methods. Some policy gradient algorithms will also use a value function estimate to update the policy, which are commonly known as actor-critic methods. Both policy- and value-gradient methods remain widely used for tackling RL problems. Other methods also combine the concepts of policy and value gradients for a more general approach to solving RL via gradient descent [38].

2.2.2 Q-Networks

Most state-of-the-art approaches to RL use Neural Networks (NNs) for function approximation. This is partly due to the significant developments in NNs in other supervised learning applications. Since NNs can approximate any nonlinear function, this makes them suitable for estimating value functions [37]. The term “Q-network” refers here to a NN that approximates an action-value function for Q-learning. The parameters $\boldsymbol{\theta}$ of a Q-network are all the parameters that can be updated when training the network.

Improvements in computational power for running large models gave rise to “deep-learning” methods that take raw, unprocessed inputs, such as images, and create abstract representations of these by passing through a multi-layered NN, also known as a Deep Neural Network (DNN). This idea of taking raw inputs and using DNNs for function approximation saw a breakthrough in RL with the development of the Deep Q-Networks (DQN) algorithm from DeepMind [11]. This uses image pixels as state inputs to estimate action-values with a DNN. By training on these “raw” states of images without feature extraction, DQN showed state-of-the-art performance in the benchmark Atari games environments [39].

In addition to the use of larger networks, the DQN algorithm incorporates several features to improve its performance. One-step Q-learning only performs updates on the most recently observed state transition at every timestep. This has the limitation of rarely updating the policy from infrequent state transitions that could contain useful information for the agent. Furthermore, it is data inefficient to perform only one update at each timestep. These issues are addressed in DQN by storing “experiences” of state transitions in an agent’s replay memory. The agent then updates using minibatches of experiences sampled from this memory. This concept is known as “experience replay” and is useful for off-policy RL methods.

The agent’s replay memory is denoted $\mathcal{D}$ and contains for each timestep, t, tuples of $(s_t, a_t, r_{t+1}, s_{t+1})$, or experiences, that define a state transition. To update the network parameters $\boldsymbol{\theta}$, the agent samples a minibatch of k experiences $\left(s_j, a_j, r_j, s'_j\right)$, $j = 1, 2, \ldots, k$, where for each sampled transition $s_j = s_t$, $a_j = a_t$, $r_j = r_{t+1}$, and $s'_j = s_{t+1}$. In DQN, experiences are sampled with equal probability, however new approaches to

prioritised sampling have also been proposed [40]. The parameters update based on the gradient of the squared TD-error for each state transition. To avoid the replay memory becoming too large, its size is limited to a maximum number of experiences. Old experiences are then removed on a first-in-first-out basis.

Another limitation of Q-networks is that they are inherently unstable and weight updates tend to diverge causing an "explosion" in weights. DQN mitigates this issue by using a separate network for approximating target action-values that is referred to as a "target network". The target network's parameters are initially identical to the "prediction" Q-network then held constant while the prediction network updates. As the prediction network updates, the agent periodically copies its parameters to the target network. This means the target network also tends towards the optimal action-value function while the weight updates are decoupled from the target prediction. The parameters of the target network are denoted $\boldsymbol{\theta}^{-}$. The TD-error used to update the Q-network from Equation 2.8 can then be written as

$$\delta_j = r_j + \gamma \max_a Q_{\theta^-}(s'_j, a) - Q_\theta(s_j, a_j), \tag{2.10}$$

for transition j in a minibatch where as before, $Q(s_T, a) = 0$ for all a.

The final notable feature of the DQN is how it trades off exploration and exploitation using an ϵ-greedy policy. It is sensible for an agent to explore more at the start of its training and decrease the amount of exploration as its Q estimates get closer to the optimal action-value function. This can be achieved by decreasing the exploration probability ϵ as the number of episodes (or timesteps for continuing tasks) increases. A simple way to do this is to linearly decrease ϵ from an initial value of ϵ_0 to ϵ_f over N_ϵ episodes after which point $\epsilon = \epsilon_f$. The value of ϵ after ep episodes is then given by

$$\epsilon = \max \left\{ \epsilon_0 - \frac{ep}{N_\epsilon} \left(\epsilon_0 - \epsilon_f \right), \epsilon_f \right\}. \tag{2.11}$$

Deep Q-Networks Algorithm

Algorithm 1 shows pseudocode for an implementation of the DQN algorithm. As stated above, the policy π can be any policy that allows exploration and is usually an ϵ-greedy

policy. The hyperparameter C controls how often the prediction network parameters are transferred to the target network.

Algorithm 1 Deep Q-Networks algorithm

Inputs: number of training episodes N_{ep}, agent hyperparameters
1: initialise network parameters $\boldsymbol{\theta}$ with random weights
2: **for** $ep = 1$ to N_{ep} **do**
3: initialise state $s_t \leftarrow s_0$
4: **while** state s_t is non-terminal **do**
5: select action a_t according to policy π
6: execute action a_t and observe r_{t+1}, s_{t+1}
7: update memory $\mathcal{D}$ with $(s_t, a_t, r_{t+1}, s_{t+1})$
8: select random minibatch of k experiences (s_j, a_j, r_j, s'_j) from $\mathcal{D}$
9: $\delta_j = r_j + \gamma \max_a Q_{\theta^-}(s'_j, a) - Q_\theta(s_j, a_j) \; \forall \; j$ in minibatch
10: update parameters $\boldsymbol{\theta}$ using TD-error δ_j for each experience
11: after C time-steps set $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
12: **end while**
13: **end for**

2.3 Extreme Learning Machines

Despite the wealth of research and growing interest in NNs, most applications use the same underlying principle of gradient descent for training. This is an optimisation procedure that minimises a cost by following a negative gradient. When training a NN, this cost is often related to the difference between the predicted and desired output. While this simple idea of gradient descent has proven to be powerful, it has limitations. Most notably, gradient descent is very susceptible to converging on local minima and requires many iterations of training.

A proposed alternative to iterative methods of tuning NN weights is Extreme Learning Machine (ELM) [41]. In this method of updating, weights are calculated using an algebraic method instead of calculating gradients. ELM has several advantages over other update rules for certain types of problems. The main advantage is that it can be considerably faster than iterative methods for optimising network weights since they are instead calculated analytically. The ELM algorithm has seen many improvements and adaptations since its inception that have occurred alongside broader developments

in neural network architectures. For example, ELM has been applied to training deep networks [42] and convolutional networks [43]. More recent work has combined theories of ELMs with those of physics informed neural networks [44] and their Bayesian variants [45]. This section describes the underlying theory of ELM and its developments that are relevant in the remainder of this thesis.

2.3.1 Extreme Learning Machine Theory

In its most widely used form, ELM is an update method for Single-Layer Feedforward Networks (SLFNs) that are initialised with random input weights and biases. The equations in the following derivation of ELM come from [41] with some notational changes for consistency. Consider an arbitrary set of N samples of training data $(\mathbf{x}_j, \mathbf{y}_j)$ where $\mathbf{x}_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]$, $\mathbf{y}_j = [y_{j1}, y_{j2}, \ldots, y_{jm}]$, and n and m are the input and output sizes respectively. A SLFN can be mathematically modelled as

$$\sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g\left(\mathbf{w}_i \cdot \mathbf{x}_j + b_i\right) = \mathbf{o}_j, \;\; j = 1, 2, \ldots, N, \tag{2.12}$$

where $\tilde{N}$ is the number of hidden nodes, $\boldsymbol{\beta}_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^\top$ is the output weight vector which connects the ith hidden node to the output nodes, g is the activation function, $\mathbf{w}_i = [w_{i1}, w_{i2}, ..., w_{in}]^\top$ is the input weight vector which connects the ith hidden node to the input nodes, b_i is the bias of the ith hidden node, and $\mathbf{o}_j$ is the output of the SLFN for input $\mathbf{x}_j$. Figure 2.2 shows this transformation from input to output in a SLFN graphically. In this example, the network has an input size of $m = 3$, an output size of $n = 2$, and $\tilde{N} = 4$ hidden nodes in the hidden layer.

The error between the output $\mathbf{o}_j$ and the targets $\mathbf{y}_j$ is denoted $\mathbf{e}_j$ and defined by

$$\mathbf{e}_j = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g\left(\mathbf{w}_i \cdot \mathbf{x}_j + b_i\right) - \mathbf{y}_j, \;\; j = 1, 2, \ldots, N. \tag{2.13}$$

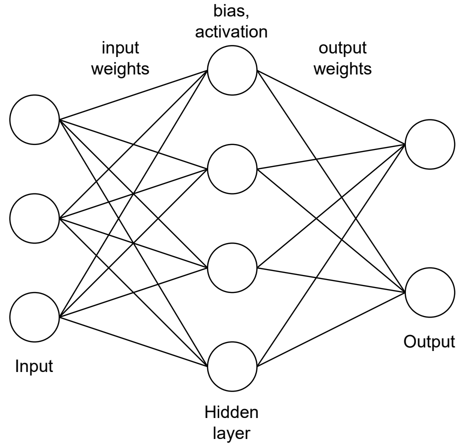


Figure 2.2: Diagram of a single-layer feedforward network indicating the relevant parameters defining the transformation.

In the case where the network output $\mathbf{o}_j$ has zero error compared to the targets $\mathbf{y}_j$ for all N samples, i.e. $\sum_{j=1}^{N} \|\mathbf{e}_j\| = 0$, it can be written that

$$\sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g\left(\mathbf{w}_i \cdot \mathbf{x}_j + b_i\right) = \mathbf{y}_j, \;\; j = 1, 2, \ldots, N. \tag{2.14}$$

Writing this in a more compact form gives

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T}, \tag{2.15}$$

where $\mathbf{H}$ is the hidden layer output matrix, $\boldsymbol{\beta}$ is the output weight vector matrix, and $\mathbf{T}$ is the target matrix. These are defined as shown:

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \tag{2.16}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^\top \\ \vdots \\ \boldsymbol{\beta}_{\tilde{N}}^\top \end{bmatrix}_{\tilde{N}\times m}, \quad (2.17) \qquad \mathbf{T} = \begin{bmatrix} \mathbf{y}_1^\top \\ \vdots \\ \mathbf{y}_N^\top \end{bmatrix}_{N\times m}. \quad (2.18)$$

ELM calculates the output weight matrix by solving the linear system defined in Equation 2.15 for $\boldsymbol{\beta}$. The calculated weights, denoted $\hat{\boldsymbol{\beta}}$, are given by

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T}, \tag{2.19}$$

where $\mathbf{H}^\dagger$ denotes the Moore-Penrose generalised inverse of $\mathbf{H}$, defined as shown:

$$\mathbf{H}^\dagger = \left(\mathbf{H}\mathbf{H}^\top\right)^\dagger \mathbf{H}^\top. \tag{2.20}$$

This is used since, in general, $\mathbf{H}$ is not a square matrix and so cannot be inverted directly.

The approach of ELM for calculating output weights has some benefits compared to other gradient-based methods of updating neural networks. It is proven in [41] that $\hat{\boldsymbol{\beta}}$ is the smallest norm least squares solution for $\boldsymbol{\beta}$ in the linear system defined by Equation 2.15, which is not always the solution reached using other updates. ELM also avoids several common issues in gradient descent such as converging to local minima and improper learning rates.

2.3.2 Regularized Extreme Learning Machine

In the statistical learning theory of Vapnik [46], there are two different forms of "risk" minimisation: Empirical Risk Minimisation (ERM) and Structural Risk Minimisation (SRM). ERM refers to reducing the prediction error between the output of a model and training data. On the other hand, the principle of SRM creates a trade-off between the model's prediction accuracy and complexity. The standard form of ELM detailed above considers only ERM. Later work extended the theories of ELM to balance ERM and SRM [47]. The resulting algorithm, Regularized ELM, is less sensitive to outliers in training data and not as prone to overfitting as ELM. The equations in the

following derivation of Regularized ELM come from [47] with some notational changes for consistency.

The empirical risk is quantified by the sum of errors squared $||\mathbf{e}||^2$, where $\mathbf{e}$ is defined as in Equation 2.13. The structural risk is quantified by $||\boldsymbol{\beta}||^2$. Regularized ELM solves the optimisation problem

$$\begin{aligned} \underset{\boldsymbol{\beta}}{\text{minimise}} \quad & \frac{1}{2}||\boldsymbol{\beta}||^2 + \frac{1}{2}\bar{\gamma}||\mathbf{e}||^2 \\ \text{subject to} \quad & \mathbf{e}_j = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i g\left(\mathbf{w}_i \cdot \mathbf{x}_j + b_i\right) - \mathbf{y}_j, \\ & j = 1, 2, \ldots, N, \end{aligned} \tag{2.21}$$

where $\bar{\gamma}$ is a regularisation parameter that controls the balance of ERM and SRM. The approach to solve 2.21 uses the following Lagrangian:

$$\begin{aligned} L(\boldsymbol{\beta}, \mathbf{e}, \boldsymbol{\alpha}) &= \frac{1}{2}\bar{\gamma}||\mathbf{e}||^2 + \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{j=1}^{N} \lambda_j \left(\sum_{i=1}^{\tilde{N}} \beta_i g\left(\mathbf{w}_i \cdot \mathbf{x}_j + b_i\right) - \mathbf{y}_j - \mathbf{e}_j \right) \\ &= \frac{1}{2}\bar{\gamma}||\mathbf{e}||^2 + \frac{1}{2}||\boldsymbol{\beta}||^2 - \boldsymbol{\lambda}\left(\mathbf{H}\boldsymbol{\beta} - \mathbf{T} - \mathbf{e}\right), \end{aligned} \tag{2.22}$$

where $\lambda_j \in \mathbb{R}$, $j = 1, 2, \ldots, N$ is the Lagrangian multiplier. Finding the points at which the gradients of L with respect to $\boldsymbol{\beta}$, $\mathbf{e}$, and $\boldsymbol{\lambda}$ are zero gives the following optimality conditions for 2.21:

$$\frac{\partial L}{\partial \boldsymbol{\beta}} = \boldsymbol{\beta}^\top - \boldsymbol{\lambda}\mathbf{H} = 0, \tag{2.23}$$

$$\frac{\partial L}{\partial \mathbf{e}} = \bar{\gamma}\mathbf{e}^\top + \boldsymbol{\lambda} = 0, \tag{2.24}$$

$$\frac{\partial L}{\partial \boldsymbol{\lambda}} = \mathbf{H}\boldsymbol{\beta} - \mathbf{T} - \mathbf{e} = 0. \tag{2.25}$$

Substituting 2.25 into 2.24 gives

$$\boldsymbol{\lambda} = -\bar{\gamma}(\mathbf{H}\boldsymbol{\beta} - \mathbf{T})^\top \tag{2.26}$$

and then substituting 2.26 into 2.23 gives the solution for the output weights

$$\boldsymbol{\beta} = \left(\frac{I}{\bar{\gamma}} + \mathbf{H}^\top\mathbf{H} \right)^\dagger \mathbf{H}^\top\mathbf{T}. \tag{2.27}$$

When $\bar{\gamma} \rightarrow \infty$, Equation 2.27 becomes the conventional ELM as in Equation 2.19. Compared to conventional ELM, regularized ELM has an extra hyperparameter to tune, $\bar{\gamma}$, but still maintains many of the advantages of ELM over gradient-based updates.

2.3.3 Incremental Extreme Learning Machine

Solving for the output weights via ELM or Regularized ELM requires all the training data to be available at once. More often, training samples arrive sequentially in "chunks". Such is the case for RL, which often uses regular updates performed on minibatches of data. A further enhancement to the ELM algorithm proposes incremental updates [48]. This is suitable for sequential training data and, therefore, RL updates. The algorithm used is referred to as Least Squares Incremental Extreme Learning Machine (LS-IELM). The equations in the following derivation of LS-IELM come from [48] with some notational changes for consistency.

Sequential updates occur at discrete timesteps t. From Equation 2.27, the output weights $\boldsymbol{\beta}$ at timestep t can be expressed as

$$\boldsymbol{\beta}_t = \left(\frac{I}{\bar{\gamma}} + \mathbf{H}_t^\top \mathbf{H}_t\right)^\dagger \mathbf{H}_t^\top \mathbf{T}_t, \tag{2.28}$$

where $\mathbf{H}_t$ and $\mathbf{T}_t$ are given by Equations 2.16 and 2.18. These are the hidden layer output matrix and target matrix for all data up to time t. To simplify the notation, define A_t as

$$A_t = \frac{I}{\bar{\gamma}} + \mathbf{H}_t^\top \mathbf{H}_t \tag{2.29}$$

such that Equation 2.28 can be rewritten as

$$\boldsymbol{\beta}_t = A_t^\dagger \mathbf{H}_t^\top \mathbf{T}_t. \tag{2.30}$$

At time $t = 0$, the algorithm initialises $\boldsymbol{\beta}_t$ with an initial set of training data. These training data give the initial hidden layer output matrix, $\mathbf{H}_{t=0}$ and initial target matrix $\mathbf{T}_{t=0}$ from which $\boldsymbol{\beta}_t$ and A_t are initialised by the equations

$$\boldsymbol{\beta}_{t=0} = A_{t=0}^\dagger \mathbf{H}_{t=0}^\top \mathbf{T}_{t=0}, \tag{2.31}$$

$$A_{t=0} = \frac{I}{\bar{\gamma}} + \mathbf{H}_{t=0}^{\top}\mathbf{H}_{t=0}. \tag{2.32}$$

Suppose new sets of training data arrive in chunks of k samples. Let N denote the number of training samples used to update the network up to timestep t. The hidden layer output matrix and targets for a new set of k samples are denoted $\mathbf{H}_{IC}$ and $\mathbf{T}_{IC}$ and defined as

$$\mathbf{H}_{IC} = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_{N+1} + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_{N+1} + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_{N+k} + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_{N+k} + b_{\tilde{N}}) \end{bmatrix}_{k \times \tilde{N}}, \tag{2.33}$$

$$\mathbf{T}_{IC} = \begin{bmatrix} \mathbf{y}_{N+1}^{\top} \\ \vdots \\ \mathbf{y}_{N+k}^{\top} \end{bmatrix}_{k \times m}. \tag{2.34}$$

The matrices $\mathbf{H}_{t+1}$ and $\mathbf{T}_{t+1}$ can then be written as

$$\mathbf{H}_{t+1} = \begin{bmatrix} \mathbf{H}_t \\ \mathbf{H}_{IC} \end{bmatrix}_{(N+k) \times \tilde{N}}, \tag{2.35}$$

$$\mathbf{T}_{t+1} = \begin{bmatrix} \mathbf{T}_t \\ \mathbf{y}_{IC} \end{bmatrix}_{(N+k) \times m}. \tag{2.36}$$

From Equation 2.30, the output weights $\boldsymbol{\beta}$ at timestep $t+1$ can be expressed as

$$\boldsymbol{\beta}_{t+1} = A_{t+1}^{\dagger}\mathbf{H}_{t+1}^{\top}\mathbf{T}_{t+1}, \tag{2.37}$$

where

$$A_{t+1} = \frac{I}{\bar{\gamma}} + \mathbf{H}_{t+1}^{\top}\mathbf{H}_{t+1}. \tag{2.38}$$

Substituting Equation 2.35 into Equation 2.38 gives

$$A_{t+1} = \frac{I}{\bar{\gamma}} + \begin{bmatrix} \mathbf{H}_t^{\top} & \mathbf{H}_{IC}^{\top} \end{bmatrix} \begin{bmatrix} \mathbf{H}_t \\ \mathbf{H}_{IC} \end{bmatrix}^{\dagger}. \tag{2.39}$$

Following matrix multiplication this becomes

$$A_{t+1}^{\dagger} = (A_t + \mathbf{H}_{IC}^{\top}\mathbf{H}_{IC})^{\dagger}, \tag{2.40}$$

which can be expanded to

$$(A_t + \mathbf{H}_{IC}^{\top}\mathbf{H}_{IC})^{\dagger} = A_t^{\dagger} - A_t^{\dagger}\mathbf{H}_{IC}^{\top}\left(\mathbf{H}_{IC}A_t^{\dagger}\mathbf{H}_{IC}^{\top} + I_{k\times k}\right)^{\dagger}\mathbf{H}_{IC}A_t^{\dagger}. \tag{2.41}$$

This allows Equation 2.37 to be rewritten as

$$\boldsymbol{\beta}_{t+1} = \left[A_t^{\dagger} - A_t^{\dagger}\mathbf{H}_{IC}^{\top}\left(\mathbf{H}_{IC}A_t^{\dagger}\mathbf{H}_{IC}^{\top} + I_{k\times k}\right)^{\dagger}\mathbf{H}_{IC}A_t^{\dagger}\right]\begin{bmatrix}\mathbf{H}_t^{\top} & \mathbf{H}_{IC}^{\top}\end{bmatrix}\begin{bmatrix}\mathbf{T}_t \\ \mathbf{T}_{IC}\end{bmatrix}. \tag{2.42}$$

To further simplify the notation, define K_t as

$$K_t = I - A_t^{\dagger}\mathbf{H}_{IC}^{\top}\left(\mathbf{H}_{IC}A_t^{\dagger}\mathbf{H}_{IC}^{\top} + I_{k\times k}\right)^{\dagger}\mathbf{H}_{IC} \tag{2.43}$$

then substitute K_t into Equation 2.42 and expand to give the following:

$$\begin{aligned}\boldsymbol{\beta}_{t+1} &= K_t A_t^{\dagger}\begin{bmatrix}\mathbf{H}_t^{\top} & \mathbf{H}_{IC}^{\top}\end{bmatrix}\begin{bmatrix}\mathbf{T}_t \\ \mathbf{T}_{IC}\end{bmatrix} \\ &= K_t A_t^{\dagger}\mathbf{H}_t^{\top}\mathbf{T}_t + K_t A_t^{\dagger}\mathbf{H}_{IC}^{\top}\mathbf{T}_{IC}\end{aligned} \tag{2.44}$$

Substituting $\boldsymbol{\beta}_t$ from Equation 2.30 into this expression gives

$$\boldsymbol{\beta}_{t+1} = K_t\boldsymbol{\beta}_t + K_t A_t^{\dagger}\mathbf{H}_{IC}^{\top}\mathbf{T}_{IC} \tag{2.45}$$

and substituting K_t from Equation 2.43 into Equation 2.41 gives

$$A_{t+1}^{\dagger} = K_t A_t^{\dagger}. \tag{2.46}$$

Equations 2.45 and 2.46 describe the weight updates for LS-IELM. This method of updating allows sequential updates instead of training on all observed data at each timestep.

2.4 Summary

This chapter introduced the fundamental concepts for RL that are relevant to the methods applied in later chapters. The DQN algorithm shown in Algorithm 1 forms the basis for these methods. As will be shown in Chapter 4, its discrete action space and use of replay memory means it can learn effective policies in fewer environment interactions compared to other methods. Since DQN is an off-policy algorithm, this also allows it to incorporate demonstrations as shown in Chapter 5. Sequential forms of ELM as described above are an alternative method of updating Q-networks. Compared to conventional gradient-based updates, EQLM provides an efficient approach to update online in response to a changing environment as will be shown in Chapter 6.

Chapter 3

A Taxonomy of Intelligent Control

Intelligent Control (IC) has received a great deal of attention in many control applications since the term was first coined by Fu [49]. Due to its wide use, a "terminology war" ensued where there were several competing definitions for different concepts within IC—especially "adaptive" and "learning" control [50]. The works of Saridis and Antsaklis provided clear definitions of IC and established what control systems can, and cannot, be classified as "intelligent" [51, 52]. Under these definitions, there are still many different types of control systems that fall under the category of IC with varying levels of complexity. The goal of the work presented in this chapter is to quantify the similarities and differences between these control systems by categorising the different levels of intelligence of IC systems.

When considering the intelligence of machines, it is clear that "intelligence" cannot be a binary label but rather a scale from non-intelligent to highly-intelligent. As pointed out in [50], it is hard to strictly define a threshold for something to be considered intelligent, as for some definitions even a thermometer could be classed as intelligent. Obviously, in this extreme example a thermometer possesses a very low level of intelligence—if any. The problem then becomes quantifying the level of intelligence of a control system.

From the initial development of control theory up to the present day, one key concept in control has been uncertainty. This is where some aspect of a system is unknown due to lack of knowledge or stochastic behaviour. As control systems developed over the past

century they have expanded their ability to autonomously deal with uncertainties to a point where they require human-like behaviours such as reasoning and learning. This can be considered a marker of intelligence [53]. A control problem where everything about the system is known can be very simple to solve using well established methods. Accounting for uncertainties makes this more difficult. This work suggests that the level of intelligence of a control system is related to the level of uncertainty at design time: more intelligent controllers cope with greater levels of uncertainty.

This chapter presents a review of literature related to IC, starting with a background to the evolution of control theories that results in IC. The existing literature establishes the scope of IC and how it relates to uncertainty in control systems. This gives rise to the view that the level of intelligence of a control system can be described in terms of levels of uncertainty within the control system. This is formalised in a taxonomy that can be applied to existing control systems.

Previous works have created other classifications of IC such as that of Krishnakumar [54]. This has four levels of intelligence that focus on a controller's ability to self-improve where each level in the classification is additive to the previous. In contrast, the taxonomy proposed here allows for different levels of intelligence in each of the dimensions of IC, which gives a more descriptive classification of these systems. An aerospace industry survey from the American Institute of Aeronautics and Astronautics defined six "stages of intelligent reasoning" for spacecraft [55]. These classifications are specific to spacecraft operations. The examples presented here demonstrate that the proposed taxonomy can be applied to any field that uses IC. One of the motivations for developing this taxonomy was to identify gaps in existing applications where future work should investigate to improve the level of intelligence in IC. Where existing classification methods fall short in this respect is the lack of a multidimensional view of "intelligence". As will be shown in later examples, some IC systems can have varying levels of intelligence in each dimension of the proposed taxonomy and some dimensions remain less explored than others.

Classifications of systems within the broader field of AI have received greater attention recently both for regulatory purposes and to aid discussion of different types of

intelligent systems [56]. For example, in 2024 the European Union passed into law the AI Act which specifies a hierarchy of AI systems that depends on their level of risk[1]. Whereas these classifications aim to measure the societal impacts of AI, the taxonomy presented here considers a more technical hierarchy for intelligent systems.

This chapter describes a novel taxonomy of IC methods. The main contributions presented here are:

- A unique view of IC in multiple dimensions that considers the level of uncertainty in different aspects of the control system.
- A taxonomy based on this multi-dimensional view of IC that categorises how intelligent a control system is in each dimension.
- Examples of the application of this taxonomy to highlight its applicability to different types of IC system.

The remainder of this chapter is organised as follows. Section 3.1 describes the development of control systems towards intelligent control with a particular focus on the ability to handle uncertainties. Based on definitions from literature, Section 3.2 describes key aspects defining IC and intelligent systems, from which the main dimensions of IC are extracted. These dimensions form the basis of a taxonomy of IC introduced in Section 3.3. Finally, Section 3.4 demonstrates the application of the taxonomy to relevant examples of IC systems.

3.1 Path to Intelligent Control

Control of dynamical systems has been of fundamental importance in engineering for centuries and remains a widely studied field. There are some historical examples of what can be considered "control systems" from as early as 2000 years ago that use instruments such as water level devices to control some aspect of a system [57]. However, the application often considered to be the earliest example of a control system that made the first step towards modern control theory is Watt's flyball governor. This device

[1] https://artificialintelligenceact.eu/the-act/

was conceived in the late 18th century to regulate the pressure in a steam engine by means of rotating masses. Later, Maxwell analysed these types of governor systems via their mathematical models expressed as differential equations [58]. This gave rise to the important concept in control theory of stability. Following these initial steps towards understanding how to control a system, in the early 20th century modern control theory developed rapidly and has continued to be expanded since. This section describes the progression of control systems over this time and how it leads towards the current definition of intelligent control.

At its most fundamental, a controller makes a system behave in a desired manner. To achieve this, a controller takes actions on the system through potentially several types of actuator. Clearly, a concept this broad is applicable to many different systems, such as electrical, economic, political, and even human systems. All share the characteristic of having some goal, defined abstractly or more concretely, and taking actions to achieve this. The simplest type of control system, illustrated by the block diagram in Figure 3.1, does not observe measurements from the system and decides what actions to take based only on the desired system state. This is called *open-loop control*. Prior to the development of more advanced systems, many controllers took this form of what Mayr refers to as "control by rigidly predetermined program" [59]. As suggested by this description, such systems have control laws entirely defined prior to operation. In general, this results in a controller that cannot cope with any uncertainties and would not be referred to as "intelligent".

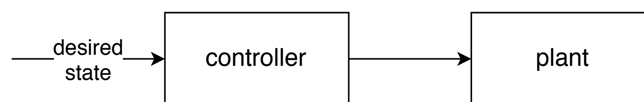


Figure 3.1: Open loop control system.

In most real systems, their mathematical models are imperfect. Where this is the case, it is difficult or impossible to design an open-loop control system that gives satisfactory performance. Therefore, most controllers use measurements of the system's state to dictate their control actions. This is *feedback control*, also known as closed loop control, as shown by the block diagram in Figure 3.2. Although control systems using

feedback existed for thousands of years, the formalisation of this kind of system did not happen until the early 20th century with the work of Nyquist [60]. Feedback is the foundation of most modern control systems. Of the many forms of feedback controller created, PID control remains one of the most common architectures that came to existence around the start of modern control theory [61]. Having feedback from the system can remove the requirement for a precise model and better handle uncertainties than an open-loop controller.

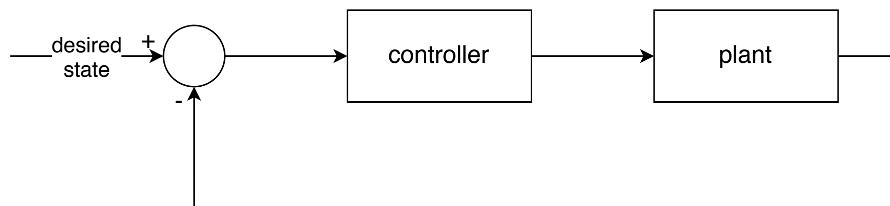


Figure 3.2: Feedback control system.

Within the realms of feedback control there are many different methods and architectures for designing controllers. In engineering this naturally leads to the question of how to create the “best” control system. The concept of creating a controller with the quantifiably best performance is encapsulated in the field of *optimal control*. Techniques from optimal control solve the problem of maximising or minimising a measurable characteristic of a dynamical system. This field has its origins from Bellman whose pioneering work in dynamic programming defined criteria for optimal systems [62]. In his work describing the development of optimal control, Bryson refers to it stemming from improvements in conventional control systems, where feedback control gains were optimised with respect to some cost function [63]. Figure 3.3 shows a block diagram of an optimal control system where, at this stage of development, the optimisation of the control system does not occur online using feedback from the real system.

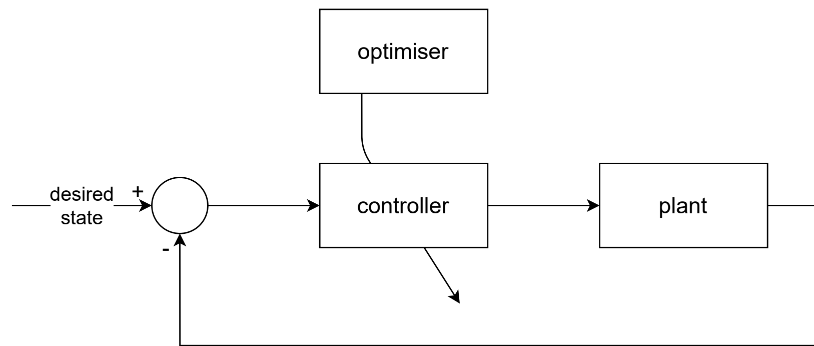


Figure 3.3: Optimal control system.

The types of control systems described above only deal with epistemic uncertainties that are due to a lack of knowledge of the system. Additional methods are required for control systems with aleatoric uncertainty that behave in a non-deterministic manner. These fall into the class of control systems known as *stochastic control*. In this case, uncertainties about the system are modelled as probability densities and cannot be specified as exact values [51]. As shown in Figure 3.4, uncertainties can be associated with different parts of the control system, such as disturbing forces or sensor noise.

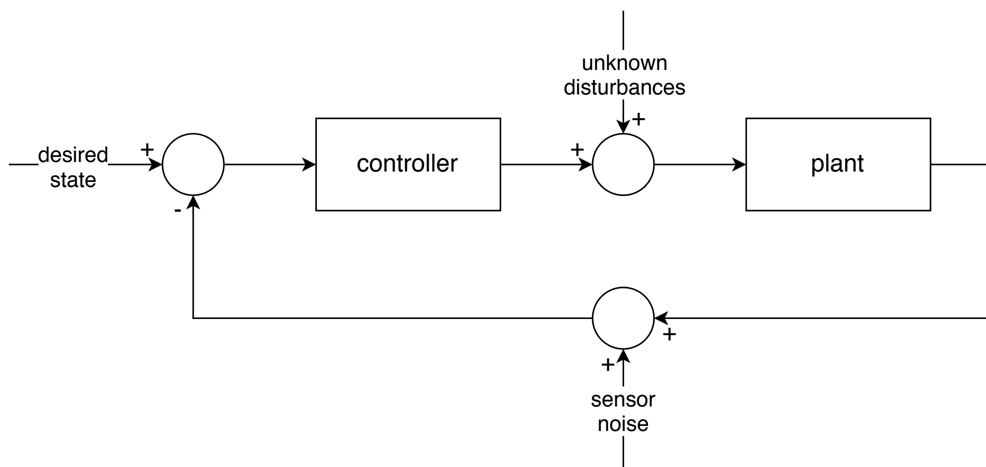


Figure 3.4: Stochastic control system.

At this point in the development of control systems, lack of system knowledge becomes a more significant issue. Most stochastic control systems still require reasonable

knowledge of the system dynamics and quantifiable uncertainties. Beyond such systems that can be modelled, there are classes of systems where the dynamics may change over time, which degrades the controller's performance. Furthermore, for certain systems the dynamics may be entirely unknown or incompletely known and so the methodologies discussed so far cannot be used. Approaches to solving these problems are referred to as *Self-Organising Control (SOC)* systems, which are simply defined as any system with features "beyond stochastic control systems" [7].

Adaptive control methods handle changing environments by adjusting the control scheme online. This means the controller maintains a favourable performance even as the environment varies. Adaptive controllers broadly fall under two categories: direct and indirect. Indirect adaptive control schemes do not alter the controller directly, but instead adapt other components that affect the control scheme, such as a system model. Direct adaptive control schemes adapt the controller parameters themselves. These can also be referred to as *Parameter-Adaptive* (indirect) or *Performance-Adaptive* (direct) SOC. Online adaptation is not only desirable in changing environments. Where the system being controlled is deterministic and "static", adaptive control can also be used to improve the controller performance online. Most often, however, adaptive control is used where the environment behaviour is broadly understood at design time, but subject to significant uncertainties, for example, in parameters of the mathematical model.

Learning control is the final step towards intelligent control methods. This can be seen as a more specific form of SOC where the controller retains information pertaining to the system's operation and uses this knowledge to alter its control scheme. The distinction between adaptive and learning control is often unclear in literature. This work considers the distinction that a learning system stores information about the system being controlled, whereas an adaptive learning system does not and instead adapts directly to observations. At this level, control systems can incorporate planning, where future actions are selected by the system in advance based on its knowledge. In addition, learning control represents a more behavioural approach towards control that aims to mimic human problem solving. Since learning is a fundamental aspect of human intelligence, the ability to learn about a system demonstrates a degree of

intelligence in that control system. Learning can occur offline such that the control system is trained before operation, or online where knowledge is accumulated during its operation. This distinction is important when it comes to intelligent control systems, as will be discussed. Figure 3.5 shows a block diagram for a generic SOC system where the adaptive or learning process occurs online.

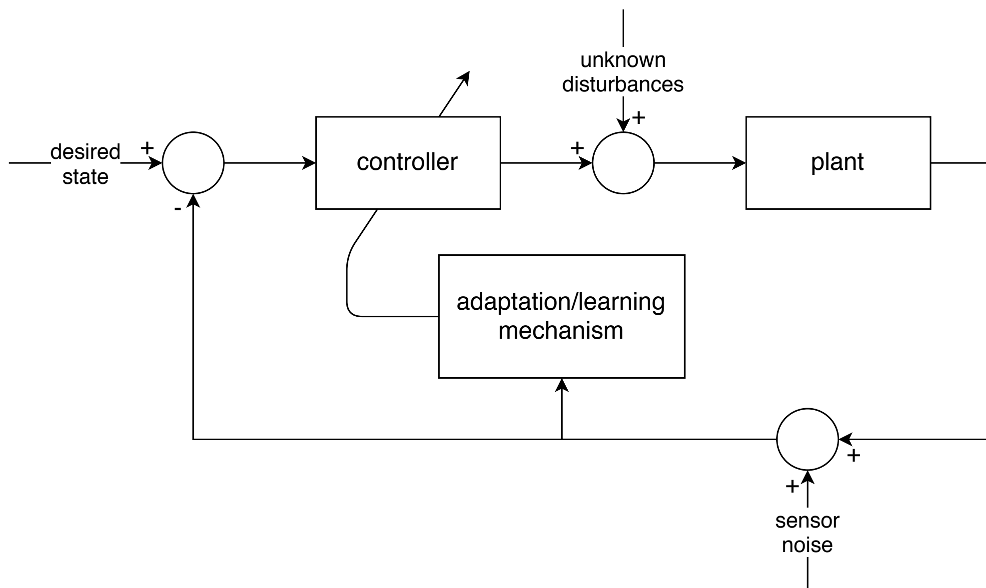


Figure 3.5: Adaptive or learning control system.

3.2 Defining Intelligent Control

As with many new concepts, since its introduction, the term "Intelligent Control" very quickly became widely used and often abused by many scholars both from the control community and wider fields. This made it difficult to create a suitable definition for IC since it was so commonly used to describe disparate concepts. As a result, in 1993 the IEEE Control Systems Society designated a task force to research and define "Intelligent Control" [52]. In their report they gave the following defining characteristics of IC systems:

> "An intelligent control system is designed so that it can autonomously achieve a high level goal, while its components, control goals, plant models and control laws are not completely defined, either because they were not known at the design time or because they changed unexpectedly."

This importantly shows that IC systems deal not only with uncertainties in the environment, or "plant models", but also cases where the controller does not have specifically defined goals or structures. Saridis gives a more general definition of IC as an interaction between three fields: Artificial Intelligence, Operations Research, and Automatic Control Systems (Figure 3.6) [7]. This builds on the definition given by Fu, who originally described IC as the "intersection of artificial intelligence and automatic control" [49].

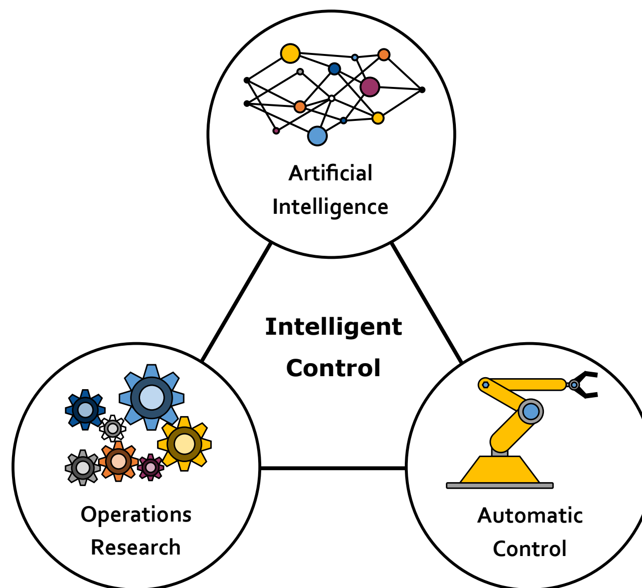


Figure 3.6: Intelligent control is the interaction between Artificial Intelligence, Operations Research, and Automatic Control Systems.

An often overlooked aspect of Fu's original definition is that IC lies at the *intersection* of these fields. This means that, on their own, theories from automatic control or artificial intelligence would not be classed as "intelligent" control. It is in their combination, potentially with theories from operations research, that IC is found. Considering

the definitions of conventional control methodologies presented previously, adaptive control systems and learning control systems can be classed as IC where they incorporate theories from AI. The task force report goes on to give the following definition of an intelligent system [52]:

> "An intelligent system must be highly adaptable to significant unanticipated changes, and so learning is essential. It must exhibit high degree of autonomy in dealing with changes. It must be able to deal with significant complexity, and this leads to certain sparse types of functional architectures such as hierarchies."

This definition highlights another essential aspect of IC systems, which is that they must adapt or learn from changes in the environment. Therefore, control systems that use theories from automatic control and AI may not be considered "intelligent" control without some means of updating the control system. This definition also makes reference to the fact that hierarchical structures are common in IC systems, as will be shown in later examples.

3.2.1 Methods for Intelligent Control

The methods comprising the three fields of artificial intelligence, operations research, and automatic control systems are broad ranging. Nevertheless, there are certain AI techniques that are most commonly used for IC. These are Machine Learning (ML), Evolutionary Computing (EC), and Fuzzy Logic (FL). Other methods from AI do have applications in control systems, however these three methods are seen most frequently in the literature. Furthermore, there are significant synergies between these methods, which means they are often combined in control systems, as shown schematically in Figure 3.7.

Given below is a brief description of how ML, EC, and FL can be incorporated into IC systems. This is not an exhaustive list but demonstrates the most common architectures of intelligent controllers.

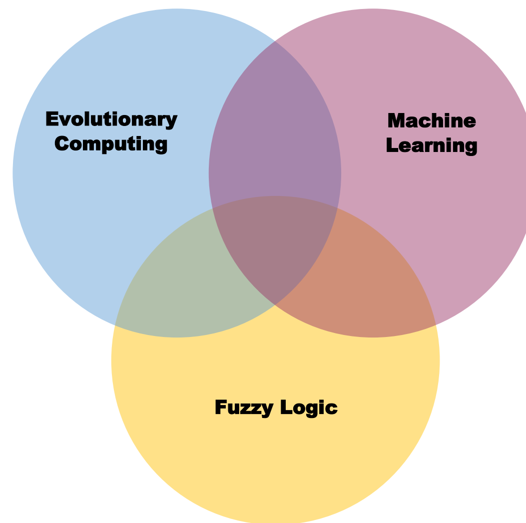


Figure 3.7: Intersections of AI techniques commonly used in IC systems.

Machine Learning

Since these techniques are by definition used to learn, they are well suited to being used in IC. Of all the techniques classed as ML, Neural Networks (NNs) are by far the most common, both within IC and in AI methods more broadly. These architectures can be used as nonlinear function approximators, which has several applications in control. For example, NNs can approximate uncertainties, model entire systems, or be incorporated directly as a controller [64].

Evolutionary Computing

This broad field of AI encompasses Evolutionary Algorithms (EAs), Genetic Algorithms (GAs), and Genetic Programming (GP). While these have subtle differences in their implementation, they are all based on biological evolution and develop solutions by "evolving" populations of potential solutions that are evaluated based on a fitness function. They are often very computationally expensive, which can be prohibitive for online learning, however there are still several applications where EC is used for online parameter adjustment in IC [65].

Fuzzy Logic

The name of this field reflects its main characteristic of dealing with partial truth to develop reasoning. One of the advantages of FL is its ability to incorporate expert knowledge when designing a controller, thus improving the interpretability of the derived control scheme. FL controllers are commonly used in a conventional manner where its control scheme does not update online, but can also be used for IC by adjusting parameters while operating [66].

Hybrid Methods

Due to the different strengths and weaknesses of each of the methods presented above, they are often combined to exploit the advantages of each method. For example, a NN can be used to approximate parts of a FL controller, or EC can update the structure of a NN. Any IC method that employs multiple AI techniques is here referred to as a "hybrid" method.

3.2.2 Dimensions of Intelligent Control

As discussed above, a defining characteristic of IC systems is their ability to deal with substantial uncertainties. Therefore, it is sensible to define the level of intelligence of a controller in terms of the level of uncertainty it experiences. In the task force characterisation of IC systems, there are three clear dimensions where uncertainty can be present: the environment, the controller laws and components, and the control goals. In more abstract terms, this is *what* is being controlled, *how* it is being controlled, and *why* it is being controlled.

Environment

Knowledge of an environment can be classed as the ability to express a model of the environment mathematically, where the environment includes both the system being directly controlled and any external effects. The design of a control system conventionally requires such a model and the level of uncertainty in this model affects the level of

intelligence necessary in the controller. Equation 3.1 shows the mathematical form of a generic system being controlled.

$$\dot{x} = f(x, u), \tag{3.1a}$$

$$y = h(x), \tag{3.1b}$$

where y is the system output, u is the system input, x denotes the system's state variables, and the functions f and h are mappings from their inputs to appropriately dimensioned vectors. The following equations only show expressions for $\dot{x}$ for simplicity. The environment model may also contain some parameters, $\zeta(t) \in \mathbb{R}^{n_\zeta}$, which vary with time. In this case the deterministic mapping from current state and control action to system output in Equation 3.1 no longer applies and now becomes time dependent. This case is shown in Equation 3.2.

$$\dot{x} = f(x, u, \zeta(t)) \tag{3.2}$$

The above models assume the function f is known to a precision that allows reasonable tracking accuracy between the model and real environment. This is not possible when the environment's dynamics are poorly understood. This is expressed in Equation 3.3 with the function $\hat{f}$ representing an uncertain mapping.

$$\dot{x} = \hat{f}(x, u, \zeta(t)) \tag{3.3}$$

Controller

Similarly to the environment, a controller can be mathematically modelled with varying levels of knowledge about its components. More intelligent controllers are more flexible and have less precise knowledge of their control laws at design time. A general feedback controller is described as

$$u = g(e), \tag{3.4}$$

where $e = y_d - y$ is the error between the desired system output, y_d and actual system output. This represents a controller with fixed parameters that are selected at design time. A general adaptive controller has control parameters $K \in \mathbb{R}^{n_K}$, which can vary depending on any inputs to the controller. Such a controller is described in Equation 3.5.

$$u = g\left(e, K(\cdot)\right) \tag{3.5}$$

The error e between desired and true system output can also be uncertain due to the behaviour of sensors and actuators. Even if the environment itself is well modelled, there may be errors, for example, in the measurements or unmodelled actuator dynamics. This case can be expressed as

$$u = \hat{g}\left(\hat{e}, K(\cdot)\right), \tag{3.6}$$

where $\hat{e} = y_d - \hat{y}$ is the measured error given the measured and uncertain state variables $\hat{y}$ and $\hat{g}$ denotes uncertain actuator response. More sophisticated controllers can vary more in their structure than just the control parameters. In this case, there may be multiple different control laws to select based on observations, or new control laws may be derived online. A general form of such a controller is

$$u = \hat{g}_i\left(\hat{e}, K_i(\cdot)\right), \quad i = 1, 2, \ldots, n_g, \tag{3.7}$$

where n_g is the number of different control schemes.

Goals

Compared to the other two dimensions, goals are more abstract in general and less rigorously mathematically defined. The level of knowledge of goals can then be thought of as how simply a goal can be expressed mathematically, as well as the level of awareness of goals in the controller. In most cases a controller's goal is defined as fulfilling some stability criterion or maintaining some performance measure across its operating range. In this case, the goal is entirely defined at design time and the controller has no awareness of this goal.

Another approach to defining control goals is to have some cost function that gives the controller an indication of its performance in a task. The controller then seeks to minimise this cost function with its control policy. In this case, the controller has some awareness of its goals and must find ways to achieve them instead of following prescribed routines to achieve a predetermined level of performance.

Beyond control systems with defined goals or cost functions, the goals become more abstract and defined in high level language rather than mathematically. In some cases, specific short-term goals may change over time as determined by the controller's internal planning. This is done with respect to some global goal, which remains constant. In cases where a global goal cannot be defined mathematically and the controller can only be given high level goals, this requires an intelligent system to deduce how to act appropriately and achieve such goals.

3.3 Taxonomy

IC methods are used where there is a substantial lack of knowledge at design time. This lack of knowledge comes under three main categories: the environment, the controller, and the goals. Within each of these categories, any controller, including conventional ones, can have a varying degree of uncertainty at design time. This section presents a classification scheme for control systems, which is based on the level of knowledge present in the control system at design time. Higher levels of uncertainty require a greater level of "intelligence" in the control system.

3.3.1 Environment Knowledge

0. *Complete and precise environment model:*

   If the environment is precisely known (where Equation 3.1 captures all dynamics), an open loop controller could be used, thus requiring no degree of intelligence. In reality there are often aspects of the system that are not perfectly modelled or subject to uncertainties. This then requires a more sophisticated controller.

1. *Complete environment model subject to minor variations:*

   Any real system can only be modelled to a certain degree of precision. This level of environment knowledge consists of only minor, bounded uncertainties. These uncertainties are small enough such that simple feedback controllers are sufficient to control the system without significant adaptation. These controllers are not necessarily intelligent, since they only require low levels of adaptation for dealing with slight uncertainties and do not learn online. There are still some examples of intelligent controllers within this category.

2. *Environment subject to change during operation:*

   At this level the environment has time-varying parameters that describe its behaviour (Equation 3.2). Now a higher degree of intelligence is required, since substantial changes in the environment cannot always be predicted or may be too complex to model. At this level of uncertainty, some conventional adaptive control methods can still perform sufficiently as well as intelligent ones.

3. *Underlying physics of environment not well defined:*

   Denoted here as an uncertain mapping from states and actions to future states as in Equation 3.3. This is an uncommon scenario for Earth applications where it is possible to reduce environmental uncertainty at design time. However, this can be an issue in applications such as space exploration. In this case, some information about the environment is known, but there are still substantial knowledge gaps requiring an intelligent controller.

4. *No knowledge of environment:*

   Where no model exists for the environment and the control designer cannot incorporate any environmental knowledge into the controller, this requires an intelligent control system to safely explore and create its own model of the environment online.

3.3.2 Controller Knowledge

0. *Stationary, globally stable controller:*

   Most feedback controllers have guarantees of stability and maintain a certain level of performance under given assumptions. In simple cases, these assumptions allow the control system to perform well with a fixed set of parameters without any need for adaptation (Equation 3.4).

1. *Varying controller parameters:*

   There are many examples of intelligent and non-intelligent applications that vary some control parameters online (Equation 3.5). This accounts for a lack of knowledge in the controller parameters, where fixed parameters at design time are insufficient to cover the entire operating range of the system.

2. *Unknown sensor/actuator behaviour:*

   This comes under the broad category of fault tolerant control, which itself has many dimensions. In this classification, fault tolerance represents a level of uncertainty in the controller, where measurements may be erroneous and actions may not create the predicted effect (Equation 3.6). Some fault tolerant systems use simple thresholds for indicating faults that are specified at design time, but since these are known at design time this does not fall under this category. Instead, this level of knowledge refers to a control system that must deal with unknown faults.

3. *Varying controller configurations:*

   At higher levels of intelligence, a controller can alter its own control structure online (Equation 3.7). This is commonly done offline using techniques such as EC to define the controller structure. An intelligent controller requires online adaptation and therefore an efficient means of adjusting its configuration while operating.

4. *No known controller structure:*

   The controller itself designs the control system from scratch using, for example, mathematical operations, control blocks, and intelligent architectures. An intel-

ligent controller must be able to do this online, but perhaps with a rudimentary initial controller to give a stable starting point.

3.3.3 Goal Knowledge

0. *Goals entirely predetermined by designer:*

   Most control systems, including intelligent systems, have a clearly defined goal that entirely shapes the control system design. In this case the control system is not "aware" of its goals and is therefore unable to update them or deliberately improve its performance with respect to the current goals. An example of such systems are those where the tracking error between a reference state and the current state must be reduced to zero.

1. *Goal specified implicitly, for example, as a reward function:*

   Many optimal control problems come under this category, since the aim of the controller is often to minimise or maximise a defined cost function when the means of optimising this function are not specified. The high level goal of the controller is then to derive a control policy that achieves optimal control with respect to this cost function. This is also the case where the controller is punished for detrimental actions and must find a control policy which avoids such actions. These examples fit well into the framework of Reinforcement Learning (RL), where an agent learns by interacting with the environment and observing its state and a reward.

2. *Specific goals subject to change during operation with a globally defined goal:*

   In a dynamic environment, the definition of specific goals may depend on contingent events and observations. Moreover, if the allocation of goals is performed remotely from the control system, such as in a space mission, the controller will have to wait for new instructions every time a new, unforeseen event occurs or a new set of observations is available. This requires an intelligent goal planner to elaborate new specific goals based on changes in the environment.

3. *One or several abstract goals with no clear cost function:*

   There are cases where the goals cannot be easily defined mathematically and so the controller requires an understanding of high level goals. At this level, goals are much broader in scope and defined in less machine-friendly formats, such as using natural language. For example, a controller's goal might be "safely navigate this environment and capture images of scientifically interesting events". The controller must be able to deduce how to navigate "safely" and what events are "scientifically interesting".

4. *No knowledge of goals:*

   The controller has to deduce what actions to take when, to begin with, it has no knowledge or indication of what actions are favourable.

3.4 Classification of Relevant Examples

This section gives some examples of intelligent controllers and their classification according to the taxonomy presented above. The following notation is used in the classifications below:

- E: Environment Knowledge
- C: Controller Knowledge
- G: Goal Knowledge

The methods presented here illustrate the applicability of the taxonomy to a range of IC techniques with varying levels of intelligence. The applications of these methods come from various engineering domains that require some level of autonomy, where the control system must achieve its goals without human intervention.

- *E-1, C-1, G-0:* Ichikawa and Sawa give an early example of NNs being used as direct controllers [67]. In their paper they combine a direct NN controller with genetic model reference adaptive control, which trains the NN based on a model

of the ideal system dynamics. This system is designed to deal with changing environment dynamics and continually updates its network to optimise performance.

- *E-1, C-2, G-0:* A common hybrid architecture for IC and particularly Fault Detection, Isolation, and Recovery (FDIR) is the Adaptive Neuro-Fuzzy Inference System (ANFIS), which was developed by Jang [68]. An example IC application of ANFIS is presented by Wang et al. [69]. Their system comprises an adaptive backstepping sliding mode controller augmented with an ANFIS FDIR system that controls a robotic airship. The ANFIS observer predicts the environment state at each time step. If these values disagree with those from the sensors, then a sensor fault is declared and the ANFIS output is used as input to the controller. The Goal classification is 0 since the goal of the control system is to minimise a tracking error following a predetermined trajectory.

- *E-1, C-3, G-0:* The NN controller proposed by Wu et al. [70] has a unique feature which makes its classification C-3. The controller can change the network topology and its parameters online based on the output of a learning algorithm. Such a change in the topology requires a trade-off between maintaining sufficient computational speed for online usage and the required precision in its output values.

- *E-2, C-2, G-0:* Another example of FDIR in IC systems is presented in [71]. Here a fault tolerant control scheme based on a backstepping controller integrated with a NN recognises unknown faults, with online adaptation of the NN weights. The overall system uses two networks to approximate unknown system faults and compensate for their effect. NN weights are updated online using a modified back-propagation algorithm.

- *E-3, C-1, G-0:* Such an uncertain environment as a Mars entry vehicle benefits from having an intelligent control system [72]. In this paper the authors develop a NN based sliding-mode variable structure controller. This controller has a fast loop, which is a conventional PID controller, and a slow loop, which contains the adaptive NN element. Their model of the environment also includes aerodynamic

terms that are difficult to model precisely. The goal is completely defined by the user through the definition of a nominal entry trajectory.

- *E-1, C-1, G-1:* Handelman et al. developed a hybrid IC architecture that comprises a Knowledge Based System (KBS) for devising learning strategies and a NN controller, which learns the desired actions and performs these consistently in real-time [73]. This is designed to mimic human learning by combining rule based initial learning and fine tuning by repetitive learning. The environment and controller considered here have low levels of uncertainty, and the control goals are only implicitly defined.

- *E-1, C-4, G-1:* An example of the use of Genetic Programming (GP) for IC is presented in [74], where it is used to derive a control law for a mobile robot moving in an environment with both known and unknown obstacles. The use of GP to create a control law gives this system a controller classification of C-4 since it derives the control law only using predefined mathematical functions without any prior knowledge of the controller structure. The environment has slight uncertainty from the unknown obstacles.

- *E-3, C-4, G-1:* An approach similar to [74], which also uses GP, is proposed by Marchetti et al. [75]. Here, GP generates a control law online and the controller is tested on different failure scenarios. In addition to these, they consider the case of an unknown environment model at design time.

- *E-1, C-1, G-2:* The Autonomous Sciencecraft Experiment onboard NASA's Earth Observing One is one of the most advanced satellite IC systems operated to date [76]. The control system has a hierarchical structure which is common in IC systems that incorporate planning. The highest level in the control hierarchy is the CASPER planner, which uses information from the onboard science to plan its activities. This is fed to the spacecraft command language, which then carries out the plan using lower level actions. This level can also adapt to environmental changes and make control adjustments as necessary. Below this level is conventional software, which simply carries out control actions as instructed by higher levels.

While this system does not operate in a substantially varying environment, it alters its controller parameters online and contains highly autonomous decision making and goal updating.

- *E-2, C-1, G-2:* WISDOM is a control system for rovers that is capable of high level planning and adaptive control [77]. This is another example of an IC system with a hierarchical structure. The top layer is responsible for generating plans, which are fed to the adaptive controller at a lower level. The adaptive system deals with immediate changes in the environment and gives instructions to the lowest level in the hierarchy, which is connected directly to the actuators. This system adapts to changing or uncertain environments and has varying parameters. The goals are also evolved over time in the system's planner.

The above examples demonstrate how the taxonomy can be applied to a range of fields and applications involving IC. From these examples, it can be seen that different AI techniques can perform similar tasks within an IC system, such as FDIR or planning. Although the AI approaches are different, the overall IC systems can be grouped under a certain classification, which highlights the consistency of the taxonomy. A few examples are listed below.

- *Robotic navigation and manipulation—E-2*: Robotic systems that operate in the presence of unknown obstacles or environment characteristics are classified as E-2. Under this category fall control systems that deal with uncertainties in the dynamic model, such as random disturbances [78], and robotic exploration in an uncertain environment, as with WISDOM [77].

- *Adaptive intelligent control—C-1*: Where the system is not subject to unknown faults, all control systems that adapt their parameters online are classified as C-1. This can be achieved with various AI techniques, but the common aspect within this category is some adaptation mechanism that updates the control law parameters. Examples in this category include adaptive NN control [67], adaptive fuzzy logic control of an electric motor with nonlinear friction [79], and neuro-fuzzy model-reference adaptive control of another electric drive system [80]. Although

each system employs different AI techniques, they are all classified as C-1 since they adjust control law parameters online.

- *Fault Detection, Isolation, and Recovery (FDIR)—C-2*: This encompasses control systems that can deal with failures in sensors or actuators in addition to an adaptation mechanism as in C-1. An often used technique in this category is ANFIS as in [81] for fault detection and diagnosis of an industrial steam turbine and in [69] where a controller is designed to reliably track the trajectory of a robotic airship in the presence of sensor faults. Other approaches typically use NNs, which can be incorporated for FDIR alongside a conventional control system, such as in [71] and [82].

- *Activity planning—G-2*: Systems that incorporate planning and reasoning are classified as G-2. These also use a variety of AI techniques to choose the desired states and how to achieve them. For example, in [76] the control system uses information from the onboard science subsystem to plan its activities. In [83] a reasoning strategy based on forward chaining is adopted to find optimal concentrations of chemicals for an electrolytic process.

Figure 3.8 shows the distribution of classifications of IC applications that were previously surveyed along with the year of publication of each application [27]. The line colour for each classification indicates the number of applications with the corresponding classification. It is clear from this figure that most surveyed applications have a modest level of intelligence with classifications of E-1, C-1, G-0. Uncertainties in the goal dimension are the least explored in these applications, whereas the control system dimension has some applications at the highest level of C-4.

3.5 Summary

It is clear that the use of Intelligent Control covers many classes of systems that require various levels of intelligence. Simpler environments can benefit from low levels of intelligence to tune their performance online, whereas more complex and uncertain

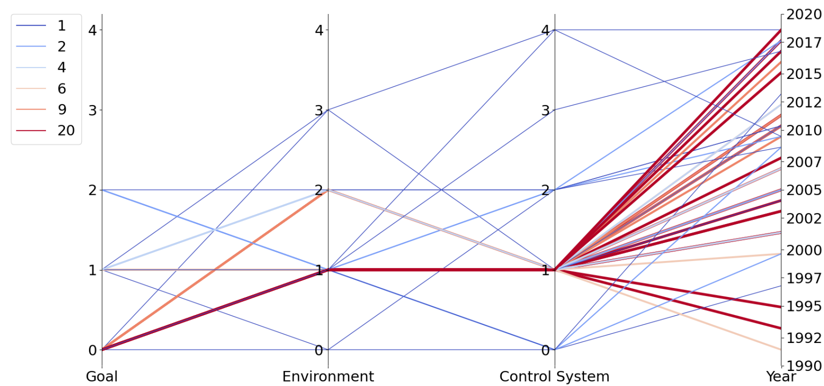


Figure 3.8: Parallel coordinates plot showing the number of applications of each IC classification previously surveyed.

environments require much higher levels of intelligence to operate effectively. This level of intelligence can be related to the extent to which the control system deals with uncertainties.

This chapter introduced a taxonomy of IC that classifies control systems based on their level of knowledge in three dimensions: environment, controller, and goals. The taxonomy can be applied to any control system to give a measure of its ability to handle uncertainties. From the applications studied, most of the IC systems focus on uncertainties in the environment and controller dimensions, which tend to have higher levels of intelligence than the goal dimension. This is to be expected since most control systems are designed with a very specific goal in mind, however there are still some examples of higher levels of intelligence with respect to goals.

Future applications of IC will likely continue to benefit from control systems with lower levels of intelligence. In addition, the development of more autonomous machines will require higher levels of intelligence. As progress continues towards more general purpose AI systems, this motivates more research in improving the level of goal intelligence in control systems, since the most general goals cannot be easily defined mathematically. An important compromise in online adaptive and learning IC systems is their high computational cost, which can limit their feasibility in resource-constrained systems such as space missions. Nevertheless, there are already examples of IC systems

running onboard spacecraft that handle uncertainties in all three dimensions. The following chapters will present another possible approach for creating IC systems that can learn online and run onboard a spacecraft.

Chapter 4

Q-Learning for Spacecraft Powered Descent

Chapter 3 described intelligent control systems as capable of handling large uncertainties in control problems. This chapter introduces an example control problem in aerospace that has several uncertainties: spacecraft powered descent. These uncertainties include atmospheric models, spacecraft aerodynamic coefficients, and initial conditions, among others [84,85]. Although the model of the spacecraft dynamics used here is a simplification of the real environment, it still incorporates some uncertainties and demonstrates the applicability of intelligent control. The nature of this problem lends itself well to the RL paradigm of learning through interaction and observing rewards as introduced in Chapter 2. When treating this problem as a RL problem, the agent is usually trained entirely offline. This would not be classified as "intelligent" in the same way online adaptive intelligent controllers are, but can still be useful for uncertain environments. For example, by incorporating uncertainties into the training process, a controller can learn how to handle uncertainties when deployed.

As with many machine learning problems, one bottleneck to the use of RL in practice is training times [86]. This is especially the case for more complex problems that require larger DNN architectures, such as in DQN learning to play Atari-2600 games from image inputs [39]. While these environments have high-dimensional state spaces, they benefit from a discrete action space. This allows efficient computation of action values via

Q-networks with one output per action. Many continuous action problems can also be simplified by appropriately discretising the action space, which allows the use of methods such as DQN to solve the problem.

Another limitation of certain RL algorithms is their sensitivity to hyperparameters [87]. Since some of the update rules used for training agents are inherently unstable, this means their hyperparameters require careful selection to avoid converging on poor solutions. This is commonly done using rules of thumb or trial and error [88]. Alternatively, optimising hyperparameters more methodically can improve the resulting performance. This is not a trivial task due to the high dimensionality of hyperparameter search spaces but is not intractable with current computational methods and capabilities.

This chapter shows how the spacecraft powered descent problem can be formulated in such a way that is is solvable using DQN. The main contributions presented here are:

- Application of the state-of-the-art RL method DQN to the spacecraft powered descent problem, which differs from other approaches of applying RL to this problem (see below).
- A detailed hyperparameter study to both find the best performing hyperparameters and show the effect of changing hyperparameters on the agent's performance in this problem. Furthermore, this study uses two different optimisation criteria representing different high level goals and shows how these affect the best hyperparameters and corresponding performance.
- Comparison of results for different state representations, which demonstrates the importance of properly defining the RL problem to avoid undesirable solutions.

The remainder of this chapter is organised as follows. Section 4.1 gives an overview of relevant literature in spacecraft powered descent. Section 4.2 describes the formulation of the environment as a RL problem. Finally, Section 4.3 presents results from applying DQN to the powered descent environment.

4.1 Relevant Literature

Autonomous GNC for extra-terrestrial landing is a well studied problem dating back to the Apollo lunar missions [89]. Following the success of these missions, there have been great advancements both in the computational capabilities onboard spacecraft and the methods for autonomous GNC. This is now an important area of research as further missions to the Moon and Mars are planned [85]. Existing approaches to powered descent guidance use a variety of methods spanning conventional control theories, optimal control, and ML [89].

The earliest landing GNC systems of the Apollo missions had to rely on conventional control methods. These missions had tight constraints on computing power which necessitated simple and robust control algorithms to generate feasible trajectories online [90]. Specifically, the onboard GNC used polynomial guidance algorithms that find analytical expressions for the spacecraft's acceleration in time that are used to control its thrusters. The Apollo guidance algorithms remained popular long after the Apollo missions, with some recent works aiming to expand on its capabilities [91, 92]. Another class of algorithms referred to as Zero Effort Miss (ZEM) and Zero Effort Velocity (ZEV) uses a similar set of assumptions to Apollo guidance and calculates analytical expressions for thrust commands [93]. Such methods have been applied to Mars powered descent [94,95] and other more general spacecraft control problems including asteroid intercept [96].

Optimal control approaches are also commonly applied to spacecraft powered descent to make best use of a spacecraft's highly constrained resources. These approaches mostly aim to minimise fuel consumption (or control effort), subject to constraints on the lander's final position and other aspects of the trajectory and actions [97]. There are various optimisation methods that can solve the minimum fuel powered descent problem offline for specific initial conditions. Some of the earlier research into this problem derived certain characteristics of the optimal descent trajectories, including the "bang-bang" thrust profile, and used Non-Linear Programming (NLP) approaches to find optimal solutions [98]. When designing control systems that are to be optimised online, there are additional challenges. As with intelligent control methods, computational resources

onboard spacecraft are a significant limitation. In addition, some optimisation methods, such as NLP approaches, are not guaranteed to converge on a solution, which is necessary for mission critical operations [89]. For these reasons, another class of approaches for optimal powered descent guidance uses convex programming methods [99]. These approaches solve the optimal control problem online by convexifying any nonconvex constraints, such as thrust and glideslope requirements [100–103]. Convex programming approaches are suitably efficient and lightweight to be implemented on spacecraft hardware—as demonstrated by terrestrial flight-tests using a vertical-takeoff/vertical-landing rocket [104, 105]. A hybrid approach to find optimal solutions for powered descent has also been proposed to combine the convergence guarantees of convex programming with the accuracy of solutions of NLP approaches [106]. To account for more general nonconvex problems, such as the nonlinear relationships between aerodynamic disturbances and the spacecraft state, methods of successive convexification are also used. In this case, the optimisation algorithm iteratively generates solutions to convex approximations of the nonconvex problem using linearization [107–109]. Similarly, another approach based on Model Predictive Control (MPC) uses piecewise affine approximations of the lander's dynamics to find optimal solutions in an efficient manner without requiring nonlinear MPC [110].

One of the main issues with optimal control problems is they assume a deterministic environment and are often not robust to substantial uncertainties inherent in powered descent. Other work has explicitly included uncertainties in the optimal control problem using a method called covariance steering [111]. In this approach, a feedback controller is designed to control the state covariance, i.e. the uncertainty in the state. This method has also been used to adapt convex programming approaches to handle uncertainties [112]. Although such methods effectively deal with uncertainties along the spacecraft's trajectory, they still require certain assumptions on the system, such as on the initial magnitude of the state covariance and the mass profile over the trajectory.

Due to its performance in other domains, RL has also gained a lot of interest as an approach to many spacecraft GNC problems. The most commonly used method for these application is Proximal Policy Optimisation (PPO), which performs well in

high-dimensional tasks with continuous actions but requires longer training times than similar discrete-action methods. RL has been used to solve a variety of trajectory optimisation problems, including interplanetary trajectories and orbit transfers. These aim to provide greater robustness to uncertainties compared to non-ML methods, such as in [113] where the authors include noise in various aspects of a low-thrust Earth-Mars trajectory problem. Other works also investigate using RL for low-thrust manoeuvres applied to transfers in the Earth-Moon system [114–117]. As well as training directly on the problem without prior knowledge, RL can also be used effectively in combination with conventional control methods, such as by feeding velocity commands from a RL agent to lower-level controllers for proximity operations [118], deriving optimal "Q-laws" for spacecraft transfers with PPO [119, 120], and relative motion guidance via RL and ZEM/ZEV [121].

Recent work has explored the possibility of incorporating Large Language Model (LLM)-based agents into IC systems for a spacecraft. One of the earliest examples of this approach fine-tunes a language model to for use as a controller in a variety of spacecraft control problems including powered descent [122]. In this example the LLM directly outputs control actions, such as 3D thrust vectors, based on textual input. Similarly, other work has applied LLMs to the problem of manoeuvring satellites in non-cooperative scenarios [123]. Again this system uses textual inputs and outputs where the inputs consist of real-time telemetry and outputs are commanded actions. Compared to most of the other AI approaches described here, LLMs are some of the most computationally heavy AI systems. Although they can be used in offline training, this makes them less suitable for online updates on spacecraft hardware. However, this area of research is still in its infancy and simultaneous advances in hardware and the efficiency of language models could make such approaches suitable for online IC systems.

Previous works have also investigated the application of RL to spacecraft powered descent. The Farama Foundation's Gymnasium[1] software is a set of benchmark environments for testing RL agents that has a highly simplified 2D lunar lander environment [125]. This environment only starts from a small range of initial conditions and

[1]Formerly OpenAI Gym [124].

incorporates limited uncertainties. When considering a larger problem space, training agents from scratch becomes more challenging as they might never reach the desired final state. This makes careful tuning of the reward function necessary as demonstrated in [23] where the authors train an agent on a 6-Degrees-of-Freedom (DOF) Mars lander problem using PPO. Other approaches to training agents for powered descent also combine RL with previously described analytical methods such as ZEM/ZEV guidance. In this framework, RL has been used to adapt parameter values [24] and also to guide the spacecraft between determined waypoints [21]. Another work combines pseudospectral optimisation methods with RL for determining the "handover" point where a spacecraft switches from entry-phase to powered descent-phase [22]. In this case the action space of the RL agent is very limited since it is only used for determining a switching point. LLMs have also been incorporated into training via RL for spacecraft powered descent [126]. This work extracts information from an example lunar landing "manual" and uses this to inform the design of the lander's reward function. The reward function is further refined during training based on information from the training process.

Most applications listed above use PPO, which is an effective method for solving RL problems in continuous action spaces. However, since it is on-policy it requires optimisation over trajectories which is not as data efficient as many alternative off-policy approaches. This work aimed to show that DQN is capable of solving spacecraft powered descent in a discrete action space.

4.2 Powered Descent Problem

The goal of spacecraft powered descent is to achieve a pinpoint soft landing, as defined by limits on the terminal state, while minimising fuel consumption or control effort. This section describes the main elements of powered descent as a RL problem. First, the environment dynamics are described, which define the state transition probabilities in a classical MDP. Then the reward function is defined with a sparse component to reward successful episodes and a shaped component to guide the agent. The final parts of the problem are the state space and action space, which represent what the agent observes of the environment along with the reward and how the agent affects the

environment. This formulation of the problem comes from previous work by Gaudet et al [23].

4.2.1 Environment Dynamics

The lander's position is denoted $\mathbf{r} = \begin{bmatrix} r_x & r_y & r_z \end{bmatrix}$ and similarly the velocity is $\mathbf{v} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$. The forces acting on the lander are gravitational force, engine thrust $\mathbf{F}_T = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}$ and disturbance forces $\mathbf{F}_D = \begin{bmatrix} F_x & F_y & F_z \end{bmatrix}$. This disturbance force represents lumped uncertainties in the system due to wind, changes in atmospheric density, and uncertainties in thrust magnitude. Initially, this disturbance force is excluded from the simulations, i.e. $\mathbf{F}_D = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ at each timestep, and these will be included in results presented in later chapters.

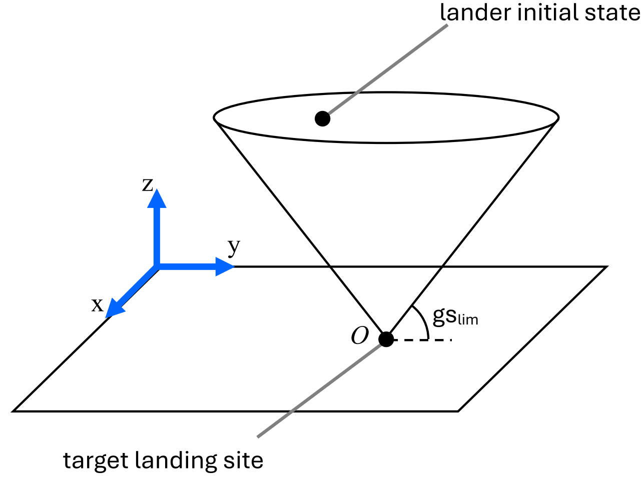


Figure 4.1: Illustration of the lander powered descent frame of reference and glideslope limit.

Figure 4.1 shows the surface fixed frame of reference used to define the lander's equations of motion. Positions $\mathbf{r}$ are defined relative to the origin at the target landing location on the surface. Equations 4.1 to 4.3 give the 3-DOF equations of motion of a lander of mass m subject to gravitational acceleration $\mathbf{g}$ in the surface fixed frame of reference. This form of the equations assumes a uniform gravitational field strength

over the duration of the powered descent, which is $\mathbf{g} = \begin{bmatrix} 0 & 0 & -3.7114 \end{bmatrix} N/kg$ for Mars.

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{4.1}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}_T + \mathbf{F}_D}{m} + \mathbf{g}, \tag{4.2}$$

$$\dot{m} = -\alpha_T \|\mathbf{F}_T\|, \tag{4.3}$$

where $\alpha_T = \frac{1}{I_{sp} g_{ref}}$ defines the fuel consumption rate, I_{sp} is the specific impulse of the engine and $g_{ref} = 9.81 N/kg$ is the reference gravitational field strength. The value of $I_{sp} = 210s$ is the same as used in previous works to represent typical performance of a lander's thruster [23]. In this 3-DOF formulation, only the lander's translational dynamics are considered without the coupled attitude dynamics. The sampling time used for transmitting data to the agent is $0.2s$.

One of the important considerations for powered descent guidance and control is the range of initial conditions for the lander's position, velocity, and mass. This uncertainty is incorporated into the training process by sampling uniformly over a wide range of initial conditions in training episodes. The range of possible initial values for each uncertain variable are shown in Table 4.1, which are also the same as used in previous works [23]. These values give a reasonable range of initial conditions for the powered descent phase for which the controller should be expected to perform.

Table 4.1: Range of initial conditions in the Mars lander powered descent environment.

Parameter	Min. Value	Max. Value
Downrange position, r_x (m)	0	2000
Crossrange position, r_y (m)	-1000	1000
Elevation position, r_z (m)	2300	2400
Downrange velocity, v_x (m/s)	-70	-10
Crossrange velocity, v_y (m/s)	-30	30
Elevation velocity, v_z (m/s)	-90	-70
Mass, m (kg)	1900	2100

4.2.2 Shaped Reward Function

Powered descent can be readily formulated as a sparse reward problem where all state transitions except to a terminal state yield zero reward. The terminal reward indicates either success or failure of the task, for example as a positive or negative reward respectively. These problems are particularly challenging for RL agents since they must first explore the environment to "find" the correct terminal state which can take many trials. The additional goal of minimising control effort or fuel consumption can be incorporated as a reward at each step that is inversely proportional to the level of thrust at that timestep. However, this would discourage the agent from exploring the environment and therefore decrease the likelihood of achieving a desired terminal state.

One way of dealing with this problem as presented in [23] is to use a shaped reward function that guides the learning towards the goal. Although this can cause a loss in optimality with respect to minimal fuel consumption, this vastly speeds up the learning by avoiding excessively long periods of exploration. The reward function for this environment combines a sparse reward for a successful landing and a shaped reward that penalises control effort while motivating the agent to move towards the desired landing site. Numerically, the reward is defined as

$$
\begin{aligned}
r = {} & \alpha \left\| \mathbf{v} - \mathbf{v}_{targ} \right\| + \beta \frac{\|\mathbf{F}_T\|}{\mathbf{F}_T^{\max}} + \eta \\
& + \kappa \left((r_z < 0) \wedge (\|\mathbf{r}\| < r_{lim}) \wedge (\|\mathbf{v}\| < v_{lim}) \wedge (gs \geq gs_{lim}) \right), \qquad (4.4)
\end{aligned}
$$

where $\mathbf{v}_{targ}$ is a target velocity defined below, $\mathbf{F}_T^{\max}$ is the maximum thrust magnitude, and the coefficients α, β, η, and κ weight different parts of the reward. The α term is a negative reward that penalises deviation from a defined target velocity. The β term is also a negative reward for the control effort normalised with respect to the maximum thrust. η is a positive constant that motivates the agent to keep advancing in the environment. Finally, κ is the positive reward gained for a successful landing within defined limits on the terminal state. r_{lim} and v_{lim} are limits on the magnitude of the terminal position and velocity respectively. gs_{lim} defines the minimum allowable glideslope angle. The glideslope gs is then the maximum glideslope angle over the

trajectory:

$$gs = \max_t \left(\arctan \frac{r_z}{\sqrt{r_x^2 + r_y^2}} \right), \ t = 0, 1, ..., T-1 \tag{4.5}$$

The experiments shown in this chapter do not consider this constraint on the trajectory, i.e. $gs_{lim} = 0$, but this constraint will be included in later results. Figure 4.1 also illustrates this glideslope limit, which effectively defines a cone of allowable states with vertex at the target landing site.

The target velocity is defined as shown:

$$\mathbf{v}_{targ} = -v_0 \left(\frac{\hat{\mathbf{r}}}{\|\hat{\mathbf{r}}\|} \right) \left(1 - exp\left(-\frac{t_{go}}{\tau} \right) \right) \tag{4.6}$$

$$v_0 = \|\mathbf{v}_0\| \tag{4.7}$$

$$t_{go} = \frac{\|\hat{\mathbf{r}}\|}{\|\hat{\mathbf{v}}\|} \tag{4.8}$$

$$\hat{\mathbf{r}} = \begin{cases} \mathbf{r} - [0 \ 0 \ 15], & \text{if } r_z > 15 \\ [0 \ 0 \ r_z], & \text{otherwise} \end{cases} \tag{4.9}$$

$$\hat{\mathbf{v}} = \begin{cases} \mathbf{v} - [0 \ 0 \ \ -2], & \text{if } r_z > 15 \\ \mathbf{v} - [0 \ 0 \ \ -1], & \text{otherwise} \end{cases} \tag{4.10}$$

$$\tau = \begin{cases} \tau_1, & \text{if } r_z > 15 \\ \tau_2, & \text{otherwise} \end{cases} \tag{4.11}$$

This motivates the agent to follow a velocity pointing towards $15m$ above the desired landing location which decreases exponentially as it approaches this point. Then over the final $15m$ of descent the target velocity is entirely normal to the surface in the z-direction. The values of constant terms in the reward function are shown in Table 4.2. These were originally defined in [23] with the aim of balancing each component to achieve desirable learning performance.

Table 4.2: Values of terms defining the powered descent reward function.

Parameter	Value
α	-0.01
β	-0.05
η	0.01
κ	10
r_{lim}	$5m$
v_{lim}	$2m/s$
τ_1	$20s$
τ_2	$100s$

4.2.3 State Space

Like the reward function, careful selection of the state representation can assist the agent in learning an effective policy. The results presented here compare two different state representations referred to as a "shaped" state and a "raw" state. The shaped state is closely related to the shaped reward function and is shown in Equation 4.12. This state includes terms for the error between the spacecraft's velocity and the target velocity in each direction. It also includes the parameter t_{go} from Equation 4.8 which gives a crude estimate of the remaining time before landing based on the position and target velocity. The final term in the shaped state is the altitude, r_z, which is the most useful distance measure for deciding actions.

$$s_{shape} = \begin{bmatrix} v_x - v_x^{targ} & v_y - v_y^{targ} & v_z - v_z^{targ} & t_{go} & r_z \end{bmatrix} \tag{4.12}$$

Unlike the shaped state representation, the raw state does not include information on the target velocity. Equation 4.13 shows the terms used for the raw state. These are simply the position, velocity, and mass of the spacecraft at the current timestep. The values in the raw state are also scaled by a constant factor s_{scale} to avoid having excessive values input to the agent.

$$s_{raw} = \begin{bmatrix} r_x & r_y & r_z & v_x & v_y & v_z & m \end{bmatrix} \cdot s_{scale}, \tag{4.13}$$

with the values for scaling each term defined as

$$s_{scale} = \begin{bmatrix} 0.01 & 0.01 & 0.01 & 0.1 & 0.1 & 0.1 & 0.005 \end{bmatrix}. \tag{4.14}$$

4.2.4 Action Space

The actions taken by the agent for powered descent are commanded thrusts: $a_t = \mathbf{F}_T$ at timestep t. The implementation of DQN used here requires a discrete action space. For a typical powered descent problem, commanded thrusts are defined on a continuous space of $\mathbb{R}^3$ with limits on the maximum thrust in each direction. In this discrete action formulation, there are a number of potential commanded thrust magnitudes in each direction. For the z-direction, since the acceleration due to gravity acts in the negative direction, the commanded thrusts in this axis are all in the positive z-direction.

The simplest way to discretise the action space is to allow a maximum, minimum, or zero commanded thrust in each direction. Although there are no negative thrusts in the z-direction, a midpoint thrust magnitude allows better control of vertical speed. In the case where three discrete values are defined in each direction, the action space is

$$a_t = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}, \tag{4.15}$$

$$T_x \in \{-T_x^{\text{max}},\ 0,\ T_x^{\text{max}}\}, \tag{4.16}$$

$$T_y \in \{-T_y^{\text{max}},\ 0,\ T_y^{\text{max}}\}, \tag{4.17}$$

$$T_z \in \{0,\ T_z^{\text{mid}},\ T_z^{\text{max}}\}, \tag{4.18}$$

where T_i is the commanded thrust in the i-direction, T_i^{max} is the maximum possible thrust in the respective direction, and T_z^{mid} is the midpoint thrust magnitude in the z-direction. 3 dimensions with 3 possible actions gives an action space size of $3^3 = 27$, assuming each commanded thrust is controlled independently of the others. The default values for maximum thrust were initially specified as $T_x^{\text{max}} = T_y^{\text{max}} = 5kN$ and $T_z^{\text{max}} = 12kN$ based on values used in previous works to represent typical performance of a lander's thruster [23]. In addition, the midpoint thrust magnitude was initially set as $T_z^{\text{mid}} = 0.5T^z_{max}$. As discussed in the following section, these values were further tuned to improve the agent's performance.

4.3 Results of DQN Applied to Powered Descent

This section demonstrates the application of the DQN algorithm described in Chapter 2 to the 3-DOF lander powered descent problem. These results are presented in four main parts. First is the procedure for optimising hyperparameters with two different loss functions. Second is action magnitude selection, i.e. selecting an appropriate value for $T_i^{\max}$ and observing the effect its value has on certain performance measures. Third, results of training DQN with a shaped state representation are compared to those obtained using PPO in a continuous action space. Finally, results are shown from applying the method with a raw state representation. Unless otherwise stated, for all the results presented in this thesis, simulations ran on an Ubuntu 18.04 computer with a 3.6GHz Intel i7-4790 CPU and 8 GB RAM.

4.3.1 Hyperparameter Study

One of the main difficulties in optimising hyperparameters is the search space, which can contain mixtures of real, integer, and categorical values. In addition, the number of hyperparameters can be large resulting in a high-dimensional search space that is difficult to optimise. The simplest approach to the problem of automatically selecting hyperparameters is to perform a random search and select the best configuration, which can be surprisingly effective [127]. Improvements to the random search include Hyperband, which uses a bandit based approach to speed up evaluation [128]. This method assumes that an algorithm's performance in early training epochs can be used to indicate later training performance, which does not always hold well when training RL agents. Other automated hyperparameter search methods incorporate Bayesian optimisation methods, such as Spearmint [129], Tree of Parsen Estimators (TPE) [88], and SMAC [130]. The optimisation presented here uses TPE since this method has been applied successfully in computer vision tasks which possess the problems previously described. The implementation of TPE is from the Python library "hyperopt", which is designed for application to machine learning models [131].

Each trained Q-network has 3 hidden layers, which is sufficiently deep to learn useful representations for DQN. The optimisation controls the number of hidden nodes in each layer, denoted $\tilde{N}_l$, $l = 1, 2, 3$. The other hyperparameters related to the Q-network are the learning rate, α and target network update steps, C. There are 4 additional hyperparameters for DQN: initial exploration probability, ϵ_0; number of episodes to decrease ϵ, N_ϵ; discount factor, γ; and minibatch size, k. The final exploration probability ϵ_f is fixed as 0. In all cases the network activation function is *tanh* and the weight updates use RMSprop optimisation as implemented in the TensorFlow Python library [132].

To select an appropriate number of training episodes N_{ep}, an initial training run of 10,000 episodes was performed. This used an untuned, but stable, hyperparameter configuration as follows: $\tilde{N}_1 = 100$, $\tilde{N}_2 = 150$, $\tilde{N}_3 = 100$, $\alpha = 2e-5$, $C = 65$, $\epsilon_0 = 0.5$, $N_\epsilon = 2000$, $\gamma = 0.95$, $k = 100$. Figure 4.2 shows the learning curve with these settings. Over the first 1000 episodes the reward oscillates before rapidly increasing. After around 1500 episodes the rate of increase is more gradual and remains so for the rest of the episodes. To strike a balance between maximising performance and minimising training duration, $N_{ep} = 4000$ episodes was chosen as the duration for training agents.

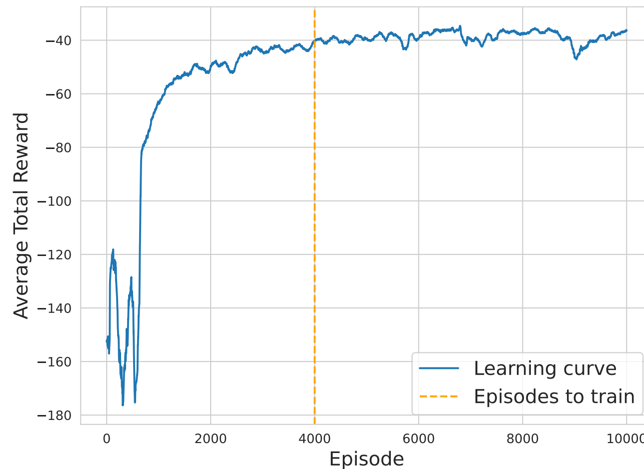


Figure 4.2: Initial agent learning curve to establish the number of training episodes.

In the context of RL problems there are several ways to define optimisation criteria for hyperparameter selection. Since the agent's main goal is to maximise its total reward, one possible criterion is the cumulative reward at the end of training. An additional

goal explored here is to minimise the time required to learn a near-optimal policy. This can be achieved by maximising the area under the learning curve. These criteria are referred to as reward-optimisation and area-optimisation respectively. Letting R_i denote the cumulative reward received in episode i, the area and reward loss functions are defined as

$$L_{area} = -\frac{1}{N_{ep}-1}\sum_{i=1}^{N_{ep}-1}\frac{(R_i + R_{i+1})}{2}, \tag{4.19}$$

$$L_{reward} = -\frac{1}{500}\sum_{i=N_{ep}-499}^{N_{ep}} R_i, \tag{4.20}$$

where the optimisation seeks to minimise the loss. The area loss is the negative area under the learning curve, which is calculated via trapezoidal integration. The reward loss takes the average reward over the last 500 episodes of training. Due to the stochastic nature of the environment and agent, each evaluation takes the average loss over 8 runs with different random seeds. The combined loss for a single evaluation is then the upper 95% bootstrapped confidence interval in the average across runs. This is to give an indication of worst-case average performance such that evaluations with high variance in loss have a higher loss than those with lower variance.

Table 4.3 shows the selected hyperparameters based on the reward-optimised and area-optimised losses. Figures 4.3 and 4.4 show the results of the hyperparameter optimisation as parallel plots. Each line in the plot represents one evaluation in the optimisation and the magnitude on each axis shows the value for its respective hyperparameter. Darker lines indicate a lower value of the respective loss. Note that the ranges of each parameter are different between the two loss functions. This is due to adjustments made on the limits while running evaluations to increase the range of certain parameters being trialled.

For both of the loss functions, the optimised Q-network structure has fewer hidden nodes in the middle layer than the outer two layers. The parallel plots highlight this tendency towards such a structure, with many darker lines in a v-shape for both optimisations. The area-optimised network is also larger than reward-optimised, with every layer containing more hidden nodes than the corresponding reward-optimised layer.

Table 4.3: Selected DQN hyperparameters when optimising for area under learning curve and final reward.

Parameter	Area-optimised	Reward-optimised
$\tilde{N}_1$	212	150
$\tilde{N}_2$	80	65
$\tilde{N}_3$	218	165
α	3.96×10^{-5}	2.08×10^{-5}
C	70	65
ϵ_0	0.367	0.269
N_ϵ	300	2700
γ	0.914	0.926
k	58	104

It is expected that the learning rate would be higher when optimising for area under learning curve such that the algorithm converges more quickly on a solution. This is the case for the optimised values for α, which is nearly 2 times larger in the area-optimised case. Another significant difference in the optimised hyperparameters is for those affecting exploration. The value of N_ϵ for area-optimised is 9 times smaller than that of reward optimised, which is also expected since the area-optimised agent aims to exploit greedy actions as quickly as possible. While the initial exploration probability ϵ_0 is larger in the area-optimised case, its exploration probability will quickly decrease to less than the reward-optimised agent. From the parallel plots, it is clear that the best evaluations for the reward loss had a wide range of values for ϵ_0, however the best for the area loss are clustered around lower values. This is also the case for N_ϵ, where the best values for the reward optimisation tend to occupy a higher range of values. The final notable difference is in the minibatch sizes, which are 58 and 104 for area-optimised and reward-optimised respectively. The reason for this is unclear but, as can be seen from the parallel plots, this trend was shown across all evaluations with the reward loss favouring larger minibatch sizes and area loss having lower values near $k = 58$. Results in the remainder of this section use both the area-optimised and reward-optimised hyperparameters to compare their performance.

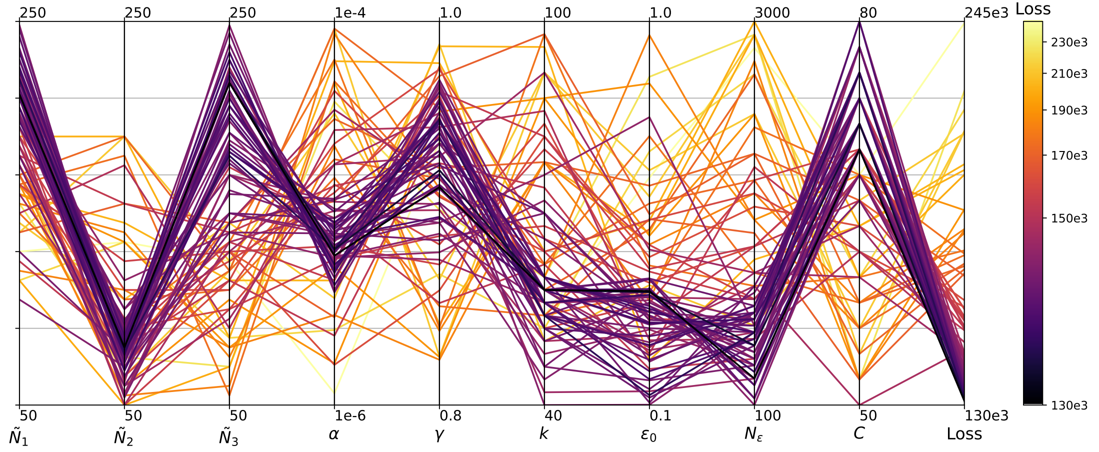


Figure 4.3: Parallel plots showing hyperparameter optimisation evaluations for area. Colour indicates loss normalised via a power norm for clarity.

4.3.2 Action Size Selection

In addition to the agent's hyperparameters, the magnitude of commanded thrusts can be tuned to improve the agent's performance. Altering the value of $T_z^{\max}$ did not noticeably affect performance and so this value was fixed as $T_z^{\max} = 12kN$. Given the rotational symmetry of the environment about the z-axis, the selection of action magnitudes can be simplified by setting $T_x^{max} = T_y^{max}$, such that only a single value needs to be determined for both directions.

To examine the effect of changing $T_i^{\max}$ in the x- and y-direction, agents were trained for 4000 episodes with each magnitude and tested for 500 episodes without further updates. Values of action magnitude were varied in the range $[5kN, 12kN]$ in steps of $1kN$. The performance metrics examined in the testing episodes were the fuel consumption and the cumulative reward as shown in Figure 4.5, where it is desired to minimise fuel consumption and maximise cumulative reward. Values shown are the average across testing episodes along with the standard error, since it is also desired to minimise variance in performance between episodes. One important observation from these figures is that the trend in fuel consumption does not match that of the total reward, despite the fact that control effort is part of the reward function and related to fuel consumption. This is due to the landing bonus and velocity tracking terms in the reward function

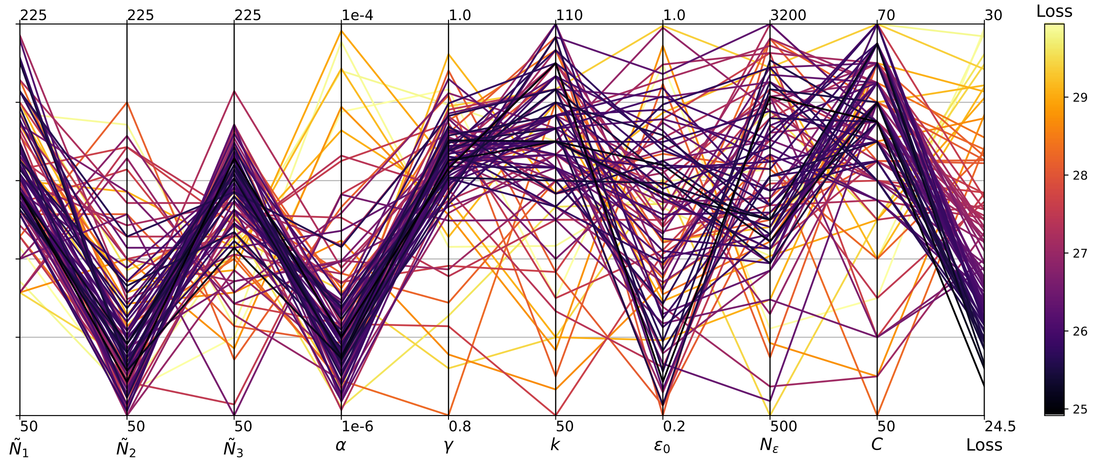
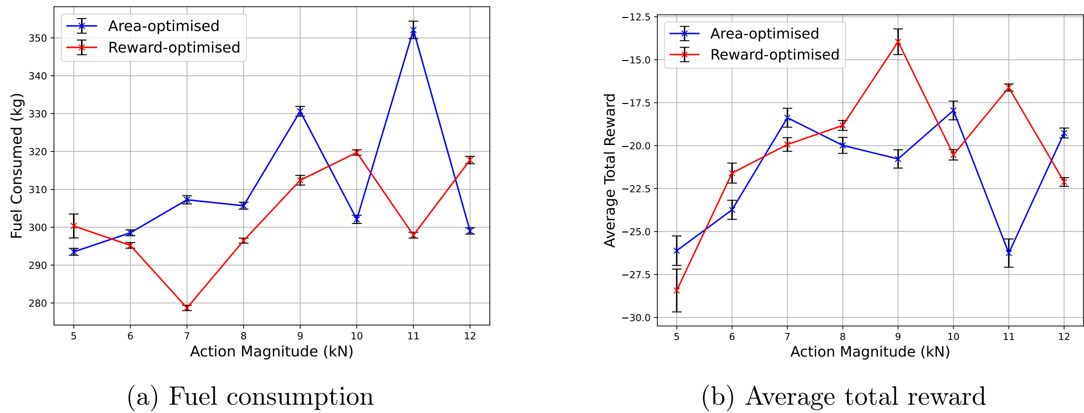


Figure 4.4: Parallel plots showing hyperparameter optimisation for reward. Colour indicates loss.

which can be more fuel intensive to achieve but produce larger rewards. Both fuel consumption and average total reward vary considerably across action magnitudes and occasionally have large peaks and troughs, for example when the magnitude is $11kN$ for the area-optimised agent.



(a) Fuel consumption

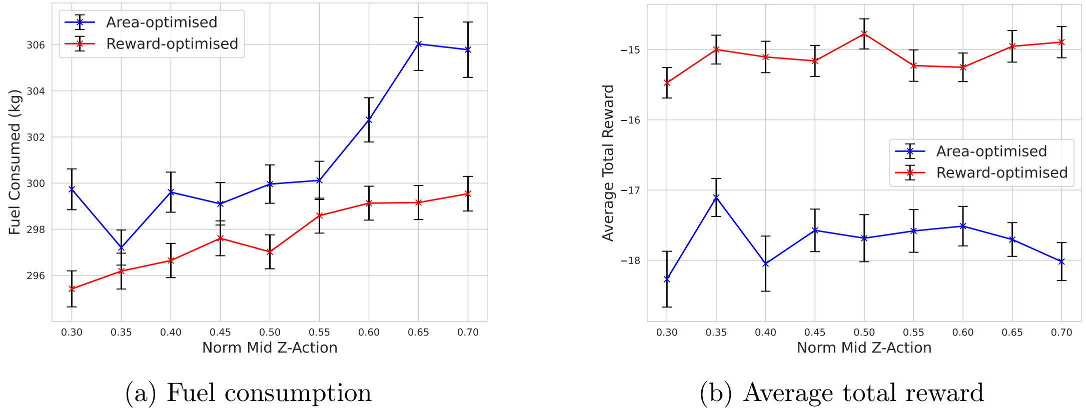
(b) Average total reward

Figure 4.5: Change in performance with varying values of action magnitude for the powered descent problem. Error-bars denote one standard error.

Considering first the reward-optimised agent, its best performance in terms of reward is at $9kN$, but this also has the third highest fuel consumption across its range. Furthermore, the reward shows higher variance at $9kN$ relative to other values. The action magnitude for the reward optimised agent was therefore set as $T_i^{\max} = 11kN$

which gives the second highest reward and median fuel consumption. This differs significantly for the area-optimised agent for which $11kN$ is notably the worst performing value. Peak reward for this agent is at $10kN$ which does also show favourable fuel consumption. However, as before the point that has lower variance in reward is more beneficial and so the action magnitude for the area-optimised agent was selected as $12kN$.

As stated previously, the midpoint thrust in the z-direction was initially fixed at half the maximum thrust. Further tests were carried out to examine the effect of varying this magnitude on performance. Using the action magnitudes for the x- and y-directions specified above, the value of T_z^{mid} was varied between $0.3T_z^{\text{max}}$ and $0.7T_z^{\text{max}}$. For this analysis a single agent for each set of hyperparameters was trained with $T_z^{\text{mid}} = 0.5T_z^{\text{max}}$ and then tested for 500 episodes. The results are shown in Figure 4.6 with the same performance metrics as above of fuel consumption and cumulative reward.



(a) Fuel consumption

(b) Average total reward

Figure 4.6: Change in performance with varying mid-points of z-direction action for the powered descent problem. Error-bars denote one standard error.

Both metrics occupy a much narrower range of values in these tests compared to those in Figure 4.5. In particular, the cumulative reward for each set of optimised hyperparameters remains nearly constant across the values of the midpoint thrust. As expected, the fuel consumption tends to increase with the midpoint thrust, especially for the area-optimised agents. However, values below 0.5 show very little improvement in the fuel consumption for both agents. Since all of these agents were trained using the same midpoint action of $0.5T_z^{\text{max}}$, testing performance could still be improved for other

values by also training the agent using these action magnitudes. Nevertheless, the results shown here suggest tuning this action has little effect on the resulting performance and so this midpoint action was kept fixed at half the maximum magnitude.

4.3.3 Training and Testing Agents

Using the two sets of hyperparameters and action magnitudes defined above, multiple agents were trained in the powered descent environment. 50 training runs of 4000 episodes were carried out for each set of hyperparameters to assess their learning performance across different random seeds. The mean cumulative rewards across runs for both hyperparameters are shown in Figure 4.7.

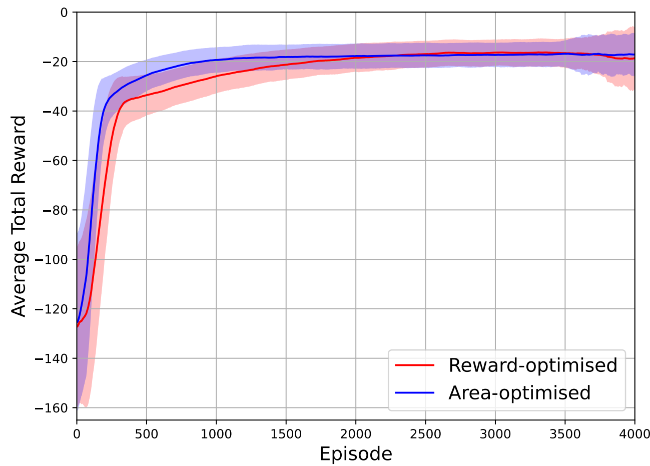


Figure 4.7: Learning curves averaged over 50 runs for each agent. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 120 episodes for clarity.

The area-optimised and reward-optimised learning curves are both similar with a sharp increase in average reward over the first 300 episodes before gradually plateauing at their maximum average reward. As would be expected, the area-optimised curves reach this maximum more quickly than the reward-optimised. Considering the standard deviation, there is little variance in performance over most of the training period. In both cases the highest variance occurs in the initial episodes when more random actions are taken. There is also a slight increase in variance over the final 500 episodes for both hyperparameters, but most notably in the reward-optimised curves. This is

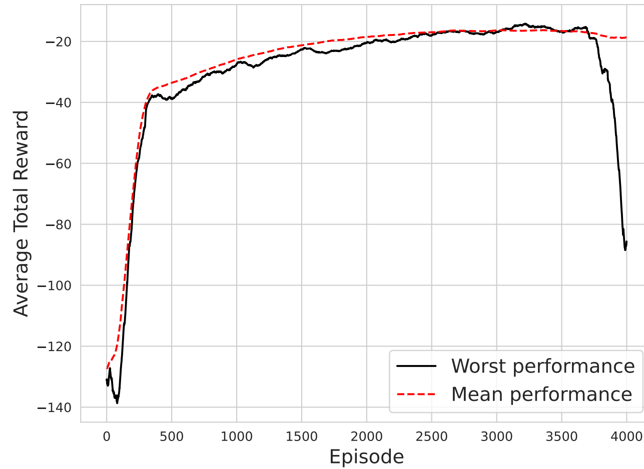


Figure 4.8: Mean and worst performing learning curves for the reward-optimised agent.

highlighted in Figure 4.8 which shows the learning curve for a reward-optimised agent that performs poorly towards the end of training, thus causing the standard deviation of reward-optimised runs to increase. Although the optimisation metric for hyperparameters aimed to encourage robustness in the solution, achieving a more robust solution that is less susceptible to diverging performance would require more training runs per evaluation. However, most of the trained agents showed favourable learning performance and can be successfully applied to this problem.

The following results use the optimal trained agents from the area-optimised and reward-optimised runs with respect to their respective loss functions (Equations 4.19 and 4.20). The individual learning curves from each agent and the number of steps per episode are shown in Figure 4.10. For comparison, a learning curve obtained from training an agent using PPO, as demonstrated in previous work [23], is shown in Figure 4.9. Note that vertical axis scales are identical in both Figures, however the horizontal scale shows the difference in the number of training episodes between DQN and PPO, which are an order of magnitude less when training with DQN. The policy trained using PPO does converge on a higher average reward than either of the DQN agents, which could be a result of the discretised action space causing a loss in optimality of the learned policy. This suggests a necessary trade-off between performance and learning time when choosing a RL algorithm to train an agent. While the average total reward is

different at the end of training, the number of steps per episode of every trained agent is consistently close to 300. This is likely a result of the shaped reward function, which specifies the target velocity for all states and therefore will tend to cause similar episode lengths.

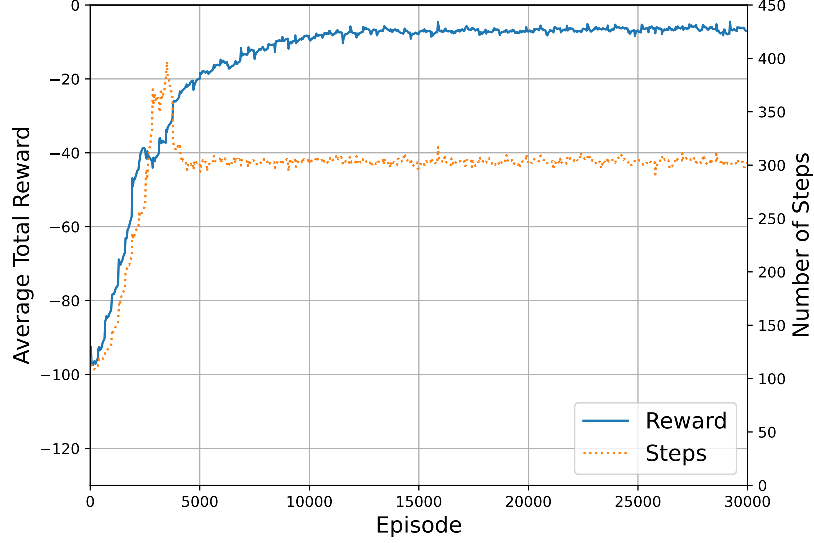


Figure 4.9: Average reward and steps per episode over a training run of the PPO agent. Data from [23].

Each of the best trained agents carried out 5000 testing episodes with varying initial conditions. These initial conditions are distributed over the same range as in training. Figure 4.11 shows the distributions of terminal velocity and position for both the reward-optimised and area-optimised agents, as well as the fuel consumption. Numerical results are shown in Table 4.4. In these and all further tabulated results, terminal xy-position is calculated as $\| \begin{bmatrix} r_x & r_y \end{bmatrix} \|$, terminal xy-velocity as $\| \begin{bmatrix} v_x & v_y \end{bmatrix} \|$, and terminal z-velocity as $|v_z|$. Both agents consistently achieve x- and y-velocities within the desired range, however the area-optimised agent has slightly higher variance in the x-velocity. While both agents mostly reach the desired landing region within a $5m$ radius, there are several trajectories for both agents that do not satisfy this requirement. The most significant difference in performance between the agents is in the terminal z-velocity. The magnitude of the mean z-velocity of the reward-optimised agent is $1.827m/s$ whereas for the area-optimised agent it is $3.927m/s$. This means the area-optimised agent is consis-

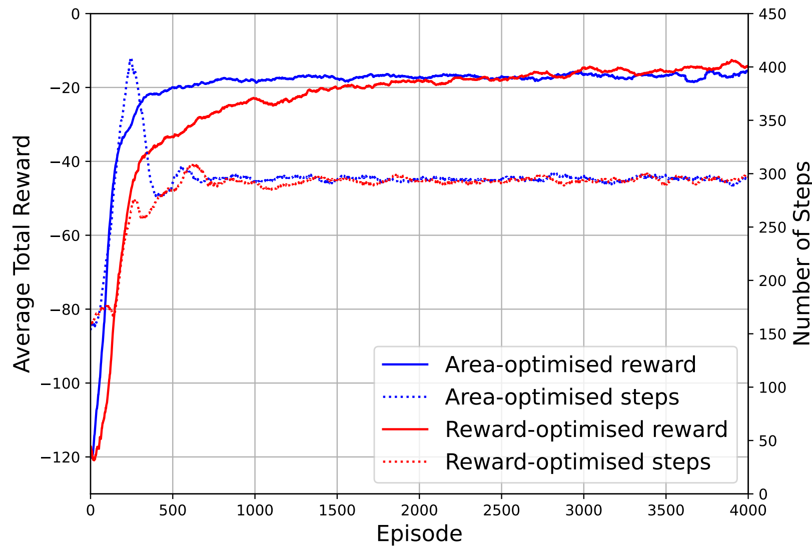
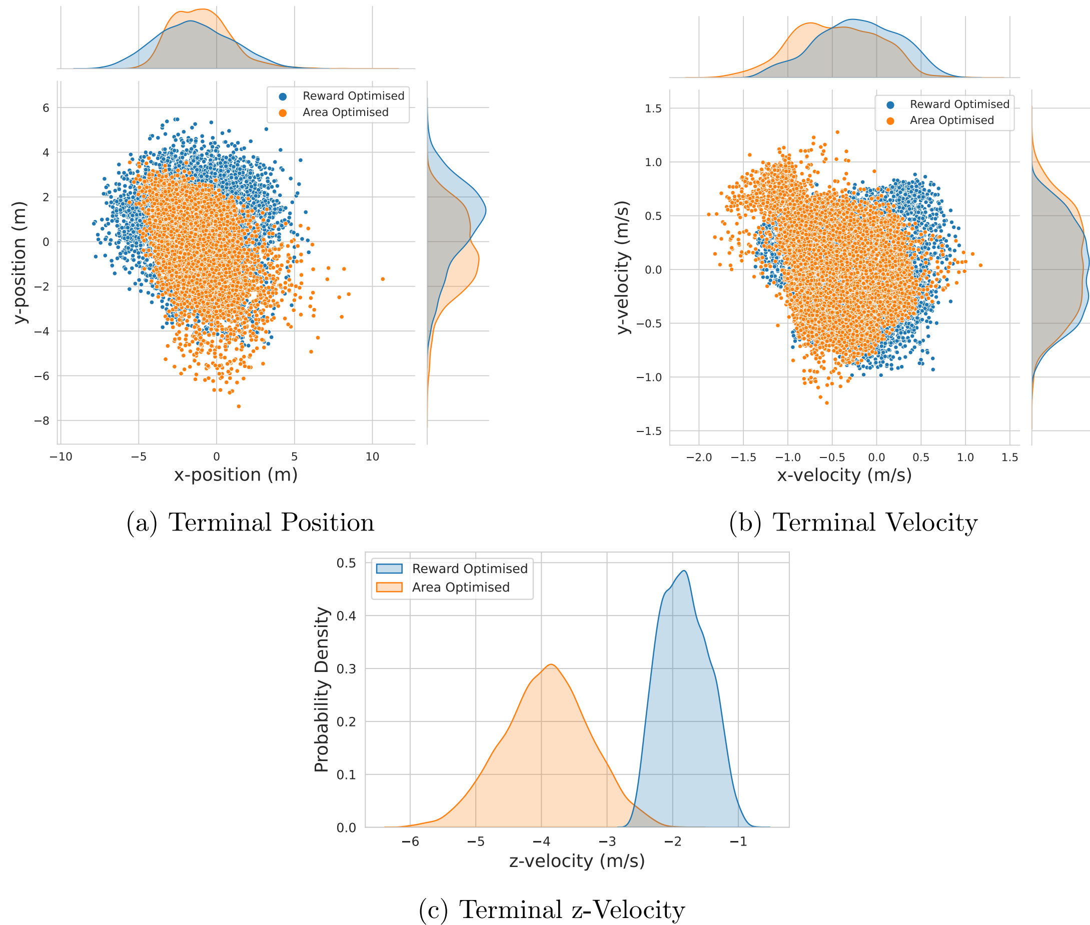


Figure 4.10: Average reward and steps per episode over a training run of DQN for two different sets of hyperparameters. Uniformly filtered (average) over 120 episodes for clarity.

tently unable to achieve the soft landing requirement, but the reward-optimised agent shows much better performance in achieving a soft pinpoint landing.

As a comparison to the above results, Table 4.5 shows the same statistics over 5000 testing episodes for an agent trained with PPO. This agent achieves a more precise landing with a maximum distance from the desired landing position of only $1.2m$, compared to $7.918m$ for the reward-optimised agent and $10.798m$ for the area optimised. Furthermore, the terminal velocity magnitudes are lower for the PPO agent. These differences can be attributed in part to the coarse discretisation for the DQN agents which limits their ability for finer control. This is particularly important for achieving the pinpoint landing over the final few metres of descent, and so training a separate controller for this final phase could improve the overall performance. In addition, increasing the action space size with more discrete magnitudes could improve the agent's landing precision, but at the expense of increased problem complexity. Finally comparing the fuel consumption, the averages for each agent are very similar: $291.1kg$ for PPO, 295.5 for area-optimised DQN, and 306.4 for reward-optimised DQN. The area-optimised agent has both the highest maximum fuel consumption at $408.8kg$ and the lowest minimum

with 254.9, however this is likely due to this agent rarely achieving a soft landing and therefore consuming less fuel to decelerate. Although the reward-optimised agent consumes more fuel than the area-optimised, it clearly shows superior landing performance and achieves comparable fuel consumption to the PPO agent.



(a) Terminal Position

(b) Terminal Velocity

(c) Terminal z-Velocity

Figure 4.11: Scatter and KDE plots showing distributions of terminal landing states for each agent.

Table 4.4: Comparison of test results from both agents trained with DQN. Statistics shown for terminal state over 5000 test episodes.

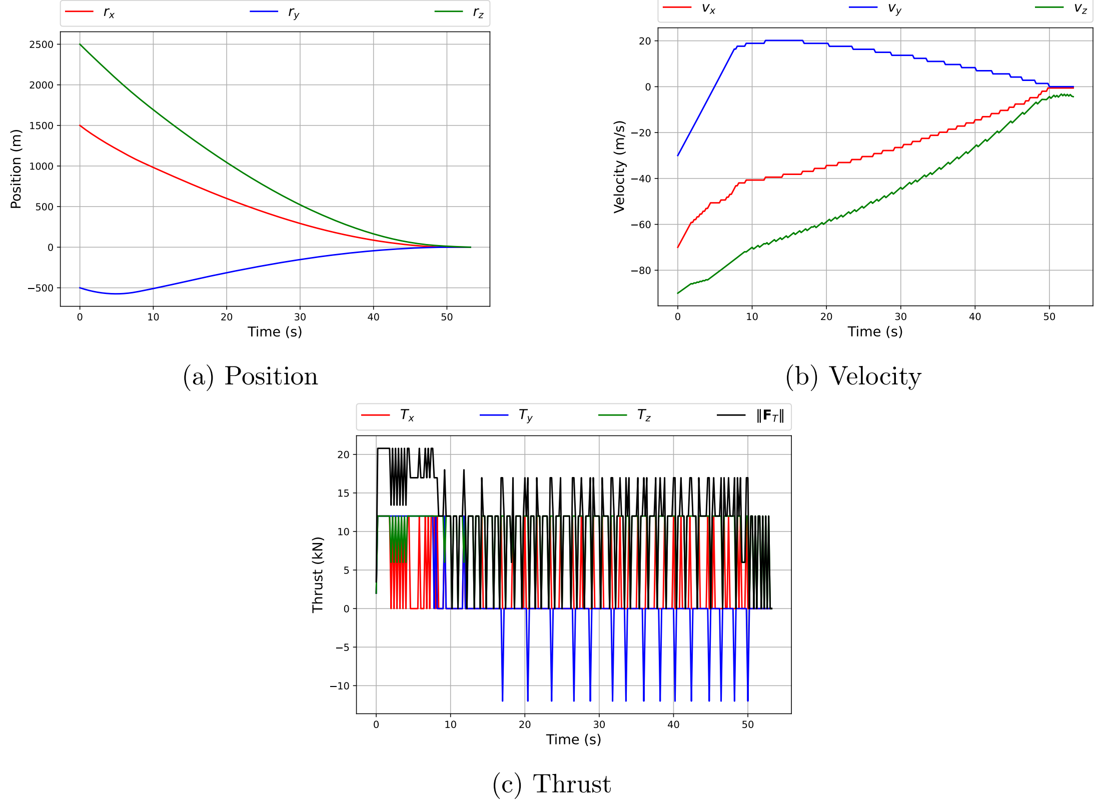
	Area-optimised				Reward-optimised			
	Mean	Min.	Max.	STD	Mean	Min.	Max.	STD
xy-position (m)	2.501	0.032	10.798	1.228	3.09	0.082	7.918	1.422
xy-velocity (m/s)	0.709	0.007	1.958	0.373	0.591	0.011	1.393	0.267
z-velocity (m/s)	3.927	1.865	6.029	0.659	1.827	0.714	2.647	0.361
Fuel (kg)	295.5	254.9	408.8	18.1	306.4	267.4	376.1	17.3

Table 4.5: Test results for agent trained with PPO. Statistics shown for terminal state over 5000 test episodes.

	PPO			
	Mean	Min.	Max.	STD
xy-position (m)	0.744	0.318	1.2	0.16
xy-velocity (m/s)	0.254	0.228	0.295	0.008
z-velocity (m/s)	0.719	0.107	0.913	0.12
Fuel (kg)	291.1	262.1	353.6	14.4

Figures 4.12 and 4.13 show an example trajectory of the area-optimised and reward-optimised agents. These show the position, velocity, and commanded thrust of a lander over the course of one episode. Both trajectories have an initial state of $\mathbf{r}_0 = \begin{bmatrix}1.5 & -0.5 & 2.5\end{bmatrix} km$ and $\mathbf{v}_0 = \begin{bmatrix}-70 & -30 & -90\end{bmatrix} m/s$ with an initial mass of $m_0 = 2000$. In both cases the position and velocity show an expected tendency towards zero with a low velocity in the z-direction over the final seconds of the episode. The reward-optimised agent has a terminal position $2.1m$ from the desired landing site with a terminal velocity of $2.4m/s$ and total fuel consumption of $316.7kg$. The area-optimised agent has a similar terminal position $1.3m$ from the desired landing site, but with a greater terminal velocity of $4.4m/s$ and total fuel consumption of $304.2kg$. This shows that the reward-optimised agent achieves a softer landing with lower vertical velocity than that of the area-optimised agent, which was common across testing episodes. For both agents the thrusts in the x- and y-directions show similar behaviour with a small difference in their magnitude as specified previously. On the other hand, while the reward-optimised agent frequently uses the midpoint action in the z-direction, the area-

optimised agent tends to switch between zero and its maximum thrust in this direction. It is possible that this represents a local minimum solution that the area-optimised agent quickly converged on, but with worse performance than the policy found after more exploration by the reward-optimised agent.



(a) Position

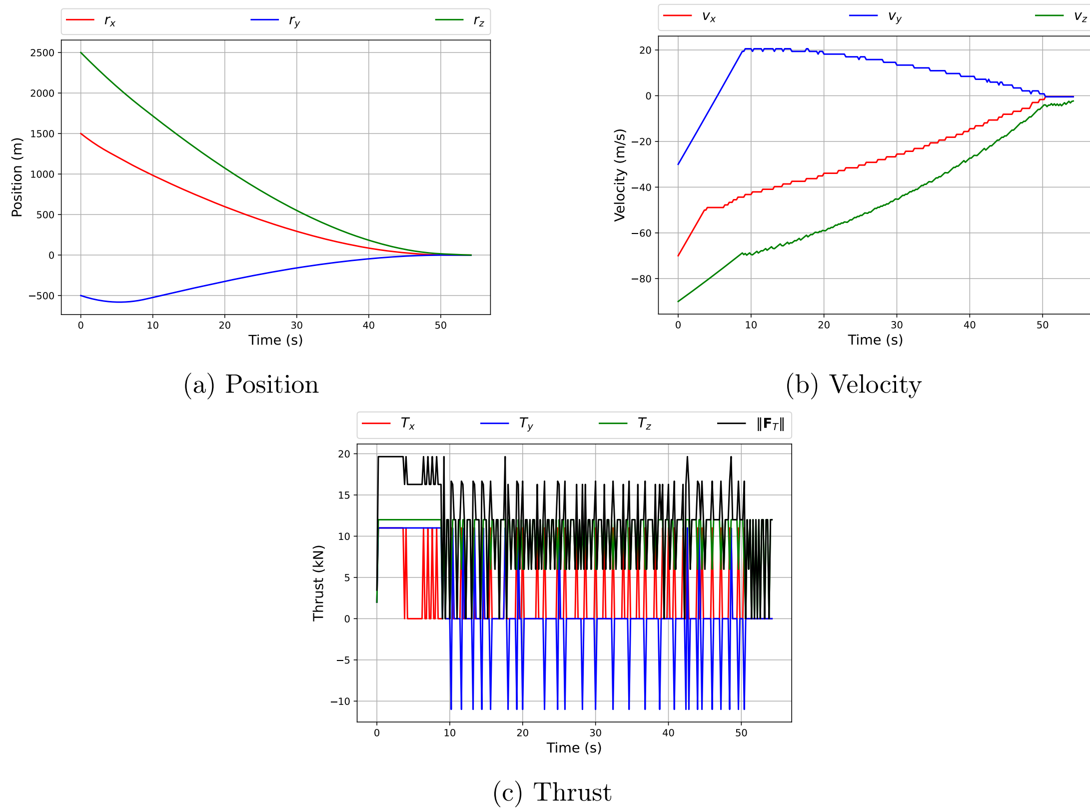
(b) Velocity

(c) Thrust

Figure 4.12: Trajectory of the area-optimised agent over a sample episode with commanded thrusts.

4.3.4 Training with Raw State Representation

Applying the same methodology as the above results for the shaped state, other agents were trained using a raw state representation (Equation 4.13). The same procedure for tuning hyperparameters was used with both area-optimised and reward-optimised losses. These gave the optimised hyperparameters as shown in Table 4.6. The action magnitudes used were the same as for the shaped state, with $T_x^{\max} = T_y^{\max} = 12kN$ for the area-optimised agent and $T_x^{\max} = T_y^{\max} = 11kN$ for the reward-optimised agent.



(a) Position

(b) Velocity

(c) Thrust

Figure 4.13: Trajectory of the reward-optimised agent over a sample episode with commanded thrusts.

These hyperparameters do not have the same patterns as those of the shaped state problem (Table 4.3). In particular the DNN structures do not have a more sparse middle hidden layer as with the shaped state hyperparameters. Furthermore, the values of ϵ_0 and N_ϵ show the opposite behaviour of what is expected, with reward-optimised favouring lower values for both compared to the area-optimised.

Table 4.6: Selected DQN hyperparameters when optimising for area under learning curve and final reward with a raw state representation.

Parameter	Area-optimised	Reward-optimised
$\tilde{N}_1$	105	120
$\tilde{N}_2$	150	200
$\tilde{N}_3$	165	130
α	4.67×10^{-5}	3.89×10^{-5}
C	65	65
ϵ_0	0.441	0.316
N_ϵ	1850	900
γ	0.941	0.923
k	95	120

Both sets of hyperparameters were used for a training run of 4000 episodes. Figure 4.14 shows the learning curves for these agents trained using each set of hyperparameters with the raw state representation. These do not converge on as high a cumulative reward as the shaped state agents, with a cumulative reward of around -40 per episode towards the end of training. Nevertheless, both agents appear to "learn" as their learning curves increase sharply throughout the early episodes. Despite the lower exploration probabilities for the reward-optimised agent, its initial cumulative reward decreases more sharply than the area-optimised agent.

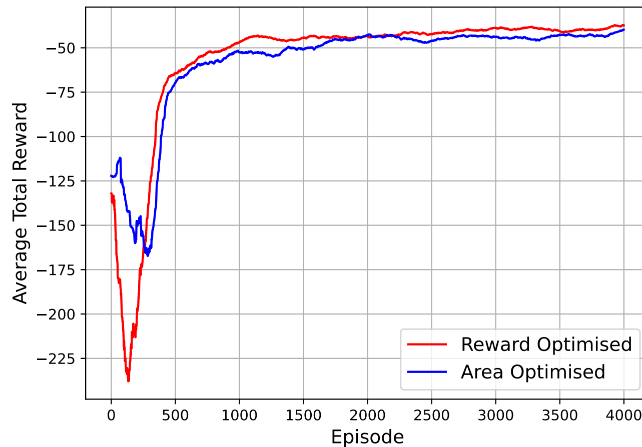


Figure 4.14: Learning curves for agents trained with a raw state representation.

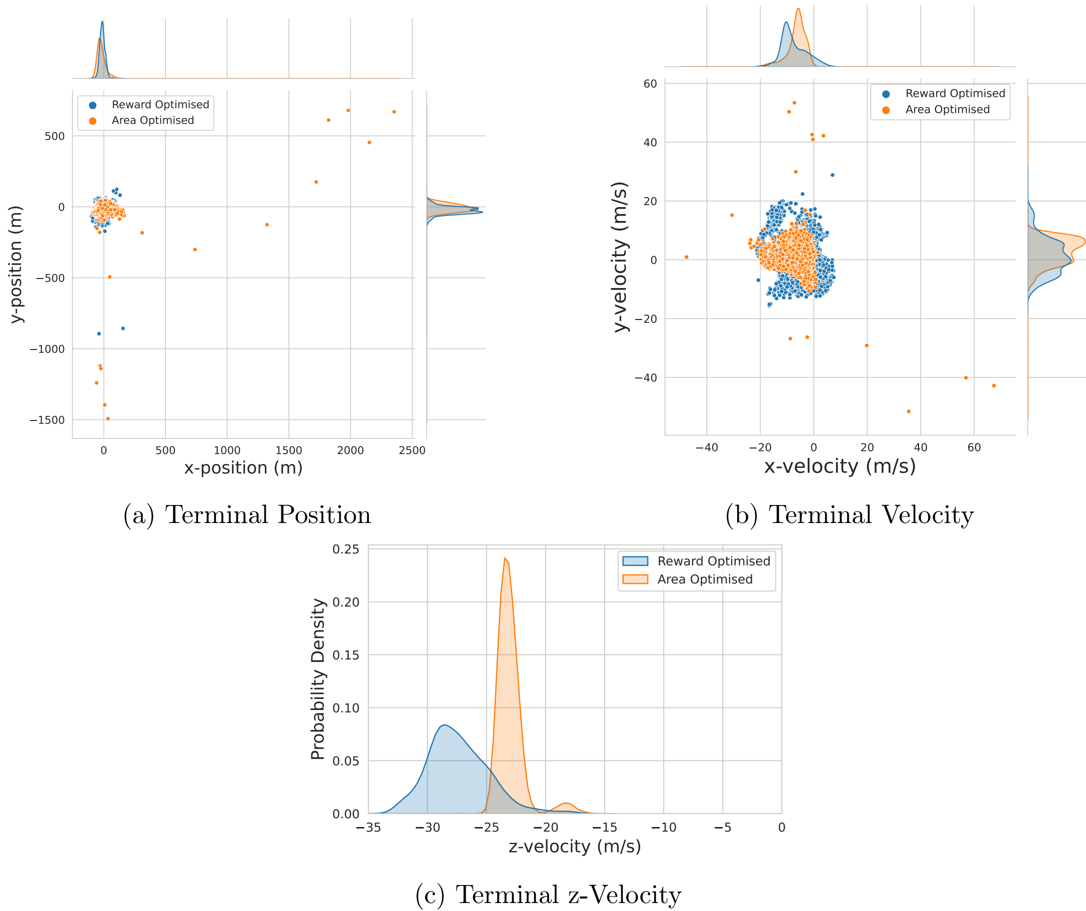


Figure 4.15: Scatter and KDE plots showing distributions of terminal landing states for each agent trained with a raw state representation. The z-velocity axis is truncated for clarity.

Testing both agents trained with a raw state representation revealed that their performance in terms of achieving a pinpoint soft landing is very poor. Table 4.7 summarises the performance of both agents over 5000 test episodes in terms of the terminal state, which have very large mean and maximum values across all measurements. This is highlighted in Figure 4.15, which clearly shows the larger spread of horizontal positions and velocities in the x- and y-directions compared to the agents using the shaped state. Although some terminal states are clustered near the desired values of 0, in both the x- and y-positions and velocities there are several outliers far away from this desired state. In addition, the z-direction velocity magnitude is very high with both agents mostly having terminal z-velocities of magnitude greater than $15m/s$. This is likely due

Table 4.7: Comparison of test results from both agents trained with a raw state representation. Statistics shown for terminal state over 5000 test episodes.

	Area-optimised				Reward-optimised			
	Mean	Min.	Max.	STD	Mean	Min.	Max.	STD
xy-position (m)	48.15	0.681	2473	89.02	40.52	0.423	895	28.09
xy-velocity (m/s)	8.457	0.26	79.8	3.877	10.1	0.314	29.68	3.956
z-velocity (m/s)	23.02	1.898	79.46	1.749	27.38	15.74	89.82	2.86
Fuel (kg)	312.6	248.7	551.6	24.9	300.8	180.8	364.0	20.6

to the nature of the reward function, which rewards the agent for following a target velocity. While the agent has learned to follow this velocity over the start of the trajectory, to avoid receiving the negative rewards over the final descent, where the target velocity decreases more sharply, it accelerates towards the end instead of decelerating. This behaviour is commonly referred to as "reward-hacking" where an agent exploits the reward function to converge on an undesirable policy [133].

4.4 Summary

This chapter described the general procedure for applying RL to a spacecraft powered descent problem. First the problem is defined in terms of its state space, action space, and reward function. To select appropriate hyperparameters for the agent, they undergo a tuning process which can consider different objectives. In this example, two different objectives related to the cumulative reward in training episodes were used. Multiple agents were trained with tuned hyperparameters to assess learning performance across random seeds and the best performing agents then carried out many test episodes.

The results shown highlight the need to appropriately define the RL problem to avoid behaviour such as reward-hacking shown by the agents trained with a raw state representation. Furthermore, the choice of optimisation criteria for tuning the agent's hyperparameters affects the performance of the trained agent. Tuning for hyperparameters that prioritise speed of learning as well as longer term performance resulted in less effective policies. The following chapter will introduce another approach for improving an agent's policy that can also prevent reward-hacking behaviour.

Chapter 5

Learning with Demonstrations

As discussed in Chapter 2, RL has its roots in optimal control theory. However, current state-of-the-art RL approaches, through their use of function approximators, mostly do not have the guarantees of optimality that are given by traditional optimal control approaches. It is also typical to consider RL problems in a "learning from scratch" manner where the agent has no prior knowledge of the environment. In many problems of practical interest, this is not the case. For example, traditional optimal control can find solutions for many systems under certain assumptions. Therefore, an agent could exploit such knowledge while learning to control the system. This is the idea behind learning with demonstrations.

In situations where demonstrations are available, this raises the question of why it is necessary to then train a separate agent. One potential reason is execution speed. If the process to generate demonstration data is computationally expensive, it may be advantageous to train an agent offline that can approximate the process used to generate the demonstrations online more quickly. Another reason is to deal with uncertainties. For example, if it is possible to generate demonstrations given certain limiting assumptions on an environment, a RL agent could use these demonstrations plus learning from interaction to find a near-optimal policy that handles uncertainties that are not considered in the demonstrations.

There are several methods for finding optimal solutions to the spacecraft powered descent problem—see Section 4.1. These solutions may not be robust to environmental

uncertainties, which is the motivation for applying RL to this problem. Furthermore, demonstrations are particularly useful in tasks with sparse rewards. Without demonstrations, an agent may need to explore for a very long time before finding the desired state. In the case of powered descent, the positive reward given for a successful landing is a sparse reward. Although there are approaches to define a shaped reward function that guides the agent to the desired terminal state, these might not be sufficient to learn an effective policy as shown in the previous chapter. Optimal control demonstrations could, therefore, be beneficial in training a RL agent for powered descent.

This chapter presents a method for exploiting optimal control demonstrations in RL, with the following main contributions:

- A heuristic method to discretise actions in optimal control problems to create trajectories suitable for use as demonstrations.
- A novel approach for incorporating demonstrations in off-policy RL algorithms.
- Application of this method to spacecraft powered descent to show how it can overcome reward-hacking.

The remainder of this chapter is organised as follows. Section 5.1 describes relevant literature in the area of RL with demonstrations. Section 5.2 introduces the proposed method of incorporating optimal control demonstrations into RL. The method for creating optimal trajectories in the powered descent problem is shown in Section 5.3. Results of the optimal trajectories and training with demonstrations are presented in Section 5.4.

5.1 Relevant Literature

There are many existing approaches for combining optimal control with various ML techniques to exploit the benefits of each. In most cases, a ML method aims to directly approximate optimal control solutions without needing to run a potentially expensive optimisation. This will often follow the same general procedure of generating a number of optimal trajectories, training a model such as a NN on these in a supervised manner,

then using this model to directly output new trajectories. This gives the advantage of not only being faster but also potentially more robust to uncertainties. For example, in [134] the authors train a NN to approximate optimal control actions for planetary landing without the need to run online optimisations. Similarly, in [135] they reduce the optimal control problem to defining a small set of parameters that vary based on the initial conditions. Again, NNs are trained to learn the mapping from initial conditions to optimal control policy. Other more advanced deep learning architectures such as transformers have also been applied to the powered descent problem [136]. In this case, the attention mechanism in the transformer allows it to predict long sequences of time-series data. As well as providing the mapping from states to optimal actions, in [137] NNs are also used to model the gravity dynamics of an asteroid for generating optimal trajectories. Since the performance of many optimisation algorithms is sensitive to the initial guess, NNs can also learn to generate suitable initial guesses. This is the approach employed in [138] for generating optimal asteroid landing trajectories. Another example application in proximity operations compares training via PPO to a deep-learning approach that trains on optimal control demonstrations [139]. These approaches to including demonstrations only learn in a supervised manner without any further training via interaction with the environment.

RL methods that aim to mimic expert demonstrations are broadly referred to as imitation learning [140]. This has a similar motivation to the ML approaches described above, where the agent's goal is to imitate the demonstration policy. In addition to learning from demonstration, RL methods also benefit from their self-learning capability. One of the early examples of such an approach that combines an expert and self learning is the algorithm DAGGER [141]. This is designed to avoid costly mistakes by the agent while learning by including demonstrations from an "expert" that guarantees a certain performance by the RL agent. Later work extended this to RL methods using DNNs [142]. In both these methods, the agent requires expert demonstrations throughout its training which may be limiting in applications where these demonstrations are difficult to obtain. Another approach similar to the method proposed here is Deep Q-learning from Demonstrations [143]. In this case, the agent uses a potentially

small set of demonstrations to initialise its policy in a supervised manner. Following this initialisation, the RL updates use a combination of loss functions that aim to give greater weight to the expert demonstrations. This concept of using demonstrations to prime a RL agent in a supervised manner is well studied and most often applied to robotics problems [140, 144, 145].

Similar approaches of including demonstrations in RL have also been applied to problems in astrodynamics. The authors in [146] use such an approach for trajectory generation. An initial policy is generated using solutions to a deterministic optimal control problem to train an agent in a supervised manner. This agent then learns a policy that is designed to be robust against missed thrust events. Other work uses demonstrations from optimal control to initialise an agent for powered descent [20]. This method uses pseudospectral optimisation to generate the demonstrations and only uses these demonstrations for initialisation prior to training via RL.

The methods described above for combining optimal control demonstrations with RL use some form of supervised learning alongside the RL updates. This differs from the method proposed here which foregoes any supervised initialisation and only uses TD-based updates. Similarly to Deep Q-learning from Demonstrations, the proposed method uses demonstration data for updates throughout training but does not require generation of new trajectories by an "expert" throughout training. Despite the relative simplicity of the proposed approach compared to other methods of including demonstrations, as will be shown it can still learn policies that avoid reward-hacking behaviour.

5.2 DQN with Optimal Demonstrations

This section describes the method for incorporating demonstrations from optimal control trajectories into DQN. It is applicable to any RL problem that can be solved as a discrete-time optimal control problem assuming deterministic dynamics.

5.2.1 Optimal Control Problem

The goal of a general optimal control problem is to minimise a cost function J subject to constraints on the state $\mathbf{x}$ and control actions $\mathbf{u}$. This can be expressed as shown in Equations 5.1 - 5.5 [147].

$$
\begin{aligned}
&\text{minimise} \quad && J(\mathbf{x}) = \int_{t_0}^{t_f} L\left(t, \mathbf{x}(t), \mathbf{u}(t)\right) dt && (5.1)\\
&\text{subject to} \quad && \dot{\mathbf{x}} = f(t, \mathbf{x}(t), \mathbf{u}(t)), && (5.2)\\
& && \mathbf{u}(t) \in \mathbf{U} \; \forall t \in [0, t_f], && (5.3)\\
& && \mathbf{x}(t) \in \mathbf{X} \; \forall t \in [0, t_f], && (5.4)\\
& && \mathbf{x}(0) = \mathbf{x}_0, \; \mathbf{x}(t_f) = \mathbf{x}_f, && (5.5)
\end{aligned}
$$

where t_f is the final time, L is a function that represents the optimisation objective, f is a mapping defining the system dynamics, $\mathbf{U}$ is the space of possible actions, $\mathbf{X}$ is the space of allowable states, and $\mathbf{x}_0$ and $\mathbf{x}_f$ are the specified initial and final states respectively. Importantly, the dynamics represented by f are considered deterministic.

The general form of an optimal control problem shown above represents an infinite-dimensional problem. This problem can be discretised in the time domain to give a finite-dimensional problem that can be solved with numerical optimisation procedures. Discrete timesteps are denoted t_k and are separated by a constant step size Δt. The state and action constraints of Equations 5.3 and 5.4 are applied at each timestep t_k. In addition, the dynamics are adjusted to account for the discrete time control inputs, for example by applying a zero-order hold to the commanded actions between timesteps. The output of this optimisation is a series of real-valued control commands at each discrete timestep, which forms the basis for the optimal control demonstrations.

5.2.2 Demonstrations from Optimal Trajectories

The DQN algorithm operates over discrete action spaces. Therefore, the continuous actions generated via the optimal control problem must be converted into a discrete action space. One simple way to compute the corresponding discrete action is to round

the action to the nearest discrete magnitude. However, this accumulates errors over time and results in trajectories far from the optimal trajectory. This can be remedied by using a heuristic action selection that rounds the action up or down depending on the velocity error as shown below.

For each direction, let n denote the number of discrete actions in that direction, $\bar{a}^*$ denote the optimal continuous action normalised between 0 and 1, and $\bar{a}$ denote the discrete action. Equation 5.6 shows the heuristic action selection method that uses the error between the spacecraft's current velocity and the velocity at the next timestep in the optimal trajectory.

$$\bar{a} = \frac{\lfloor \bar{a}^*(n-1) \rfloor + \frac{1}{2}(\mathrm{sgn}(\delta v) + 1)}{n-1}, \tag{5.6}$$

$$\delta v = \dot{x}^*(t+1) - \dot{x}(t), \tag{5.7}$$

where δv and $\bar{a}$ are calculated separately for each direction. At the end of the trajectory where the agent is close to the landing region, $\dot{x}^*(t+1)$ is held constant at zero in all directions. Although this improves the resulting trajectories compared to simply rounding the actions, the final state may still not be in the desired landing region. To ensure the trajectory with discretised actions reaches the desired terminal state, the initial state can be adjusted using an approach similar to Hindsight Experience Replay (HER) [148]. This is where the final state in the trajectory is assumed to be the goal and the states over the course of the trajectory are updated accordingly. In practice for this problem, this is achieved by updating in turn the initial velocity and position as shown in Equations 5.8 and 5.9.

$$\dot{\mathbf{x}}_0 = \dot{\mathbf{x}}_0^* - \dot{\mathbf{x}}(t_f)|_{\mathbf{x}(0)=\mathbf{x}_0^*,\, \dot{\mathbf{x}}(0)=\dot{\mathbf{x}}_0^*}, \tag{5.8}$$

$$\mathbf{x}_0 = \mathbf{x}_0^* - \mathbf{x}(t_f)|_{\mathbf{x}(0)=\mathbf{x}_0^*,\, \dot{\mathbf{x}}(0)=\dot{\mathbf{x}}_0}, \tag{5.9}$$

where $\mathbf{x}_0^*$ and $\dot{\mathbf{x}}_0^*$ are the initial position and velocity used in the optimal control problem. Provided the position and velocity error at the terminal state are not too large, this results in a full trajectory that approximates the optimal trajectory with discrete actions that can be used as demonstrations.

5.2.3 Demonstration Experience Replay

DQN is an off-policy RL algorithm that can learn from experiences generated using arbitrary policies. For this reason, optimal control demonstrations can be incorporated into training a DQN agent along with its own experience. As described in Chapter 2, in the standard DQN algorithm, experiences are stored in the replay memory, $\mathcal{D}$ up to a certain maximum capacity on a first-in-first-out basis. During training, experiences are randomly sampled uniformly from the replay memory in batches of size k. The demonstration experience replay, denoted $\mathcal{D}_{demo}$, contains experiences from the discretised optimal trajectories. These take the same form as in $\mathcal{D}$ of tuples of $(s_t, a_t, r_{t+1}, s_{t+1})$ that define a state transition. When updating the agent, in addition to the k samples from $\mathcal{D}$, k_{demo} state transitions are sampled uniformly from $\mathcal{D}_{demo}$ and are also used to update the agent.

Algorithm 2 shows pseudocode for DQN with demonstrations. First, the demonstration trajectories are generated for a set of initial states defined by the initial positions $\mathbf{x}_0^*$ and velocities $\dot{\mathbf{x}}_0^*$. These are used to create the demonstration experience replay $\mathcal{D}_{demo}$ which are then used along with the agent's own experience to update the network parameters as in the conventional DQN algorithm.

5.3 Powered Descent Optimal Control Problem

This section describes the method for generating optimal trajectories in the powered descent problem. First, the general dynamics and constraints for powered descent optimal control are described, followed by the convexification of this problem.

5.3.1 Optimal Control Problem

The dynamics of the environment are the same as presented in Chapter 4 (Equations 4.1 to 4.3). The optimisation aims to minimise fuel consumption while achieving a pinpoint landing defined by $\mathbf{r}(t_f) = \dot{\mathbf{r}}(t_f) = 0$. In addition to constraints on the final position and velocity of the lander, several other constraints on the state and control actions are often defined. These additional constraints may represent physical limitations of

Algorithm 2 Deep Q-Networks with optimal control demonstrations

Inputs: initial states $\mathbf{x}_0^*$, $\dot{\mathbf{x}}_0^*$ for running optimal trajectories, number of training episodes N_{ep}, agent hyperparameters

Optimal control demonstrations

1: **for** each initial state $\mathbf{x}_0^*$, $\dot{\mathbf{x}}_0^*$ **do**
2: calculate optimal actions for all discrete timesteps t_k
3: discretise optimal actions according to Equation 5.6
4: adjust initial velocity and position according to Equations 5.8 and 5.9
5: add all state transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ to demonstration memory $\mathcal{D}_{demo}$
6: **end for**

DQN with demonstrations

7: initialise network parameters $\boldsymbol{\theta}$ with random weights
8: **for** $ep = 1$ to N_{ep} **do**
9: initialise state $s_t \leftarrow s_0$
10: **while** state s_t is non-terminal **do**
11: select action a_t according to policy π
12: execute action a_t and observe r_{t+1}, s_{t+1}
13: update memory $\mathcal{D}$ with $(s_t, a_t, r_{t+1}, s_{t+1})$
14: select random minibatch of k experiences (s_j, a_j, r_j, s'_j) from $\mathcal{D}$
15: select random minibatch of k_{demo} experiences (s_j, a_j, r_j, s'_j) from $\mathcal{D}_{demo}$
16: $\delta_j = r_j + \gamma \max_a Q_{\theta^-}(s'_j, a) - Q_\theta(s_j, a_j)$ $\forall$ j in minibatches
17: update parameters $\boldsymbol{\theta}$ using TD-error δ_j for each experience
18: after C time-steps set $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$
19: **end while**
20: **end for**

the lander or mission requirements. Relevant constraints on the lander's position and commanded thrusts are defined numerically below.

The glideslope constraint specifies a cone of valid positions for the spacecraft with vertex at the landing site, $\mathbf{r} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$. This ensures the lander will not go sub-surface and remains a safe distance from the ground until landing. This is expressed as

$$\|A\mathbf{r}(t)\| - \mathbf{c}^\top \mathbf{r}(t) \leq 0 \quad \forall t \in [0, t_f], \tag{5.10}$$

where

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{5.11}$$

$$\mathbf{c}^\top = \begin{bmatrix} 0 & 0 & 1/\tan(gs_{lim}) \end{bmatrix}, \tag{5.12}$$

and gs_{lim} is the minimum glideslope angle as in Equation 4.4. In addition to limits on the maximum thrust available, there may be a minimum thrust limit for the lander. The thrust magnitude constraint is

$$0 < \rho_1 \leq \|\mathbf{F}_T(t)\| \leq \rho_2, \tag{5.13}$$

where ρ_1 and ρ_2 are the minimum and maximum thrust magnitudes, respectively. As will be shown, for the purposes of generating useful demonstration trajectories it is beneficial to constrain the maximum thrust magnitude in the normal (z) direction. This is given by

$$|T_z| \leq T_z^{max}, \tag{5.14}$$

There may also be constraints on the lander's attitude to ensure navigation equipment points in the correct direction. In this 3-DOF formulation, this is represented by a constraint on the angle of the thrust vector. This thrust pointing constraint is given by

$$\hat{\mathbf{n}}^\top \mathbf{F}_T(t) \geq \|\mathbf{F}_T(t)\| \cos\phi, \tag{5.15}$$

where $\hat{\mathbf{n}}$ is a unit vector specifying the reference thrust direction and ϕ is the maximum angle difference between the reference and true thrust directions.

Using the constraints specified above, Equations 5.16 - 5.20 define the minimum-fuel powered descent optimisation problem. The dynamics shown in Equation 5.17 do not include disturbance forces.

$$\max_{t_f, \mathbf{F}_T(\cdot)} m(t_f) = \min_{t_f, \mathbf{F}_T(\cdot)} \int_0^{t_f} \|\mathbf{F}_T(t)\| \, dt \tag{5.16}$$

subject to

$$\ddot{\mathbf{r}} = \mathbf{g} + \frac{\mathbf{F}_T(t)}{m(t)} \quad \forall t \in [0, t_f], \tag{5.17}$$

$$\dot{m}(t) = -\alpha_T \|\mathbf{F}_T(t)\| \quad \forall t \in [0, t_f], \tag{5.18}$$

constraints given by 5.10, 5.13, 5.14, 5.15,

$$m(0) = m_0, \ \mathbf{r}(0) = \mathbf{r}_0, \ \dot{\mathbf{r}}(0) = \mathbf{v}_0, \tag{5.19}$$

$$\mathbf{r}(t_f) = \mathbf{v}(t_f) = 0. \tag{5.20}$$

This optimisation can be solved via nonlinear methods. However, the minimum thrust magnitude constraint of Equation 5.13 is nonconvex and the thrust pointing constraint of Equation 5.15 is also nonconvex when $\phi > \pi/2$. Furthermore, the mass consumption defined by Equation 5.18 results in a nonconvex constraint when the problem is discretised. Therefore, the optimisation problem as expressed above is not solvable via efficient convex programming methods.

5.3.2 Convexification of the Optimal Control Problem

As discussed previously, there are several approaches to convexifying this problem to allow fast computation of optimal trajectories. The following method comes from the work of Açikmeşe et al, where they define a convex relaxation of control constraints [100, 102]. A change of variables gives new control inputs σ and $\mathbf{u}$:

$$\sigma = \frac{\Gamma}{m} \quad \text{and} \quad \mathbf{u} = \frac{\mathbf{F}_T}{m}, \tag{5.21}$$

where Γ is a slack variable used to convexify the thrust constraints. To deal with the nonlinearity resulting from the mass profile $m(t)$, an additional variable z is defined as $z = \ln m$. Equations 5.22 and 5.23 give a linear approximation of the thrust magnitude constraint in terms of the control variables σ and $\mathbf{u}$:

$$\|\mathbf{u}(t)\| \leq \sigma(t) \quad \forall t \in [0, t_f], \tag{5.22}$$

$$\rho_1 e^{-z_0(t)} \left(1 - \delta z(t) + \frac{\delta z(t)^2}{2}\right) \leq \sigma(t) \leq \rho_2 e^{-z_0(t)} \left(1 - \delta z(t)\right) \quad \forall t \in [0, t_f], \tag{5.23}$$

where

$$\delta z(t) = z(t) - z_0(t), \tag{5.24}$$

$$z_0(t) = \ln(m_0 - \alpha_T \rho_2 t). \tag{5.25}$$

The maximum thrust constraint is defined in terms of the control variables as

$$\mathbf{u}_z(t) \leq \frac{\sigma_z(t)}{\rho_2} T_z^{max}. \tag{5.26}$$

The final constraint to convexify is the thrust pointing constraint, which is then expressed as

$$\hat{\mathbf{n}}^\top \mathbf{u}(t) \geq \sigma(t) \cos\phi. \tag{5.27}$$

Using these convex constraints and the new control variables, Equations 5.28 - 5.33 define a new optimal control problem that is a convex relaxation of the problem presented above.

$$\min_{t_f, \mathbf{u}(\cdot), \sigma(\cdot)} \int_0^{t_f} \sigma(t) dt \tag{5.28}$$

$$\text{subject to} \qquad \ddot{\mathbf{r}} = \mathbf{g} + \mathbf{u}(t) \quad \forall t \in [0, t_f], \tag{5.29}$$

$$\dot{z}(t) = -\alpha_T \sigma(t) \quad \forall t \in [0, t_f], \tag{5.30}$$

$$\text{constraints 5.10, 5.22, 5.23, 5.26, 5.27,} \tag{5.31}$$

$$z(0) = \ln(m_0),\ \mathbf{r}(0) = \mathbf{r}_0,\ \dot{\mathbf{r}}(0) = \mathbf{v}_0, \tag{5.32}$$

$$\mathbf{r}(t_f) = \dot{\mathbf{r}}(t_f) = 0. \tag{5.33}$$

To solve this convex problem numerically requires discretising the time domain into N_t intervals with timestep Δt such that the time of flight is given by $t_f = N_t \Delta t$. This results in a finite-dimensional problem of optimising the control actions at each timestep. The total time at each timestep is given by

$$t_k = k\Delta t, \quad k = 0, \ldots, N_t. \tag{5.34}$$

Each of the constraints defined above must be satisfied at each timestep. The continuous actions are approximated as a zero-order hold between timesteps and the discrete dynamics of the problem are calculated via Euler integration. Given state variables $\mathbf{x}$ defined as

$$\mathbf{x}(t_k) = \begin{bmatrix} \mathbf{r}(t_k) \\ \dot{\mathbf{r}}(t_k) \end{bmatrix} \tag{5.35}$$

the discrete dynamics can be expressed as

$$\mathbf{x}(t_{k+1}) = \mathbf{x}(t_k) + \dot{\mathbf{x}}(t_k)\Delta t \tag{5.36}$$

$$= \mathbf{x}(t_k) + (\mathrm{A}\mathbf{x}(t_k) + \mathrm{B}(\mathbf{g} + \mathbf{u}(t_k)))\,\Delta t \tag{5.37}$$

with the matrices A and B defined as

$$A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \qquad (5.38) \qquad\qquad B = \begin{bmatrix} 0 \\ I \end{bmatrix}. \qquad (5.39)$$

Similarly, the discrete mass depletion dynamics are

$$z(t_{k+1}) = z(t_k) - \alpha_T \sigma(t_k). \tag{5.40}$$

This defines the discrete-time powered descent optimal control problem for a fixed time-of-flight t_f. To solve for the optimal time-of-flight, a line search is performed for values of $t_f \in [t_{min},\ t_{max}]$ and the value with the lowest loss following optimisation is selected.

5.4 Results of DQN with Demonstrations Applied to Powered Descent

5.4.1 Optimal Trajectories

Table 5.1 shows the minimum and maximum initial positions and velocities for which optimal trajectories were generated, which are near the extremes of the initial conditions for training the agent. These were computed for all combinations of the minimum and maximum values of each variable, plus the midpoint of all variables. This gives a total of 33 initial conditions from which optimal trajectories were generated. The initial mass is $2000kg$ and the initial altitude is $2.4km$ across all runs. Parameters defining the state and action constraints are as follows: $\rho_1 = 0N$, $\rho_2 = 19.2kN$, $gs_{lim} = 30°$, $\phi = 90°$, $T_z^{max} = 12kN$. Except for the maximum T_z thrust, these parameters are the same as defined in other works which are intended to be representative of the constraints on a real lander [102]. The value of T_z^{max} was chosen to give a suitable thrust profile as shown below.

Figure 5.1 shows a histogram of minimum glideslope angle for the optimal trajectories. Due to the constraint defined in Equation 5.10, the glideslope must be greater than $gs_{lim} = 30°$ for the whole trajectory. All of the trajectories satisfy this constraint, but 10 of the trajectories come within $2°$ of the glideslope limit.

Table 5.1: Range of initial conditions for generating optimal trajectories.

Parameter	Min. Value	Max. Value
Downrange position, r_x (m)	1000	2000
Crossrange position, r_y (m)	-1000	1000
Downrange velocity, v_x (m/s)	-70	-10
Crossrange velocity, v_y (m/s)	-30	30
Elevation velocity, v_z (m/s)	-90	-70

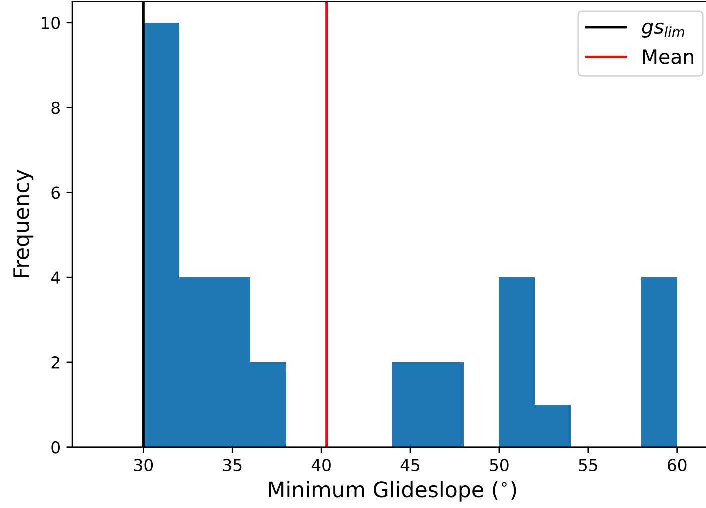
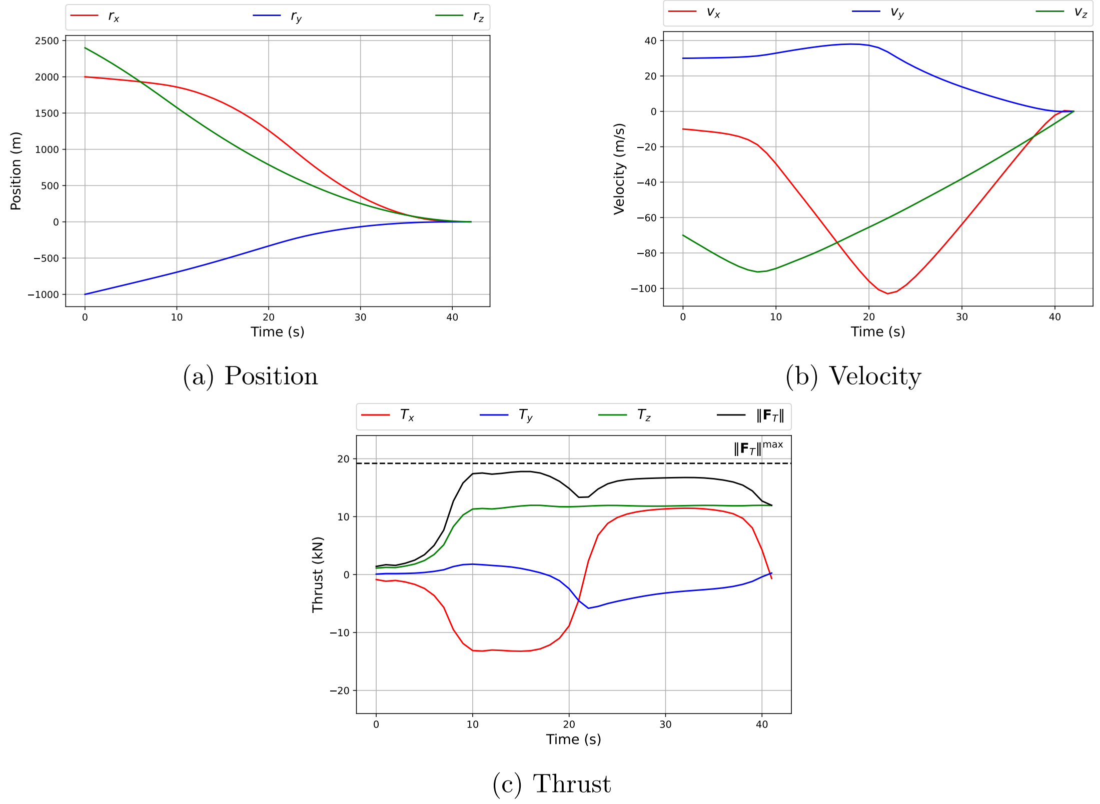


Figure 5.1: Histogram of minimum glideslope angle over optimal trajectories starting from different initial conditions.

Figure 5.2 shows an example optimised trajectory where the initial state is $\mathbf{r}_0 = \begin{bmatrix} 2 & -1 & 2.4 \end{bmatrix} km$ and $\mathbf{v}_0 = \begin{bmatrix} -10 & 30 & -70 \end{bmatrix} m/s$. In this case the thrust in the z-direction gradually increases to near its peak value and remains there for the duration of the trajectory. The velocity profile in each direction tends to increase to a maximum magnitude before decreasing almost linearly to zero.

For comparison, Figure 5.3 shows the trajectory from the same initial state without the constraint on T_z^{max}. This has a bang-bang control profile where the thrust magnitude varies between the maximum magnitude and zero. The mean fuel consumption across the trajectories is $241kg$ without the T_z^{max} constraint compared to $340.1kg$ with this constraint. Although it has lower fuel consumption, the thrust profile without the T_z^{max} constraint is more difficult to discretise and consequently less suitable for use as



(a) Position (b) Velocity

(c) Thrust

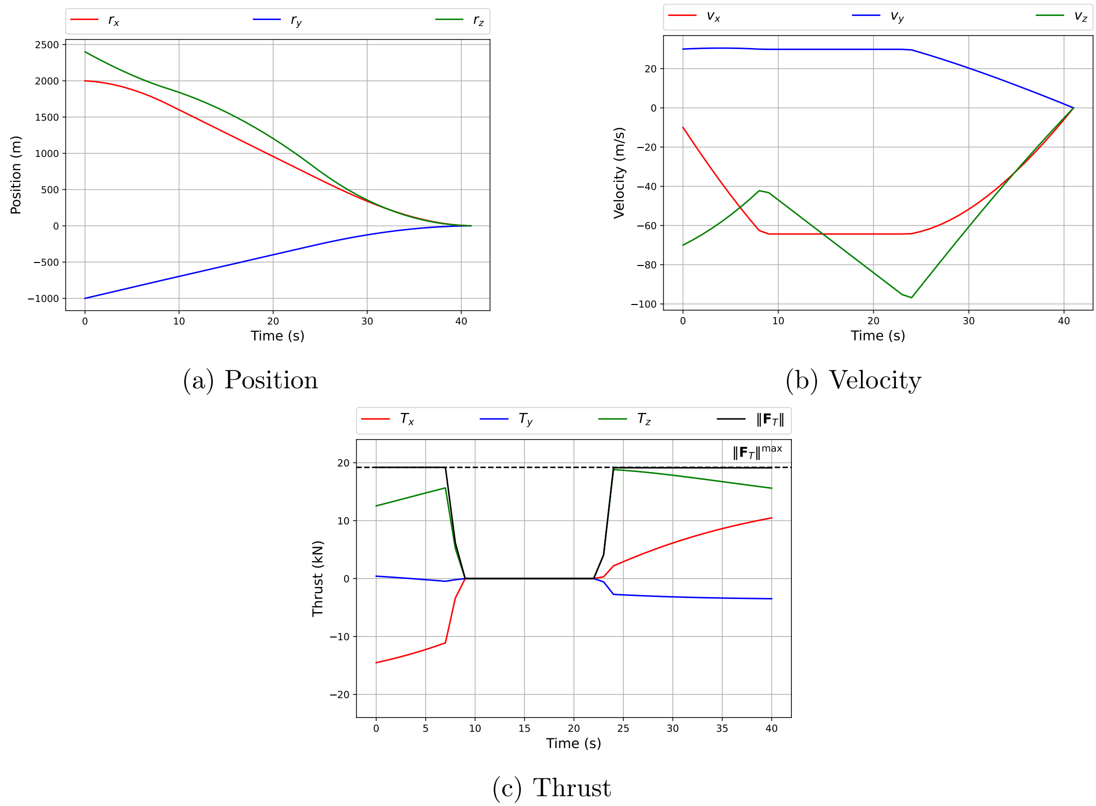
Figure 5.2: Example optimal trajectory with constraints on the maximum z-direction thrust.

demonstrations. On the other hand, the smoother control profile of the trajectory in Figure 5.2 can used as demonstrations following discretisation.

5.4.2 Discretised Optimal Trajectories

The action space size is the same as defined in Chapter 4 with 3 discrete actions in each direction (Equations 4.16-4.18). Since the z-direction only has positive thrusts, this action space satisfies the thrust pointing constraint of Equation 5.15 when $\phi = 90°$. Based on the results of the optimal control problem, the value of $T_z^{max} = 12kN$ can also be used to define the discrete action space with the midpoint thrust $T_z^{mid} = 6kN$. Given the maximum total thrust magnitude, the thrust magnitudes in the x- and y-directions can be selected as

$$T_x^{max} = T_y^{max} = \sqrt{\frac{1}{2}\left((\rho_2)^2 - (T_z^{max})^2\right)}. \tag{5.41}$$



(a) Position

(b) Velocity

(c) Thrust

Figure 5.3: Example optimal trajectory without constraints on the maximum z-direction thrust.

This results in a value of $T_x^{max} = T_y^{max} = 10.59kN$.

Table 5.2 presents statistics of the discretised trajectories, where the position and velocity errors are the norm of the terminal values as shown in Equations 5.9 and 5.8. These discretised trajectories result in a lower mean fuel consumption than the continuous ones due to the change in their initial state. All of the changes in initial position and velocity to achieve the pinpoint soft landing are sufficiently small relative to their initial value such that the new initial states are near the original initial states used for optimisation.

Table 5.2: Results of discretised optimal trajectories.

	Mean	Min.	Max.	STD
Position error (m)	68.22	28.03	154.82	35.2
Velocity error (m/s)	4.86	3.41	7.06	1.1
Fuel (kg)	286	240.3	335.8	27.1

Although all the discretised trajectories achieve the desired pinpoint soft landing, some of them violate the glideslope constraint. This is shown in Figure 5.4, which is a histogram of minimum glideslope angle for the discretised trajectories. Compared to the optimal trajectories shown in Figure 5.1, the minimum glideslope tends to be lower for the discretised trajectories as indicated by the lower mean. As a result, 6 of the trajectories have glideslopes less than 30° and therefore do not achieve a positive reward at the end of the episode. Since most of the discretised trajectories are successful, they can still be used as demonstrations.

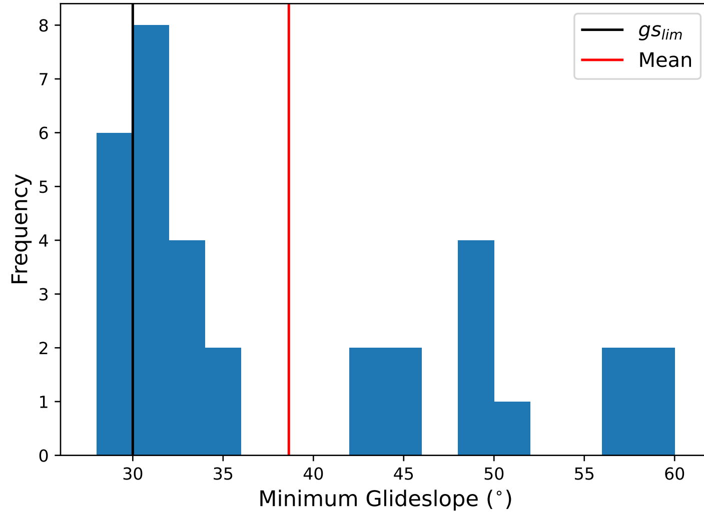
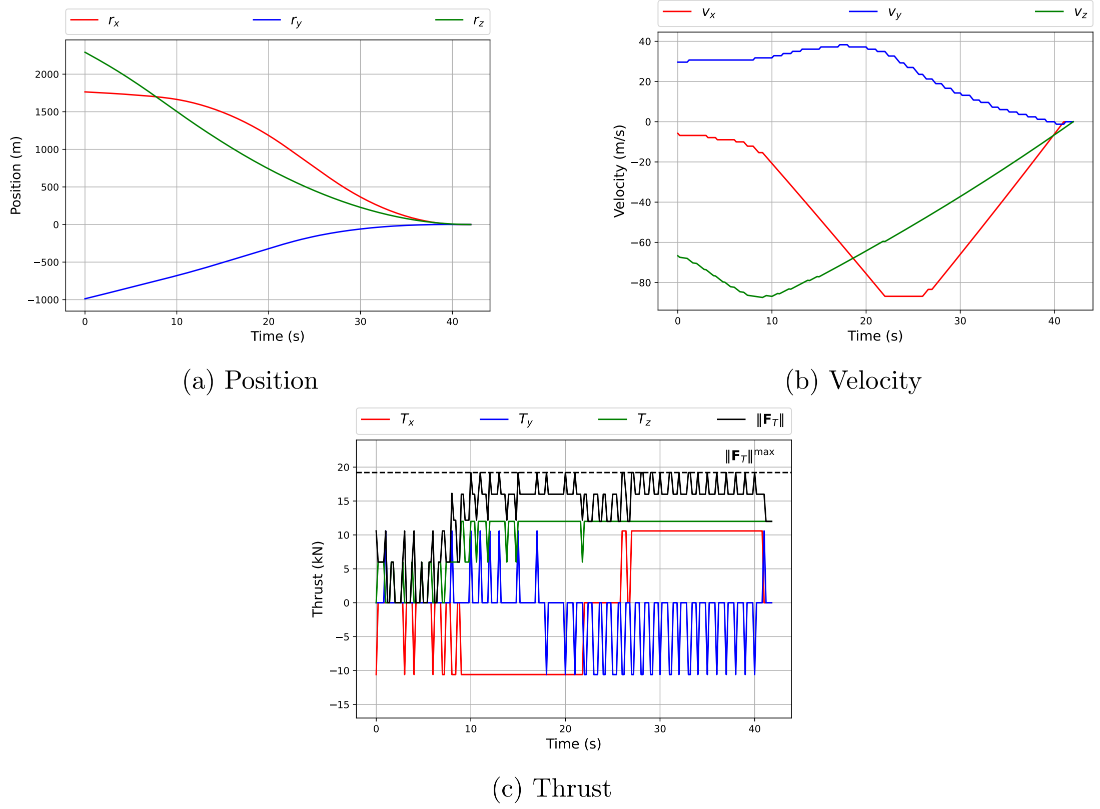


Figure 5.4: Histogram of minimum glideslope angle over discretised optimal trajectories starting from different initial conditions.

Figure 5.5 shows an example discretised trajectory. The adjusted initial state is $\mathbf{r}_0 = \begin{bmatrix} 1.764 & -0.987 & 2.291 \end{bmatrix} km$ and $\mathbf{v}_0 = \begin{bmatrix} -5.77 & 29.64 & -66.68 \end{bmatrix} m/s$, with the optimal initial state being the same as shown in Figure 5.2. The effect of action discretisation is clear when looking at the thrust profile which oscillates between discrete magnitudes in



(a) Position

(b) Velocity

(c) Thrust

Figure 5.5: Example optimal trajectory following discretisation of the actions.

each direction. Despite the noticeably different thrust profile, the position and velocity trajectory are similar to that of the continuous action trajectory shown in Figure 5.2.

5.4.3 Training Agents with Demonstrations

Using the demonstrations described above, a DQN agent was trained on the lander environment. The state representation is the raw state representation of Equation 4.13 with scaling as defined by Equation 4.14. The mass consumption rate α_T is defined as above and all other parameters of the environment and reward function are the same as used in Chapter 4.

Table 5.3 shows the hyperparameters used for training. These are the same as the reward-optimised agent trained with a shaped state representation described in Chapter 4 since this agent had the best performance. The best performing hyperparameters for DQN with demonstrations will likely be different to these, however these still achieve

reasonable performance as will be shown. The only difference in hyperparameters is the minibatch size, k which is split between the agent's own memory and demonstrations such that the total minibatch size is $k + k_{demo} = 104$. The agent was trained using a range of sizes of k_{demo}, where 0 indicates training without demonstrations and 104 is training with only demonstration data. As before, the activation function of the DNN is *tanh* and weight updates are performed using RMSProp optimisation with learning rate as shown.

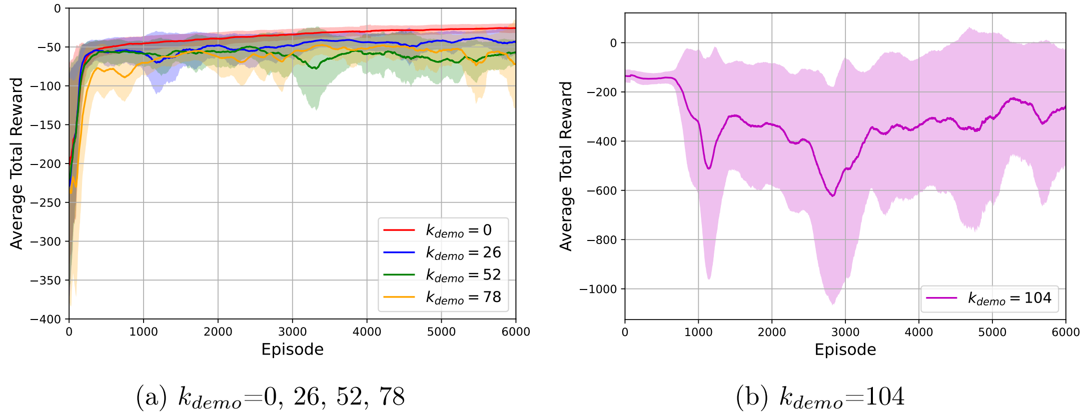
Table 5.3: Hyperparameters for training DQN with demonstrations.

Parameter	Value
$\tilde{N}_1$	150
$\tilde{N}_2$	65
$\tilde{N}_3$	165
α	2.08×10^{-5}
C	65
ϵ_0	0.269
N_ϵ	2700
γ	0.926
k	$104 - k_{demo}$
k_{demo}	$\{0, 26, 52, 78, 104\}$

The ranges of initial conditions used in training and testing are the same as previously defined in Table 4.1. For each value of k_{demo}, 8 agents were trained with different random seeds for 6000 episodes. The longer training time compared to the agents trained in Chapter 4 allows more interactions for the agent in the larger state space resulting from the raw state representation.

Figure 5.6 shows the learning curves averaged across runs for each value of k_{demo}. The agent trained without demonstrations has the highest average reward at the end of training and the lowest variance across runs for most of the episodes. The learning curves with demonstrations show similar trends to each other except for that of entirely demonstrations, shown separately in Figure 5.6b. In this case the agent begins with a higher average reward than the other agents for the earliest episodes. However, its learning quickly stagnates and then diverges after around 800 episodes causing large variations in the total reward. This shows that the method proposed here is not suit-

able for directly imitating the optimal trajectories without additional learning, since the demonstrations do not have the required diversity of states for the agent to learn successfully.



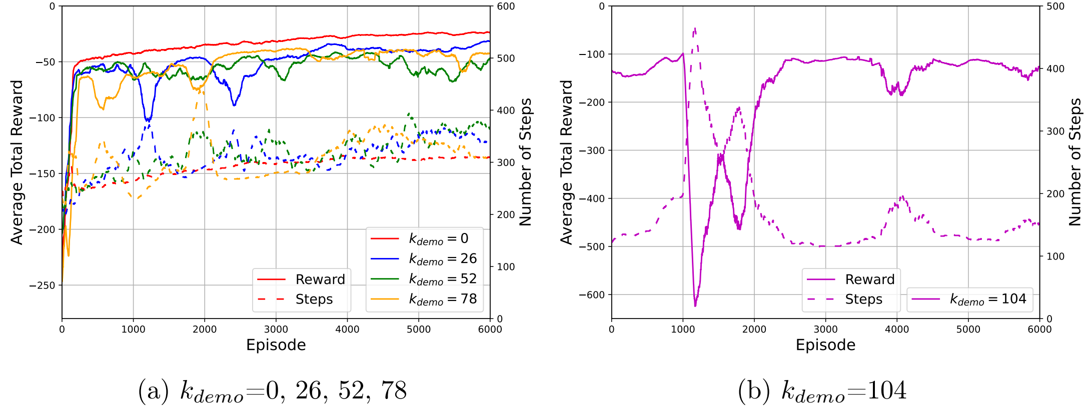
(a) k_{demo}=0, 26, 52, 78

(b) k_{demo}=104

Figure 5.6: Learning curves averaged over 8 runs for different values of k_{demo}. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 150 episodes for clarity.

5.4.4 Testing Agents

To select the agent to test for each value of k_{demo}, the runs with the highest average cumulative reward at the end of its training were used. In this case the average was calculated over the last 100 episodes. This was chosen instead of 500 as in Equation 4.20 since, as highlighted in Figure 5.6, the total reward varied substantially for some agents over the last 500 episodes and so 100 episodes gives a better estimate of performance after training. The best agents carried out 5000 test episodes without updating to assess their performance over a wide range of initial conditions. Figure 5.7 shows the learning curves and number of steps over training for the best performing runs. As with the mean learning curves, the agent trained without demonstrations has the most consistent learning curve across training.



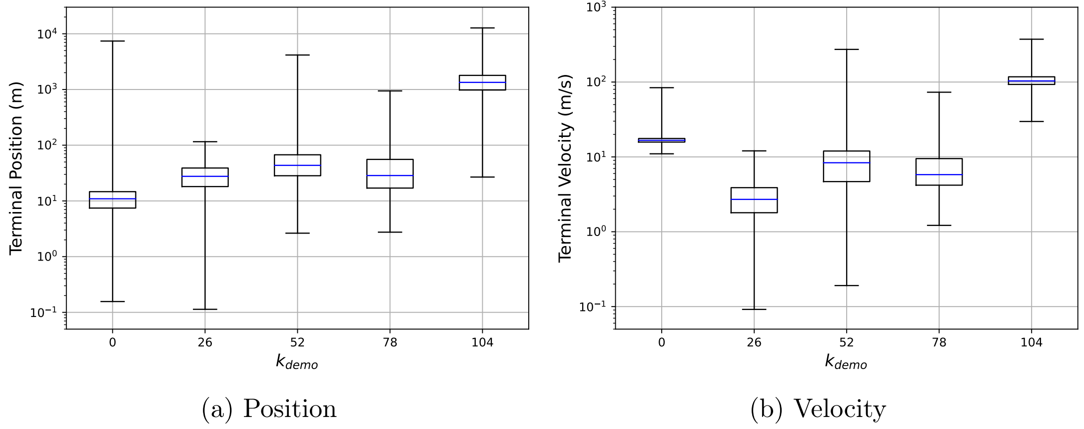
(a) k_{demo}=0, 26, 52, 78

(b) k_{demo}=104

Figure 5.7: Average reward and steps per episode over best performing training run for different values of k_{demo}. Uniformly filtered (average) over 150 episodes for clarity.

Figure 5.8 shows the distribution of terminal states for each of the trained agents over the 5000 test episodes. The terminal position is calculated as $\left\| \begin{bmatrix} r_x & r_y \end{bmatrix} \right\|$ and the terminal velocity is calculated as $\left\| \begin{bmatrix} v_x & v_y & v_z \end{bmatrix} \right\|$. Most of the agents have a large range of terminal states, with some terminal positions as high as $10km$ and terminal velocities over $100m/s$. This is from cases where the agent does not successfully land and instead the episode terminates after the maximum 500 timesteps. The agent trained without demonstrations shows the least variation in its terminal velocity, but its velocity is always over $10m/s$ as a result of the reward-hacking behaviour discussed in Chapter 4. Of the agents trained with demonstrations, the value of $k_{demo} = 26$ shows the best performance on average and in terms of its worst-case terminal state. This would suggest that it is still beneficial for the agent to train on mostly its own experiences, while a smaller amount of demonstration experiences is useful for preventing reward-hacking.

Table 5.4 compares the best performing agents trained with no demonstrations and with $k_{demo} = 26$. Although the mean velocity tangential to the surface is $4.063m/s$ for the agent trained without demonstrations, the minimum normal velocity in the z-direction is $8.562m/s$. The large value of the maximum terminal position is due to cases where the agent does not land. While the agent trained with demonstrations has a higher mean terminal position, its terminal velocity is consistently lower—especially in the z-direction. This shows that the use of demonstrations can mitigate the reward-hacking seen in this problem. Considering the fuel consumption, in the cases where



(a) Position (b) Velocity

Figure 5.8: Box plots of the terminal position and velocity of the trained agents over 5000 test episodes. Coloured line indicates median and whiskers indicate minimum and maximum values.

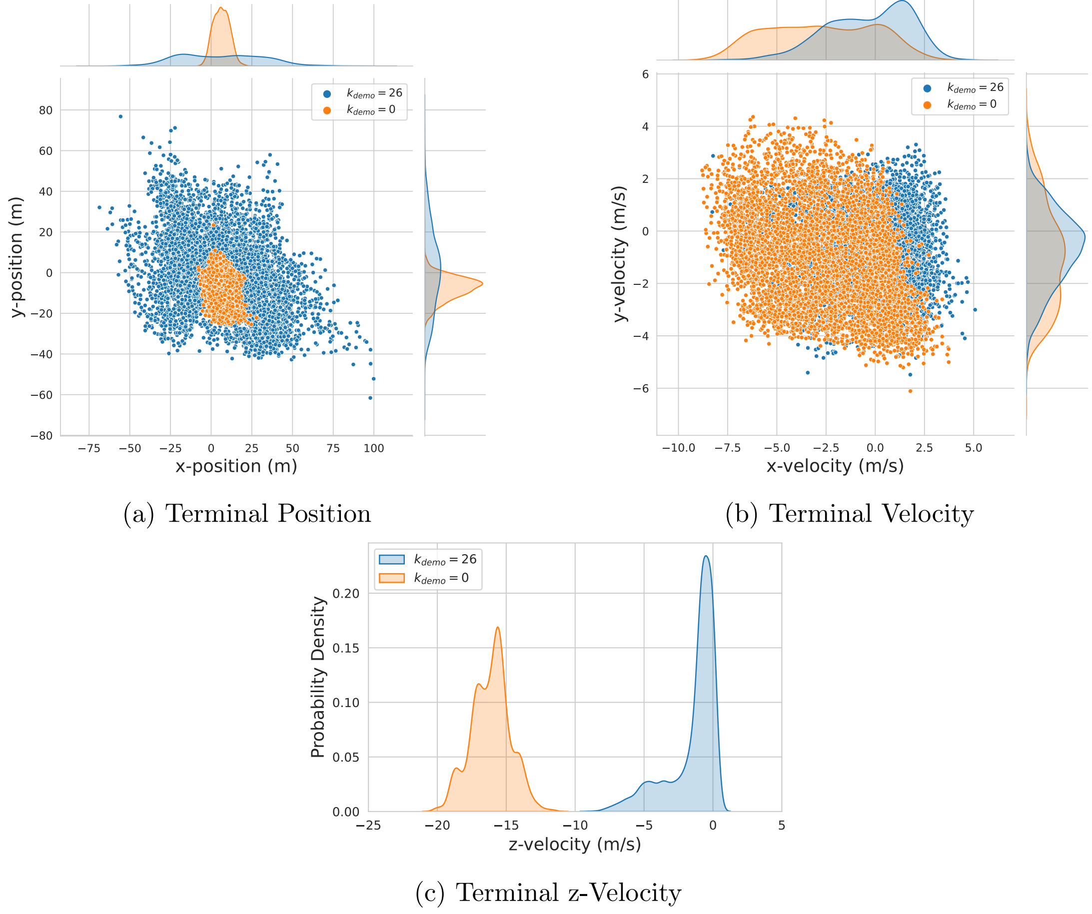
the agent trained without demonstrations does not land, it has a much higher fuel consumption as indicated by the maximum value. However, this agent has a lower mean fuel consumption. This is to be expected since the agent trained with demonstrations uses more fuel towards the end of the descent to achieve a softer landing.

Table 5.4: Comparison of test results from agents trained with and without demonstrations. Statistics shown for terminal state over 5000 test episodes.

		Mean	Min.	Max.	STD
$k_{demo} = 0$	xy-position (m)	20.425	0.156	7406.891	246.288
	xy-velocity (m/s)	4.063	0.073	73.315	2.83
	z-velocity (m/s)	16.119	8.562	41.112	1.569
	Fuel (kg)	300.0	254.8	563.6	19.2
$k_{demo} = 26$	xy-position (m)	29.4	0.113	115.603	15.161
	xy-velocity (m/s)	2.353	0.027	8.727	1.235
	z-velocity (m/s)	1.466	2.88×10^{-5}	8.717	1.738
	Fuel (kg)	338.8	275.0	433.3	32.0

Figure 5.9 compares the distribution of terminal states for the best agents graphically. These only include terminal states where the altitude $r_z \leq 0.01m$ to allow for better comparison. For the agent trained with demonstrations, all terminal states fulfil this requirement and for the agent trained without demonstrations, 8 episodes did not perform a landing. It is clear that the agent with demonstrations has a greater spread of

terminal positions in these cases and the terminal velocities in the tangential direction are distributed similarly for both agents. The main difference is in the z-velocity distributions, which is much closer to 0 for the agent trained with demonstrations. Although this agent achieves much softer landings than without demonstrations, its distribution of terminal positions is larger than desired. However, including demonstrations still results in much better performance in terms of achieving a soft landing.



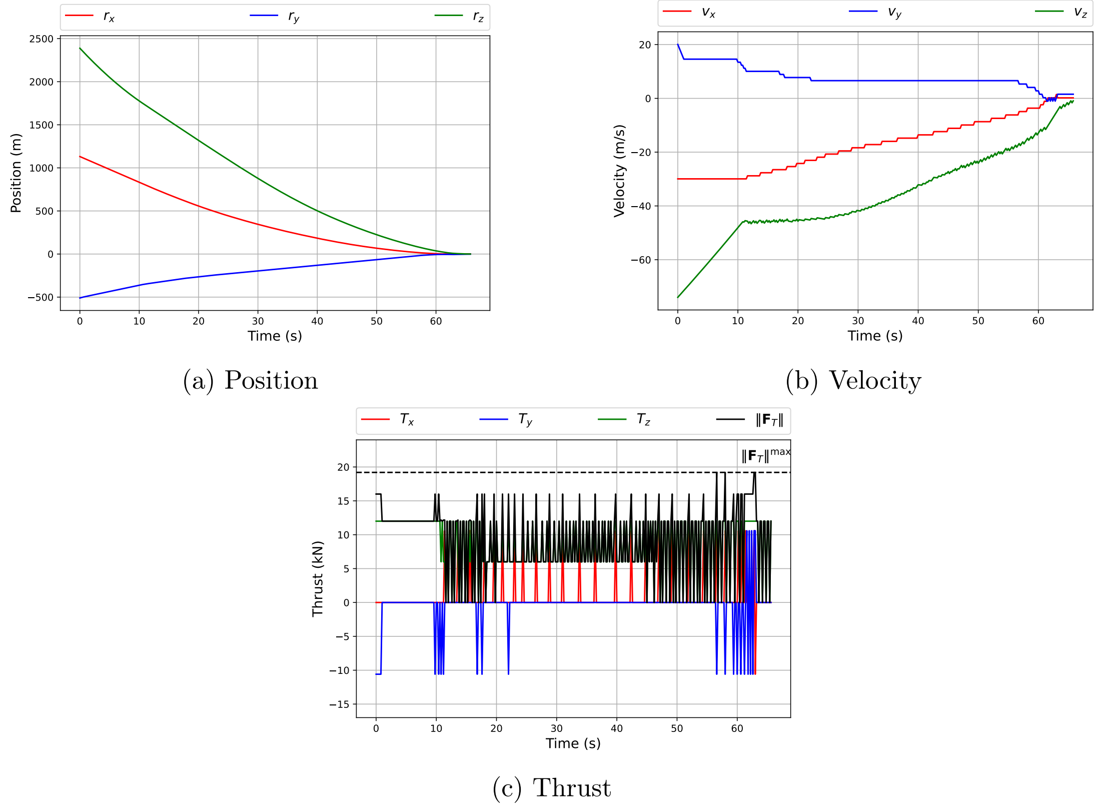
(a) Terminal Position (b) Terminal Velocity

(c) Terminal z-Velocity

Figure 5.9: Scatter and KDE plots showing distributions of terminal landing states for best agents trained with and without demonstrations, where $r_z \leq 0.01m$.

Figure 5.10 shows an example trajectory using the agent trained with $k_{demo} = 26$. This trajectory has an initial state of $\mathbf{r}_0 = \begin{bmatrix} 1.131 & -0.510 & 2.389 \end{bmatrix} km$ and $\mathbf{v}_0 = \begin{bmatrix} -29.94 & 20.06 & -73.98 \end{bmatrix} m/s$ with an initial mass of $m_0 = 1936kg$. Its terminal position is $2.0m$ from the desired landing site with a terminal velocity of $1.8m/s$ and the

total fuel consumption is $304.6kg$. Even without the shaped state representation, this trajectory clearly shows the effect of the shaped reward function that follows a certain velocity profile with gradually decreasing magnitude over the course of the trajectory.



(a) Position (b) Velocity

(c) Thrust

Figure 5.10: Trajectory of the agent trained with optimal control demonstrations.

5.5 Summary

This chapter presented a method for incorporating optimal control demonstrations into RL. The proposed method is applicable to off-policy RL algorithms that can learn from arbitrary policies. In this case, DQN was used as the RL algorithm, which has the additional requirement of demonstrations with discrete actions. Although the process of discretisation results in a loss of optimality and, in some cases, violations of constraints, the discretised trajectories are still suitable for use as demonstrations. The results of this method applied to the powered descent problem show that it is capable of mitigating

against reward-hacking behaviour from training without demonstrations.

As well as their use for demonstrations, trajectories obtained via optimal control approaches can inform the reward shaping. Appendix A shows some preliminary results from creating a shaped reward in this manner. In the results shown here, the only uncertainties considered are in the initial conditions. The following chapter shows how this method can be extended to dealing with further uncertainties in a way that brings RL into the realm of intelligent control.

Chapter 6

Online Updates for Continuous Learning

The RL approaches described in previous chapters all assume that the agent training occurs entirely offline prior to running in the environment. This is common practice when training RL agents—as is the case in many of the examples of related work shown previously. Although agents trained in this manner can learn near-optimal solutions to control problems, this would not be classed as Intelligent Control (IC) as defined in Chapter 3 since the agents do not have the capacity to update online. These agents can still be trained to be robust to uncertainties by incorporating uncertainty into their training. However, this does not exploit the adaptive and learning capabilities of IC systems that are designed to handle substantial uncertainties online.

The work presented in this chapter aims to extend the RL approaches described previously into the domain of IC by incorporating online updates. This is useful in situations where the environment may change or contain uncertainties that were unmodelled when the agent was trained offline. The proposed method of updating agents online uses a novel update mechanism referred to as Extreme Q-Learning Machine (EQLM) [31]. This is based on the paradigm of Extreme Learning Machines (ELMs) described in Chapter 2 that is an alternative to gradient-based updates of NNs. As will be shown, EQLM can change weights more substantially in online updates than gradient-based methods. The approach introduced here is similar to fine-tuning, which is often used to

improve task-specific performance of LLMs [149]. This is where, following initial training on a large dataset, a DNN performs smaller updates on its weights to "fine-tune" its performance. Similarly, the proposed method of online updates uses observed data to "fine-tune" the agent's performance.

One of the barriers to implementing IC onboard spacecraft historically has been hardware limitations. Modern deep learning systems typically use very computationally intensive architectures that require large datasets for training. While these systems can be trained offline, online updates for spacecraft place stricter limits on available computational power. Advances in computing edge hardware have now made it more feasible to run ML models onboard spacecraft. In addition to introducing a method of updating online, this chapter demonstrates the possibility of running this onboard spacecraft using suitable hardware. The specific hardware used is the Jetson Nano Developer Kit, which is designed to test edge AI applications. The Nvidia Jetson architecture has flight heritage onboard the La Jument mission, which was developed by Lockheed Martin and the University of Southern California [150]. More recently, a cubesat built by Aethero was launched with a radiation hardened Nvidia Jetson Orin NX GPU [151].

This chapter introduces a method for applying RL with online updates. The main contributions of this work are:

- An alternative update mechanism for Q-networks based on ELM that does not use gradients.
- Application of this update mechanism to online updates, which allows an agent to continue learning following its initial training and adapt to changes in the environment online.
- Results of online updates running onboard flight suitable hardware to demonstrate its potential for use onboard spacecraft.

The remainder of this chapter is organised as follows. Section 6.1 presents literature related to RL with online updates, use of ML on edge hardware, and applications of

ELM in RL. Section 6.2 introduces the method for applying online updates and Section 6.3 shows results of this method applied to spacecraft powered descent.

6.1 Relevant Literature

The general approach of training a system offline and deploying it online without further updates is not unique to RL. Indeed, many other ML methods are often used in such a manner. The alternative approach of having a ML system continue to learn after initial training is commonly referred to as "lifelong learning" or "continual learning" [152]. Within this paradigm, many approaches have been proposed with the aim of alleviating the issue of the system "forgetting" information learned earlier in training [153–156]. This is commonly referred to as "catastrophic forgetting". In the context of RL, including online updates has been shown to improve performance even where the environment does not change [157, 158]. One example online updating approach called "policy finetuning" was introduced with the aim of unifying the main ideas of both offline and online RL [159]. In this approach, an agent starts with a reference policy that performs reasonably well and can further improve the policy via online interactions. This is similar to the approach of online updates described here, but with a different method for applying online updates. A common problem with online RL is that the best performing policy when pretrained offline may not be the best performing for online updates. In [160], the authors aim to remedy this by enforcing conservative value estimates in offline pretraining. This idea is further developed in "Cal-QL", where the conservative offline value estimates are calibrated to that of a reference policy [161]. The results shown in this chapter also illustrate the discrepancy in performance between offline and online updates. However, the proposed approach does not change the method of offline pretraining, as will be shown.

One of the key challenges in deploying RL for real-world problems is that of sim-to-real. This is where an agent is trained offline in a simulated environment, but must operate online in the real environment that might be imperfectly modelled. Approaches to solving this problem typically also involve some form of online updates [162]. Although the results presented here do not directly address the sim-to-real problem, the

idea of testing an agent in a different simulated environment to which it is trained has similar issues to sim-to-real.

Another issue with deep-learning based RL methods is the lack of stability guarantees, which is especially important in systems that update online. There are efforts to mitigate this by incorporating stability theories from conventional control methods into RL controllers. A relevant example of this investigates the stability of a spacecraft attitude control system [163]. The proposed method analyses the learned value function to ensure during and after training that it satisfies criteria to achieve a stable system. Agent training is performed via a modified PPO algorithm which includes a critic loss for penalising Hamilton-Jacobi-Bellman equation violations.

Another paradigm in ML that allows for online learning is meta-learning, where a ML algorithm aims to quickly learn to solve a new task by training on related tasks [164]. This can also be viewed as a process of learning a model representation that allows for fast online updates, similarly to the online learning discussed above [165]. In meta-RL, the agent's learning procedure can be parameterised as a Recurrent Neural Network (RNN) such that its policy depends on the history of transitions it has viewed [166]. This results in fast updates of the policy in response to changes in the MDP with which the agent interacts, which is necessary for certain nonstationary environments [167]. Meta-RL has been applied to many different spacecraft control problems, including exoatmospheric interception [168], interplanetary and cislunar guidance [169–171], lunar landing [172], and near asteroid GNC [173–175]. In [25], the authors compare the performance of RL and meta-RL in planetary powered descent problems. They find that meta-RL that has an adaptive policy performs better than fixed policy RL not only in off-nominal conditions, but also under nominal operating conditions.

Some methods of updating online also require a model of the environment that updates in response to observations. In conventional control schemes, Model Predictive Control (MPC) uses a model of the system dynamics to select actions, which can also be updated [176]. These methods are also common within IC, such as with fuzzy models [177] and NN models [178]. Model-based RL incorporates a system model to allow the agent to plan its future actions [179]. An environment model can also be used for

additional online updates aside from direct observations. This has been demonstrated for Meta-RL in [180], where the agent has a model of the distribution of problems on which it is trained. Reinforcement Learning with Context Detection (RL-CD) is a RL algorithm that was introduced specifically to handle nonstationary environments using a set of "partial models" of the environment [181]. The agent's policy in RL-CD depends on its current partial model of the environment, which is derived based on observed transitions. It detects changes in "context", where the environment dynamics switches between discrete contexts of stationary dynamics that can be learned. In the approach of online updates introduced here, it is assumed the agent has access to a model of the environment for updating, as well as updating based on new observations. As described above, there are many suitable approaches for deriving such a model, but this will not be addressed here.

Section 2.3 discussed ELM as another approach to update SLFNs. ELM has been employed in a wide range of ML applications [182], however it has seen limited use within RL. One example application uses tabular RL to provide training data for a more efficient representation of the action-value function using ELM [183]. Another method uses ELM theory to derive an analytical solution for weight updates based on the loss function of gradient-based updates [184]. ELM has also been used as the update mechanism for Least Squares Temporal Difference (LSTD), which is a general approach to solving RL problems via temporal difference learning [185, 186]. In previous work, EQLM was proposed as an alternative approach for updating Q-networks [31]. Using ELM in this manner does not necessarily give the same benefits of increased speed as in other applications due to the iterative nature of updates in policy iteration. However, it can still show some advantages to gradient descent for performing updates online as will be demonstrated.

New advances in edge hardware for AI have increased the possibility for spacecraft onboard intelligence. Several manufacturers now produce high performance GPUs that are optimised for running AI algorithms while remaining computationally and physically lightweight. For example, companies such as NVIDIA [187] and Intel [188] have invested substantial resources in developing edge AI capabilities. Due to the availability

of this hardware, applications of onboard intelligence are now growing, with most being related to image processing. These are particularly beneficial for high-dimensional data that may be difficult to downlink, such as synthetic aperture radar images [189] and hyperspectral images [190]. The ESA mission Φ-sat-1 was intended as a demonstration mission for onboard AI capabilities that flew in 2020 [191]. It was successful in demonstrating autonomous classifications of images with and without clouds via an onboard DNN. Φ-sat-1 featured an Intel Movidius Myriad 2 processor, which has also been used for further demonstrations of potential onboard applications. One such example uses flight tested hardware to create segmentation maps of floods with the goal of spacecraft being able to downlink these maps in real time to enable quicker disaster response [192]. These results demonstrate the feasibility of using onboard AI for processing high-dimensional data, which shows promise for the potential of onboard IC.

6.2 DQN with Online EQLM Updates

Figure 6.1 shows a schematic of the proposed approach of online updates, which can be divided into three main parts. The first two parts occur offline for initialising the agent and the final part occurs online. First an agent is trained using conventional gradient descent methods to determine initial weights for the Q-network. The weights learned from this part are then used to initialise offline the matrix used for EQLM updates. Finally, the agent updates its output weights online.

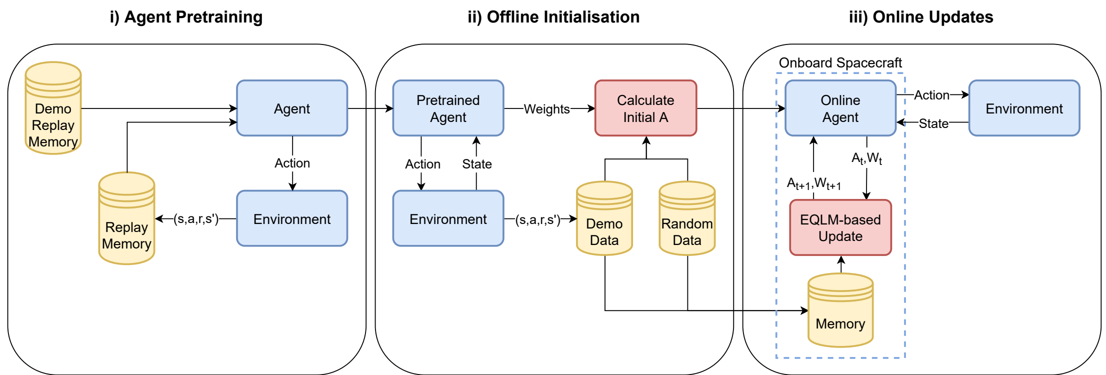


Figure 6.1: Schematic illustration of the three main phases in the proposed method of online updates.

6.2.1 Offline Agent Pretraining

The initial offline agent pretraining uses the approach described in Chapter 5 to train an agent using optimal control demonstrations. Following pretraining, all the Q-network parameters are fixed except for the output weights which will be updated online. This initial training using gradient descent allows EQLM updates to be performed on just the output layer and instead of random initial weights, the other network parameters are tuned to give a useful representation to the final layer. The output of this step is the pretrained agent's parameters, i.e. the NN weights.

6.2.2 Offline EQLM Initialisation

Sequential ELM-based updates as described in Section 2.3 define a matrix $A_t^\dagger$ that is used in updating the output weights. Prior to applying EQLM online, it is first necessary to initialise this matrix and the values of the output weights. The initialisation uses a sample of k experiences using actions from the pretrained agent, optimal demonstration actions, and a random agent. The experiences from a random agent are included to improve the diversity of states used in the initialisation and subsequent updates.

The input to the Q-network, denoted $\mathbf{x}$, is the environment state. The output of the ith hidden layer, denoted $\mathbf{h}_i$, is:

$$\mathbf{h}_i = (f_i \circ f_{i-1} \circ \cdots \circ f_1)(\mathbf{x}) \tag{6.1}$$

where f_i is a nonlinear function of the previous layer's output resulting in each output being a composite function of all previous layers. For the initial minibatch of k inputs $(s_1, \ldots, s_k)$, the matrix $\mathbf{H}$ is defined as the output of the final hidden layer (denoted $\mathbf{h}$ for simplicity) for each of the inputs as shown:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}^\top|_{\mathbf{x}=s_1} \\ \vdots \\ \mathbf{h}^\top|_{\mathbf{x}=s_k} \end{bmatrix}_{k\times\tilde{N}} \tag{6.2}$$

where $\tilde{N}$ denotes the number of nodes in the final hidden layer. Each row of $\mathbf{H}$ contains

the final hidden layer output $\mathbf{h}$ for each of the inputs $(s_1, \ldots, s_k)$. The target matrix $\mathbf{T}$ is then defined as the corresponding target action-values:

$$\mathbf{T} = \begin{bmatrix} \mathbf{y}_1^\top \\ \vdots \\ \mathbf{y}_k^\top \end{bmatrix}_{k \times m} . \tag{6.3}$$

Targets $\mathbf{y}_j$ for each experience are equal to the network output $Q_\theta(s_j, a)$ for all $a \neq a_j$. For the relevant action a_j in each experience, the target action-value is

$$y_{j,a_j} = r_j + \gamma \max_a Q_{\theta^-}(s'_j, a), \tag{6.4}$$

which comes from the TD-error defined in Equation 2.10. As with gradient based Q-network updates, $\boldsymbol{\theta}^-$ denotes the target network weights. During online updates, the target network weights remain constant to avoid potentially large divergences in target action-values.

Following pretraining only the output weights $\boldsymbol{\beta}$ are updated as shown in Figure 6.2. In this example, the network has three hidden layers each with 4 hidden nodes. Only the output weights connecting the final hidden layer to the output are updated online.

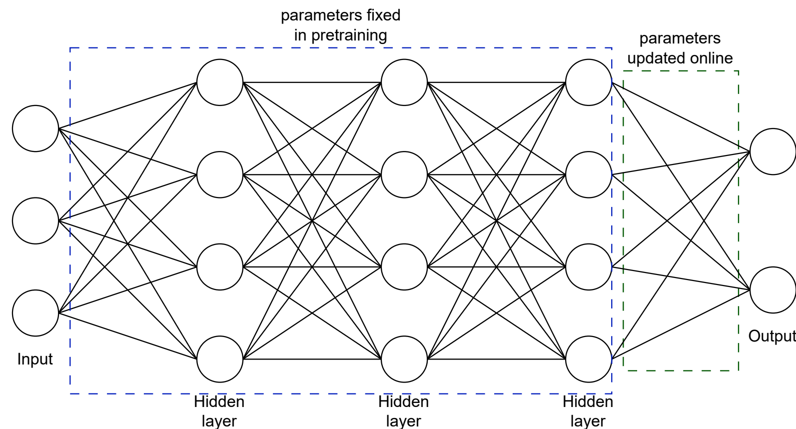


Figure 6.2: Diagram of a deep feedforward network indicating the relevant parameters that are fixed following pretraining and output weights that are updated online.

In the case where EQLM is used without pretraining, network parameters are randomly assigned and the output weights are initialised via Equations 2.31 and 2.32. However, following pretraining the output weights of the Q-network have already been trained. Therefore, the matrix $A^{\dagger}$ can instead be initialised based on the pretrained weights by rearranging Equation 2.31 as shown:

$$A^{\dagger}_{t=0} = \boldsymbol{\beta}_{t=0} \left[\mathbf{H}^{\top}\mathbf{T}\right]^{\dagger}, \tag{6.5}$$

where $\beta_{t=0}$ are the optimised output weights.

$\mathbf{H}$ and $\mathbf{T}$ are both calculated for a large batch of experiences to perform this step. Similarly to the pretraining, these experiences are sampled from two replay memories: one obtained using the pretrained agent's policy and one using a random policy. The total minibatch size for initialisation is then $k = k_{agent} + k_{demo} + k_{random}$ where k_{agent}, k_{demo}, and k_{random} are the number of experiences sampled from the agent's replay memory, optimal demonstrations, and random replay memory, respectively.

6.2.3 Online EQLM Updates

With the weights and matrix for updating weights initialised, the agent can update its weights online. As with the initialisation part, the online updates use data from agent demonstrations, optimal demonstrations, and random actions. In addition to these replay memories, the experiences observed online by the agent also make up the data used to update. These experiences are all fed to the agent in minibatches, whose size is again denoted k. If the agent has already updated from N experiences and samples a new batch of k experiences $\left(s_j, a_j, r_j, s'_j\right)$, $j = N+1, \ldots, N+1+k$ for updating, the new incremental matrices $\mathbf{H}_{IC}$ and $\mathbf{T}_{IC}$ can be defined as follows:

$$\mathbf{H}_{IC} = \begin{bmatrix} \mathbf{h}^{\top}|_{\mathbf{x}=s_{N+1}} \\ \vdots \\ \mathbf{h}^{\top}|_{\mathbf{x}=s_{N+1+k}} \end{bmatrix}_{k\times\tilde{N}}, \tag{6.6}$$

$$\mathbf{T}_{IC} = \begin{bmatrix} \mathbf{y}_{N+1}^{\top} \\ \vdots \\ \mathbf{y}_{N+1+k}^{\top} \end{bmatrix}_{k \times m} . \tag{6.7}$$

Using these matrices, EQLM updates weights according to the update rules defined by Equations 2.45 and 2.46. These updates occur after a certain number of timesteps, n_{step} in the environment. At each update, the agent's previous n_{step} experiences are all used to update. This gives a total minibatch size for online updates of $k = n_{step} + k_{agent/demo} + k_{random}$, where $k_{agent/demo}$ denotes the number of samples taken from the combined agent demonstrations and optimal control demonstrations. These updates combine "new" experiences in the environment and "old" experiences from the agent's pretraining, which is a potential issue if the environment changes online. However, as will be shown, even when using old experiences the agent's policy can still adapt based on the new ones. As an additional step prior to deployment of the agent and following initialisation, the agent carries out further training episodes with online updates. These update the output weights β and matrix $A^{\dagger}$ before they are used on the hardware with the goal of improving the online agent's performance.

Algorithm 3 shows the general procedure for updating a pretrained agent via EQLM updates. This algorithm takes as input the pretrained network parameters and matrix for updating as well as replay memories with experiences from the agent, optimal control demonstrations, and random actions. For N_{ep} episodes the agent interacts with the environment and updates every n_{step} timesteps according to the update rules defined above. This is the process used for both the offline and online EQLM updates.

6.3 Results of DQN with Online Updates Applied to Powered Descent

This section presents the results from each stage of training the online updating agents. First, multiple agents were trained offline with optimal demonstrations. Following pretraining, each of these agents ran multiple offline initialisations and training episodes with online EQLM updates. Results are then shown for the best performing of these

Algorithm 3 Agent training with EQLM updates

Inputs: pretrained parameters $\boldsymbol{\theta}$, initialised matrix $A^{\dagger}_{t=0}$, combined agent and optimal control demonstration replay memory $\mathcal{D}_{agent/demo}$, random replay memory $\mathcal{D}_{random}$, number of training episodes N_{ep}, agent hyperparameters

1: set target network parameters $\boldsymbol{\theta}^{-} \leftarrow \boldsymbol{\theta}$
2: **for** $ep = 1$ to N_{ep} **do**
3: initialise state $s_t \leftarrow s_0$
4: $stepcount \leftarrow 0$
5: **while** state s_t is non-terminal **do**
6: select action a_t according to policy π
7: execute action a_t and observe r_{t+1}, s_{t+1}
8: update online memory $\mathcal{D}$ with $(s_t, a_t, r_{t+1}, s_{t+1})$
9: $stepcount \leftarrow stepcount + 1$
10: **if** $stepcount = n_{step}$ **then**
11: select n_{step} most recent experiences (s_j, a_j, r_j, s'_j) from $\mathcal{D}$
12: select random minibatch of $k_{agent/demo}$ experiences (s_j, a_j, r_j, s'_j) from $\mathcal{D}_{agent/demo}$
13: select random minibatch of k_{random} experiences (s_j, a_j, r_j, s'_j) from $\mathcal{D}_{random}$
14: calculate $\mathbf{H}_{IC}$ and $\mathbf{T}_{IC}$ according to Equations 6.6 and 6.7
15: update output weights $\boldsymbol{\beta}$ and matrix $A^{\dagger}_{t}$ according to Equations 2.45 and 2.46
16: $stepcount \leftarrow 0$
17: **end if**
18: **end while**
19: **end for**

agents with online updates in response to changes in the environment. Finally, results from running online updates on edge hardware demonstrate the potential for this process to run onboard spacecraft.

6.3.1 Offline Pretraining

The environment dynamics in these experiments were as defined in Chapter 5, but with the disturbance force $\mathbf{F}_D$ included (Equation 4.2). The disturbance force is randomly sampled from a normal distribution with mean $0N$ and standard deviation $100N$ in each direction. This magnitude of disturbance is based on the values used in other works [23]. Samples are taken at $1s$ intervals and linearly interpolated for timesteps between samples. The hyperparameters used for both pretraining and online updates

are based on those used in previous work [28], which were found to give suitable levels of performance without the need for further tuning. This is also the case for the initial training time, which was 10,000 episodes. Table 6.1 shows the hyperparameters that were used in all pretraining runs.

Table 6.1: Hyperparameters for agent pretraining.

Parameter	Value
$\tilde{N}_1$	300
$\tilde{N}_2$	200
$\tilde{N}_3$	300
α	1×10^{-5}
C	65
ϵ_0	0.8
N_ϵ	4000
γ	0.926
k	78
k_{demo}	26

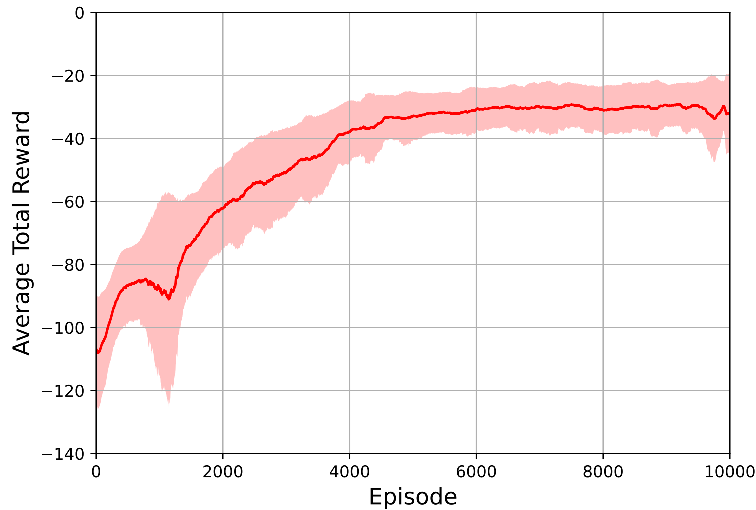


Figure 6.3: Learning curves averaged over 8 pretraining runs. Shaded area indicates +/- one standard deviation. Uniformly filtered (average) over 150 episodes for clarity.

8 separate agents were trained with different random seeds. Figure 6.3 shows the average learning curve from all of the pretrained agents. As with the learning curves shown in previous chapters, the cumulative reward varies across each run as indicated

by the standard deviation. The average curve decreases slightly at around 1000 episodes along with an increase in the variance, but then increases steadily until around 5000 episodes. The final half of the training time sees only a slight increase in the average reward, but with an increase in variance towards the end of training.

Figure 6.4 shows the average total reward and number of steps per episode for the best performing pretrained agent in terms of the mean reward over the final 100 episodes. Both curves show similar trends of gradually increasing over the initial training episodes and then levelling off. Although there are fluctuations in the average total reward, for this learning curve it still slightly increases over the latter half of the training episodes. As will be shown in the following section, this set of pretrained weights resulted in the best performing agent following EQLM updates.

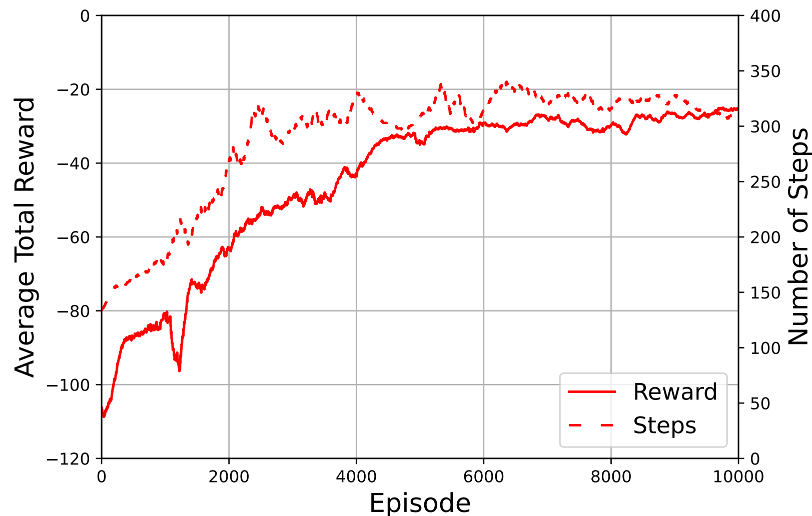


Figure 6.4: Average reward and steps per episode over an offline pretraining run of Q-learning. Uniformly filtered (average) over 150 episodes for clarity.

6.3.2 Testing Initialisation Random Seeds

In addition to the set of pretrained weights, both the initialisation of the matrix $A^{\dagger}_{t=0}$ and the EQLM updates involve a stochastic process that affects the final performance of the agent. Therefore, to obtain the best performing online updating agent requires testing a number of agents using the same set of pretrained weights but with different

random seeds for initialisation and EQLM updates. The initial minibatch size k used to initialise $A^{\dagger}_{t=0}$ was fixed at 3000 for all experiments with 1000 experiences sampled from the agent demonstrations, 1000 from random actions, and 1000 from the optimal control demonstrations. The range of initial conditions used for training and testing with EQLM updates were the same as in pretraining.

Prior to the agent testing, each of the agents ran with EQLM updates for 500 episodes. Similarly to the initialisation described above, the EQLM updates use a combination of data from new observations, agent memory, optimal control demonstrations, and random memory. The agent demonstrations are combined with the optimal demonstrations and equal numbers of samples are taken from this memory and the random memory, such that $k_{agent/demo} = k_{random} = 0.5(k - n_{step})$. Based on the results of previous work that studied the effects of these parameters [28], these values are selected as $k = 40$ and $n_{step} = 4$, resulting in demonstration data minibatch sizes of $k_{agent/demo} = k_{random} = 18$.

Agents were evaluated over 500 test episodes based on the mean terminal position and velocity of all episodes. Each of the 8 pretrained agents ran 8 initialisations of $A^{\dagger}_{t=0}$ which were each trained with EQLM updates with 8 different random seeds. This gives a total of $8 \times 8 \times 8 = 512$ trained agents. Table 6.2 shows a summary of the performance of the pareto-optimal agents in terms of their terminal position and velocity. The indicated "Pre. Run", "Init. Run", and "Seed" numbers are arbitrary to refer to the respective pretraining run, initialisation, and EQLM update seeds.

Table 6.2: Summary statistics of terminal position and velocity for the pareto optimal pretrained weights, initialisations, and EQLM update seeds over 500 test episodes.

Pre.	Init.	Seed	xy-position (m)		xy-velocity (m/s)		z-velocity (m/s)	
Run	Run		Mean	Max.	Mean	Max.	Mean	Max.
0	1	2	22.854	115.086	1.992	5.736	1.641	4.361
0	1	3	17.001	81.445	2.735	7.864	2.484	6.935
0	1	4	18.101	83.713	2.049	7.791	1.963	18.184
0	1	5	17.760	120.774	1.993	14.138	2.919	19.226
0	6	5	13.088	63.668	2.138	12.450	6.082	21.504
6	4	6	42.751	380.856	2.117	58.829	1.034	79.114

Of the 6 pareto-optimal agents, five of them use the pretrained weights of run 0 and four of these are from initialisation run 1. The one other pareto-optimal set of pretrained weights from run 6 shows worse performance in terms of the maximum values of terminal state than those of run 0. The pareto-optimal agent with pretrained weights of run 0 and initialisation run 6 performs better in terms of terminal position, but notably worse in terms of the terminal z-velocity than the other agents. For these reasons, the pretrained weights from run 0 and offline initialisation run 1 were used for further training.

With the pretrained weights and $A^{\dagger}_{t=0}$ initialisation defined, more agents were trained with EQLM updates over 32 different random seeds. This was done over this many seeds to find the best performing agent with this configuration. For these runs, the aim was to optimise for the worst-case performance of the agent and so the maximum terminal position and velocity across testing episodes was considered. Table 6.3 shows the summary statistics for the performance of the three pareto-optimal agents. As above, the "Seed" is arbitrary to refer to the seed number of the respective agent.

Table 6.3: Summary statistics of terminal position and velocity for the pareto optimal EQLM update seeds over 500 test episodes.

Seed	xy-position (m)		xy-velocity (m/s)		z-velocity (m/s)	
	Mean	Max.	Mean	Max.	Mean	Max.
1	20.834	100.412	2.441	6.414	1.986	4.742
3	23.100	94.679	2.865	11.803	2.232	6.167
25	19.068	96.925	1.746	4.873	1.315	10.232

In terms of mean and maximum terminal position, all of the pareto-optimal agents perform very similarly. Seed 25 gives the smallest mean velocities, but also the largest maximum z-velocity. Since the mean terminal z-velocity of this agent is substantially lower than the others—34% smaller than seed 1 and 41% smaller than seed 3—this maximum z-velocity is likely to be an outlier in the agent's distribution of terminal states. Therefore, this agent was chosen as the best performing agent to use for further testing.

6.3.3 Online Updates

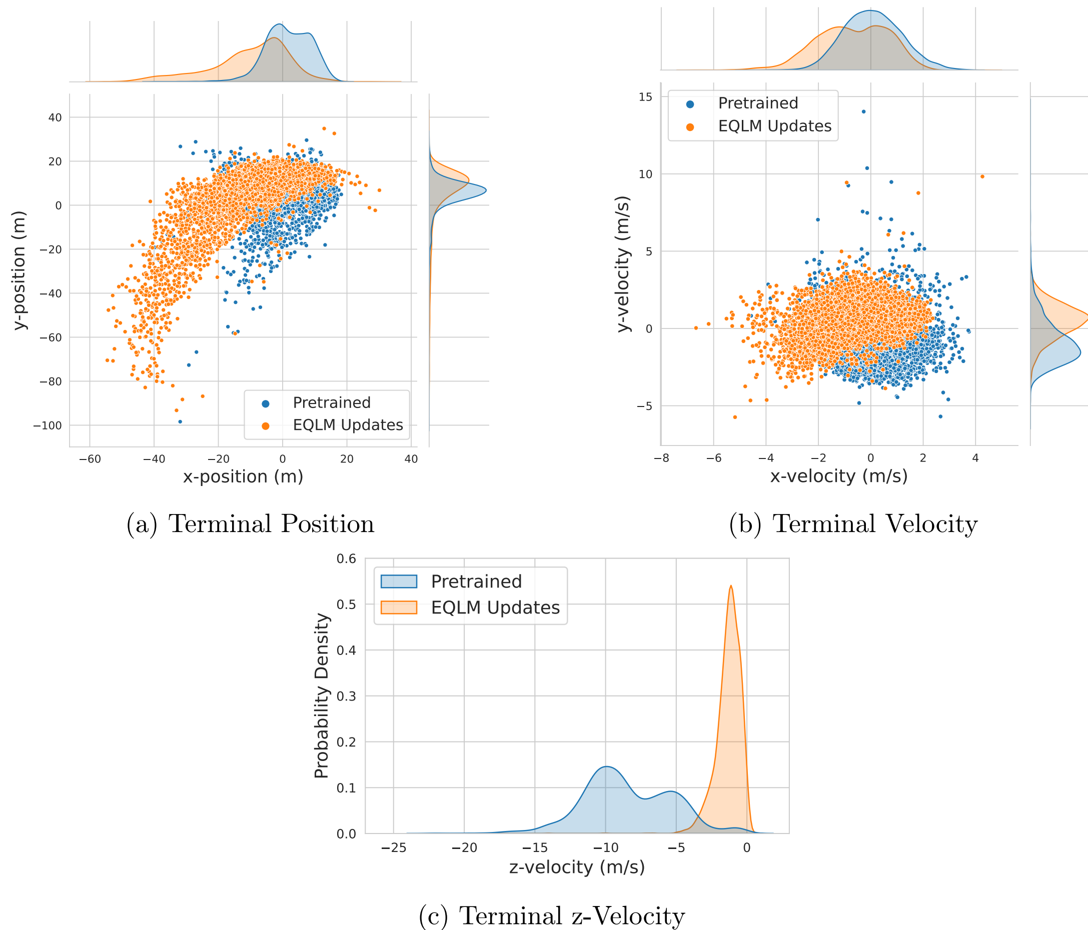
The following results use the best performing agent with EQLM updates as described above. When running test episodes, this agent also performs online updates using the same parameters as before of $k = 40$ and $n_{step} = 4$. Table 6.4 compares the performance of the pretrained agent and the best performing agent with EQLM updates over 5000 test episodes. In terms of the terminal position and fuel consumption, the pretrained agent with fixed weights performs better on average than the agent with EQLM updates. However, the pretrained agent performs worse in terms of the terminal velocity. This is particularly the case for the terminal z-velocity, which is $8.443 m/s$ on average for the pretrained agent compared to $1.317 m/s$ in the case of the EQLM agent. This is likely the reason for the larger fuel consumption of the EQLM agent, since it uses more fuel to achieve a softer landing than the pretrained agent.

Table 6.4: Comparison of test results from the pretrained agent and the best performing agent with EQLM updates. Statistics shown for terminal state over 5000 test episodes.

		Mean	Min.	Max.	STD
Pretrained	xy-position (m)	10.198	0.028	103.439	5.795
	xy-velocity (m/s)	1.824	0.055	14.032	0.887
	z-velocity (m/s)	8.443	6.48×10^{-3}	22.355	3.155
	Fuel (kg)	317.4	263.2	440.3	21.8
EQLM Updates	xy-position (m)	18.840	1.352	98.932	12.672
	xy-velocity (m/s)	1.761	0.031	10.713	0.911
	z-velocity (m/s)	1.317	1.16×10^{-3}	18.389	0.932
	Fuel (kg)	393	324.1	530.6	32.8

Figure 6.5 shows plots of the distribution of terminal states for both agents in the test episodes. The scatter of terminal positions shows that both agents have a long tail of points in the negative x- and y-directions, but more so for the agent with EQLM updates. In terms of x- and y-velocities both agents have a similar distribution that is clustered near the desired velocity of 0. Where their distributions show the most substantial variation is in the terminal z-velocity. Although both agents have similar maximum z-velocities as indicated in Table 6.4, the pretrained agent has a much higher variance in terminal z-velocity as its values are more spread over the range. On the

other hand, the distribution of the agent with EQLM updates has the highest density much closer to 0. This shows that EQLM updates can improve the overall landing performance of an agent that was initially trained with only gradient-based updates.



(a) Terminal Position

(b) Terminal Velocity

(c) Terminal z-Velocity

Figure 6.5: Scatter and KDE plots showing distributions of terminal landing states for the best agents before and after EQLM updates.

Although the EQLM agent improves on the performance of the pretrained agent, the overall velocity magnitude is still larger than desired. This is partly due to the limitation of the discrete action space, which lacks the finer control required to slow the lander to zero velocity. One approach to remedy this would be to use the optimal control approach presented in Chapter 5 with continuous actions at the end of the descent to achieve a soft landing. This combines the benefits of using the agent to deal

with uncertainties over most of the trajectory with the improved landing performance of optimal control.

The following results show how the agent's performance can be improved using optimal trajectories for the final part of the descent. Instead of terminating the episode at an altitude of $r_z = 0$, the agent's terminal state is set at $r_z = 10m$. The optimal control procedure described in Chapter 5 then controls the agent for the final $10m$ of the descent. $10m$ was chosen as a suitable altitude such that the disturbance forces do not substantially affect the optimal trajectory, but also where the optimisation can converge to a soft landing in most cases. Although the optimal trajectories could also be used to achieve terminal positions nearer the desired landing location, the main goal of these trajectories is to reduce the terminal velocity. Therefore, the target terminal position for the optimisation in the x- and y-directions is also set as the terminal position of the agent in these directions. Starting from an initial position of $\begin{bmatrix} 0 & 0 & r_z(t=T)+10 \end{bmatrix} m$ and an initial velocity of $\mathbf{v}_{t=T}$, the optimisation finds a trajectory that terminates at $\mathbf{r} = \mathbf{v} = 0$ while keeping all the relevant constraints described in Chapter 5.

Table 6.5 compares the terminal states of the agent with EQLM updates and a $10m$ optimal descent to those of the full optimal trajectories. The results shown for the EQLM updates used the same trajectories as shown in Table 6.4 with the terminal state adjusted as described above for the optimal descent. In the case of the full optimal trajectories, the initial conditions were the 33 initial states described in Chapter 5 but with disturbances included. Each of the initial states ran the corresponding optimal trajectories 100 times with different random disturbances.

Since the optimal descent does not adjust the agent's terminal xy-position, the distribution of this value is similar for the agent with optimal descent to that shown in Table 6.4. However, the mean terminal velocities are substantially reduced by the optimal $10m$ descent. In some cases, the agent's terminal velocity was still too large for the optimisation to converge to a soft landing, which is the reason for the relatively large maximum velocity magnitudes. These cases were rare though as indicated by the relatively low standard deviation of terminal velocities for this agent.

Table 6.5: Comparison of test results from the agent with EQLM updates + 10m optimal descent and the full trajectory optimisation with disturbances. Statistics shown for terminal state and fuel consumption over 5000 test episodes for the EQLM agent and 100 test episodes for each of the 33 initial states of the optimal trajectories.

		Mean	Min.	Max.	STD
EQLM Updates + 10m Descent	xy-position (m)	18.789	1.379	99.065	12.733
	xy-velocity (m/s)	0.153	0.067	7.018	0.108
	z-velocity (m/s)	0.103	1.61×10^{-4}	17.063	0.383
	Fuel (kg)	404.2	334.7	590.1	33.7
Full Optimal Trajectories	xy-position (m)	11.733	0.171	38.487	6.516
	xy-velocity (m/s)	1.601	0.010	14.509	2.572
	z-velocity (m/s)	1.807	3.80×10^{-4}	13.222	2.373
	Fuel (kg)	258.6	189.9	356.7	33.9

The optimal control trajectories also achieved a better distribution of terminal positions than the agent with optimal 10m descent. However, in terms of terminal velocity, the full optimal trajectories performed worse on average than the agent with optimal 10m descent and also show a higher standard deviation. This shows that without some feedback mechanism, the optimal trajectories could not achieve a soft landing in the presence of environmental disturbances. Considering the fuel consumption, the fully optimal trajectories with continuous actions show substantially lower fuel consumption than the agent. In addition, including the optimal 10m descent results in a slightly higher average fuel consumption for the agent compared to without the optimal descent. This indicates a necessary trade-off between how robust the agent is to disturbances and the optimality with respect to minimising fuel.

At this point it is necessary to revisit the original goal of the powered descent problem as defined in Chapter 4. With the optimal $10m$ descent, the agent achieved the desired maximum velocity magnitude of $2m/s$ in 99.9% of the testing episodes. On the other hand, in terms of terminal position the agent only achieved a maximum value of less than $5m$ in 1.48% of the testing episodes. As shown in the results over multiple random seeds, there was a performance trade-off between the terminal position and velocity in which velocity was prioritised. In addition, the optimal $10m$ trajectory did not also optimise for the terminal position, which could allow further improvements.

Considering the minimum glideslope, this goal was achieved in 84.58% of the testing episodes. Although the agent does not perform as well in the goal of a pinpoint landing, by combining the agent's policy with a short optimal trajectory it can effectively realise the soft landing requirement in most cases.

6.3.4 Online Updates with a Changing Environment

The previous results show the case where the environmental uncertainties, in the form of the disturbance forces, have the same distribution in testing as in training. In reality, this may not be the case. When this occurs, the control system must either be sufficiently robust to handle new uncertainties or it must have an adaptive mechanism that can adjust its control policy online. The following results show how the system described above can update to improve its performance where the uncertainties are different in operation compared to in training.

Table 6.6 describes the distribution of initial conditions and disturbance forces used in the following tests. The initial position and velocity are sampled from a normal distribution with parameters as shown in the table. The mean values of initial position and velocity were chosen to be near the extreme of the initial conditions in training and their standard deviations are based on the values used in [193]. In this case, as with previous results, the agent is assumed to know the lander's position and velocity precisely and the initial conditions are sampled from a normal distribution. The disturbance force has the same standard deviation as before, but now has a mean of $-500N$ in each direction. This value was chosen to give a significant change in the agent's performance, as will be shown. The change in the disturbance forces could represent certain changes in the environment dynamics such as the lander's thrust profile or atmospheric disturbances.

Initially, the same agent was tested in the environment with the same disturbances and updated disturbances to show how this affects the agent's performance. Figure 6.6 shows the distributions of terminal states for both cases using the best performing agent described above without online updates. These results clearly show the change in distribution of terminal states resulting from the change in environmental disturbances. The distribution of terminal velocities in the x- and y-directions is spread over a wider

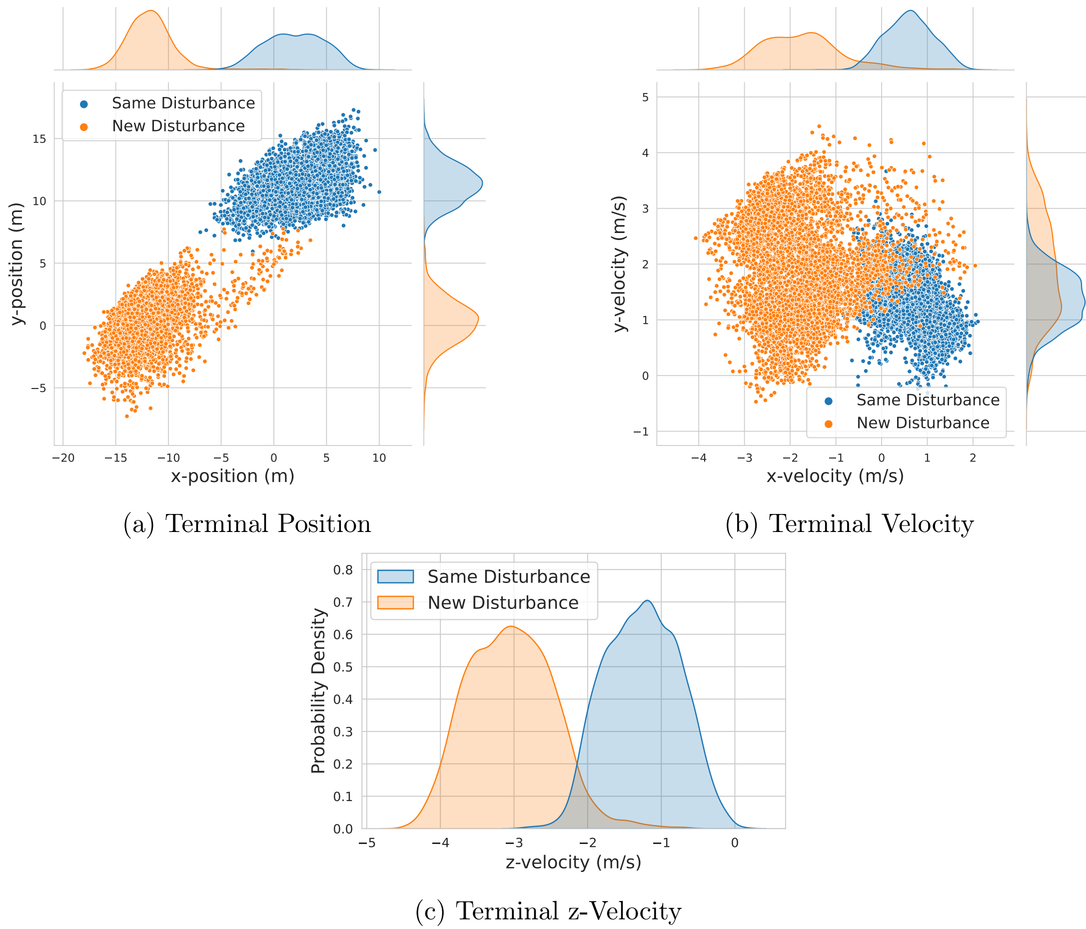
Table 6.6: Initial conditions and uncertainties for the environment with new disturbances.

Parameter	Value
Downrange position, r_x (m)	1800
Crossrange position, r_y (m)	-900
Elevation position, r_z (m)	2390
STD position (m)	5
Downrange velocity, v_x (m/s)	-60
Crossrange velocity, v_y (m/s)	-25
Elevation velocity, v_z (m/s)	-85
STD velocity (m)	1.67
Mass, m (kg)	2000
$\mathbf{F}_D$ mean (N)	-500
$\mathbf{F}_D$ STD (N)	100

range of values with the new disturbances and has a larger average magnitude in the z-direction.

The following results use online updates to improve the agent's performance with new disturbances. This assumes the agent has access to a model of the environment onboard that correctly models the new disturbances. In practice, this would require model identification where the environment model updates based on observed data. For the purposes of this work, the onboard model available to the agent is assumed to be updated without considering the mechanism for how it updates.

Based on the updated model and a known initial state with variance, the agent performs EQLM updates for a number of online training episodes. Results are shown for the number of online training episodes $N_{ep} = 2^i$, $i = 1, 2, \ldots, 7$. All of these runs used the best performing agent as above following offline EQLM updates. As with offline updates, the agent's memory consisted of state transitions from a random agent, the optimal control demonstrations, and the agent itself. Since it would be expensive to update all the replay memory with the new model, these state transitions still use the original environment model on which the agent was trained. In addition to these, agent updates include new observations from the updated environment model. For each number of training episodes, agents were trained with 10 different random seeds and tested for 500 episodes for a total of 5000 testing episodes in each configuration.



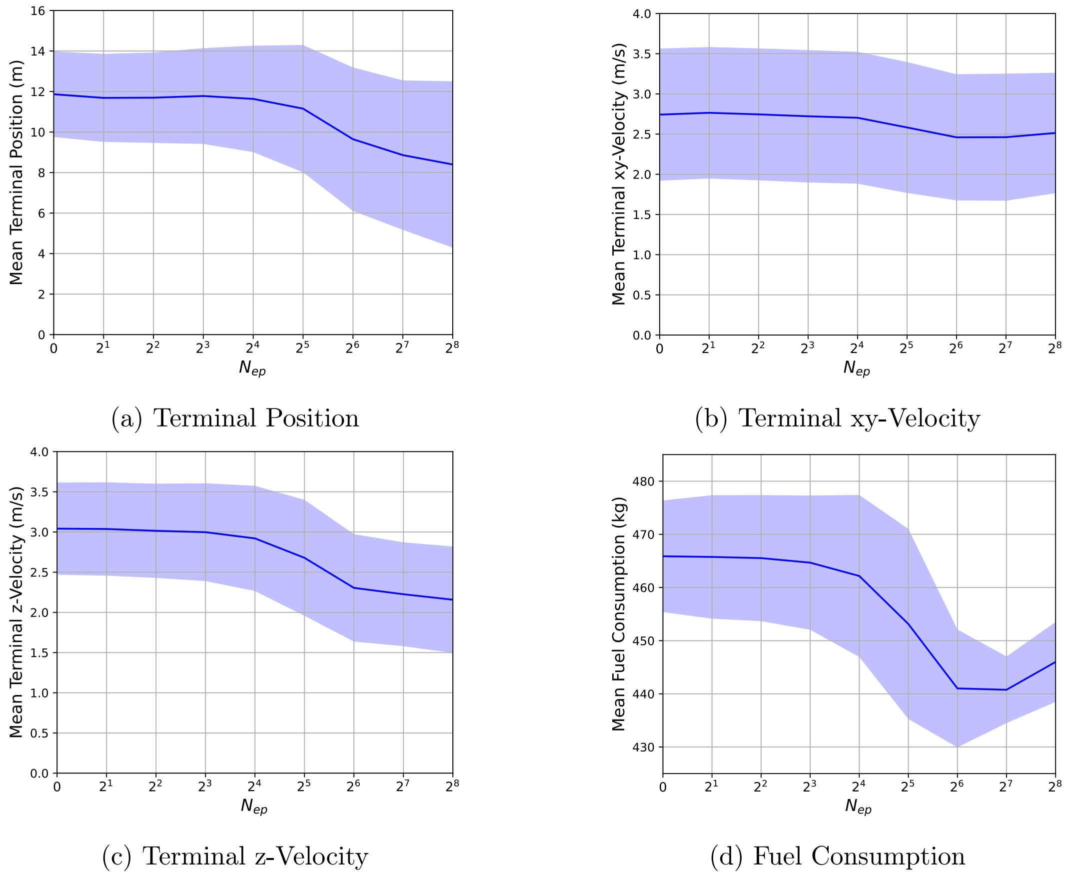
(a) Terminal Position

(b) Terminal Velocity

(c) Terminal z-Velocity

Figure 6.6: Scatter and KDE plots showing distributions of terminal landing states with the original training environment and environment with new disturbance magnitudes.

Figure 6.7 shows the mean terminal states and fuel consumption across the range of number of training episodes N_{ep}, where $N_{ep} = 0$ corresponds to the agent without updates. All of the values shown tend to decrease with increasing number of training episodes after 2^4 episodes. The mean terminal xy-velocity shows only a minor improvement compared to the position and z-velocity. Noting the truncated y scale in Figure 6.7d for clarity, although the relative change in mean fuel consumption is modest, there is still a difference of approximately 25kg between the fuel consumption without updates and with $N_{ep} = 2^7$. The fuel consumption is also greater for $N_{ep} = 2^8$ than when $N_{ep} = 2^7$. This could be due to the lower z-velocities requiring more control effort to achieve.



(a) Terminal Position

(b) Terminal xy-Velocity

(c) Terminal z-Velocity

(d) Fuel Consumption

Figure 6.7: Variation in mean terminal states and fuel consumption with number of online EQLM training episodes. Shaded area indicates +/- one standard deviation.

The following results consider the agent trained for $N_{ep} = 64$ online training episodes as a reasonable compromise between the agent's performance and training time required. Table 6.7 shows statistics of the terminal state and fuel consumption for the agent trained with only offline updates and with online updates. The online updated agent performs better on average for all of the values shown. However, this agent has a slightly larger variance in its performance compared to the offline updated agent. In addition, the worst-case performance as shown by the maximum values of each metric is larger for all but the z-velocity with the online updated agent. This highlights the additional potential for performance variations brought about by including online updates that are stochastic in nature. Nevertheless, including online updates for the changed disturbances results in better performance on average than without online updates.

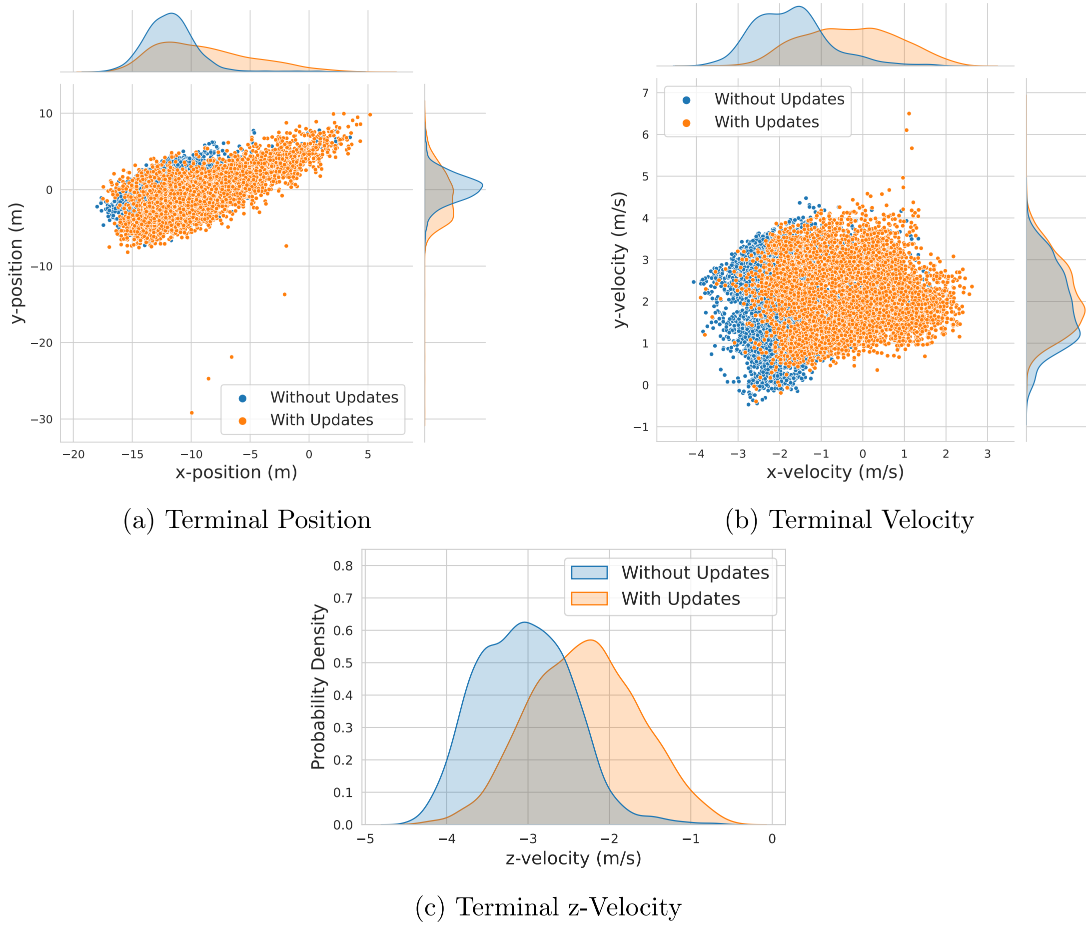
Table 6.7: Comparison of test results in the environment with disturbance changes for agents with and without further online updates. Statistics shown for terminal state and fuel consumption over 5000 test episodes.

		Mean	Min.	Max.	STD
Offline Updated	xy-position (m)	11.865	2.972	18.114	2.112
	xy-velocity (m/s)	2.742	0.950	4.755	0.822
	z-velocity (m/s)	3.042	0.623	4.490	0.574
	Fuel (kg)	465.9	424.6	491.6	10.5
Online Updated	xy-position (m)	9.646	1.102	30.835	3.541
	xy-velocity (m/s)	2.460	0.498	6.595	0.785
	z-velocity (m/s)	2.305	0.437	4.418	0.668
	Fuel (kg)	441	408.2	493.2	11.1

Figure 6.8 shows the distribution of terminal states for each of the agents over the 5000 testing episodes. The distributions of terminal z-velocities clearly show how the online updates improve the controller's performance by shifting the distribution closer to 0m/s. Although the maximum xy-positions and velocities are larger for the agent with online updates, Figures 6.8a and 6.8b highlight that this is due to outliers in the distribution. Overall, the distribution of points moves closer to the desired terminal state with updates as indicated in the average values.

Figure 6.9 shows an example trajectory from the online updated agent along with the commanded thrusts and environmental disturbances. The initial state is the mean initial state as shown in Table 6.6. Its terminal position is $10.6m$ from the desired landing site with a terminal velocity of $1.7m/s$ and it consumes $435.14kg$ of fuel over the trajectory. Although the disturbance forces have a considerably larger magnitude than those used in the initial training, the agent still effectively controls the lander to a soft landing at a short distance from the desired landing site.

As with the agent results in the unchanged environment, the terminal velocity distribution can be improved by including an optimal $10m$ descent. The agent with EQLM updates was tested with the updated disturbances the same as previously with the terminal state adjusted by $10m$ followed by the optimal descent. Table 6.8 shows a comparison of the terminal states of this agent with the full optimal trajectories with updated disturbances. Clearly, when the mean disturbance force is this large, the open-



(a) Terminal Position

(b) Terminal Velocity

(c) Terminal z-Velocity

Figure 6.8: Scatter and KDE plots showing distributions of terminal landing states for agents with and without further online updates in the environment with new disturbance magnitudes.

loop optimal trajectories are no longer effective and result in consistently large terminal velocities and positions. On the other hand, the agent with optimal $10m$ descent has a maximum terminal velocity in the xy-direction of $1.713m/s$ and in the z-direction of $1.342m/s$. This further demonstrates how the combination of RL with traditional optimal control methods can be beneficial for uncertain environments.

All of the results from online updating agents shown above use EQLM updates. This was chosen since EQLM can provide more substantial changes in weights, and therefore policy, online than gradient-based updates. To illustrate this, Figure 6.10 compares the norm change in weights for different update mechanisms, which was calculated as $\|\beta_t - \beta_0\|$ for step t where β_0 is the initial value of the output weights. The initial weights

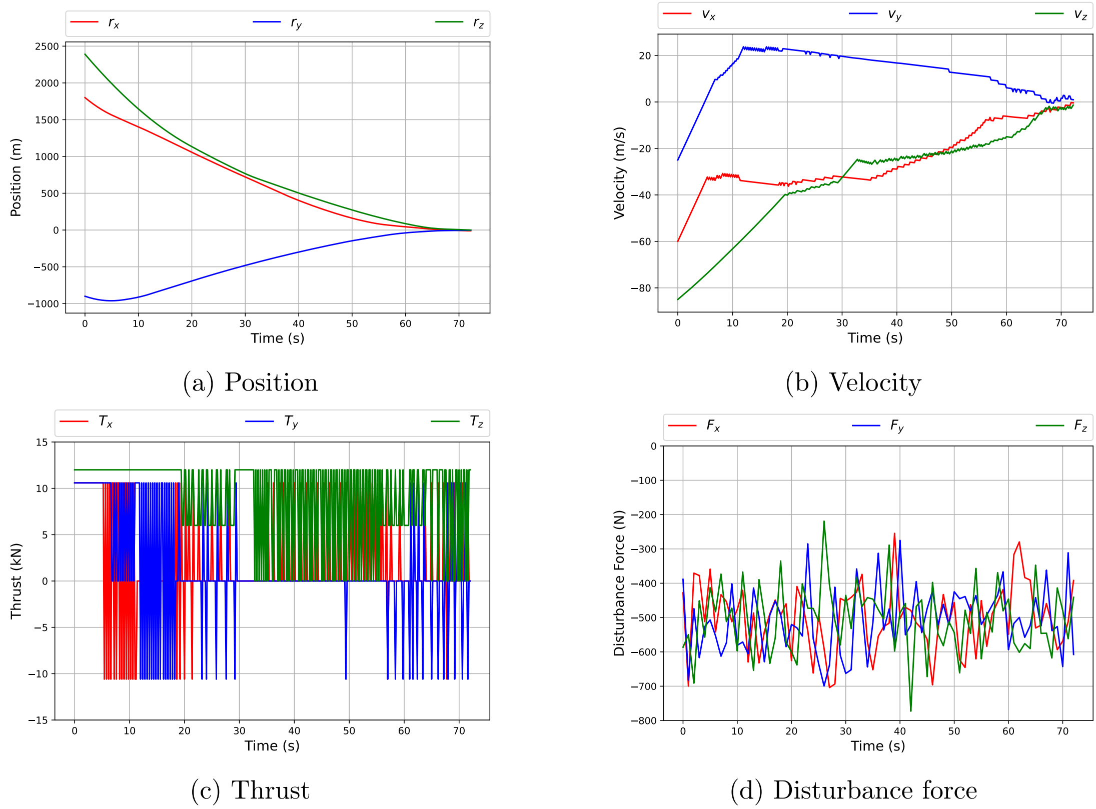


Figure 6.9: Trajectory of the agent trained with online updates and the environmental disturbance forces along this trajectory.

and the data used for updating were the same across each of the agents. The gradient-based updates used RMSProp optimisation as before with two different learning rates of 10^{-5} and 10^{-4} and updates only performed on the output weights. Results are shown in the figure for the average norm change over 8 separate runs of 100 update steps. Increasing the order of magnitude of the learning rate did also correspond to an order of magnitude increase in the norm weight change. However, the norm change in weights is still far lower for both cases than that of EQLM. The other notable difference in trends is that EQLM has more fluctuations in the magnitude of the weight updates. Since ELM-based updates aim to solve directly for the optimal output weights, large variations in their value are to be expected compared to gradient-based updates.

Table 6.8: Comparison of test results from the agent with EQLM updates + 10m optimal descent and the full trajectory optimisation with updated disturbances. Statistics shown for terminal state and fuel consumption over 5000 test episodes for the EQLM agent and 100 test episodes for each of the 33 initial states of the optimal trajectories.

		Mean	Min.	Max.	STD
EQLM Updates + 10m descent	xy-position (m)	10.452	0.804	31.863	3.827
	xy-velocity (m/s)	1.322	0.693	1.713	0.1678
	z-velocity (m/s)	0.934	0.423	1.342	0.138
	Fuel (kg)	452.6	423.6	505.2	11.1
Full Optimal Trajectories	xy-position (m)	217.739	200.163	236.467	5.174
	xy-velocity (m/s)	34.786	29.624	35.957	0.459
	z-velocity (m/s)	41.561	35.035	42.644	0.547
	Fuel (kg)	191.8	191.8	201.3	0.7

6.3.5 Hardware Results

The final results shown here are the time taken to update the agent when running on edge hardware. The hardware used for testing online updates is the NVIDIA Jetson Nano 2GB developer kit. It is equipped with a 64-bit Quad-core ARM A57 CPU at 1.43 GHz and a 128-core NVIDIA Maxwell GPU. The onboard memory for the developer kit is 2 GB 64-bit LPDDR4. The methods detailed previously were implemented in the Tensorflow Python library which makes it possible to use GPU acceleration.

Figure 6.11 shows histograms of the time taken to select an action and perform EQLM updates over 500 steps. Action selection always takes less than $0.01s$. Weight updates take longer and are mostly in the range $0.04 - 0.07s$. On a few occasions these updates last slightly longer but always less than $0.1s$. Since the sampling time in this environment is $0.2s$, this shows that updates can be performed online between timesteps sufficiently quickly using edge hardware suitable for spacecraft. As stated above, the approach to improve the agent's performance in response to changes in the environment would require model identification and multiple episodes of updates. With the update times shown here, 64 episodes of 371 timesteps on average would take at least $415s$ to simulate. This assumes the majority of simulation time is in selecting actions and updating the agent with updates every 4 timesteps. These update times could also be reduced by further optimising the code for updating weights, if necessary.

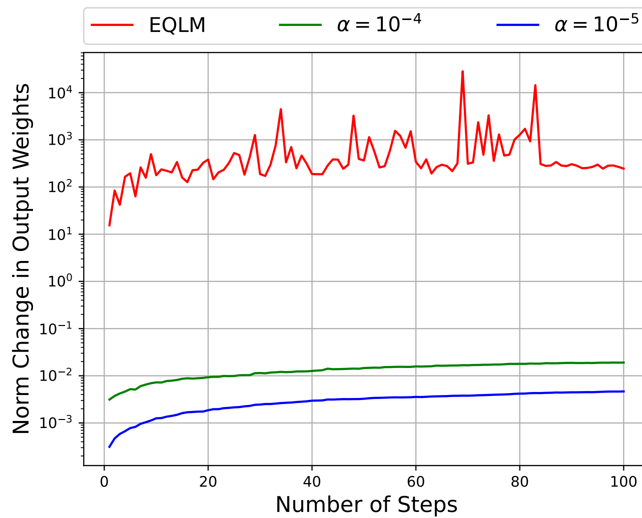


Figure 6.10: Average norm change in output weights over 8 test runs of 100 steps with EQLM updates and gradient based updates.

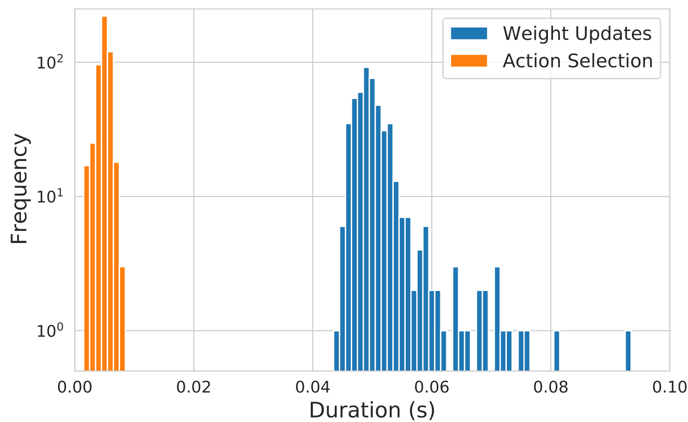


Figure 6.11: Histogram of time taken to select actions and update weights over 500 steps on Jetson Nano hardware.

Although the Jetson Nano hardware allows for GPU acceleration, the scale of network used here is relatively small and so there is not the same benefit of using a GPU as with larger networks. This is highlighted in Table 6.9, which shows statistics of the update times for different hardware configurations. The PC hardware used here ran Ubuntu 18.04 with a 3.6GHz Intel i7-4790 CPU and 8 GB RAM. Timings are also compared for the Jetson Nano configured with and without the GPU. Clearly the PC times

are fastest and on average approximately 10 times faster than either configuration of the Jetson Nano. It is interesting to note that in this case the use of a GPU results in slower updates. This is likely caused by the communication bottleneck between CPU and GPU, which is more significant for this relatively small size of network. To illustrate this, Figure 6.12 shows the variation in update times for different network sizes. These networks all have the same number of hidden layers but different numbers of hidden nodes - $(N, \frac{2}{3}N, N)$ in each layer, where $N = 300$ is the size used in previous experiments. At the smaller network sizes, the CPU updated slightly faster on average without the GPU. For networks with more than 450 output weights, GPU acceleration did provide a speed-up in update times in this example.

Table 6.9: Timings for weight updates across different hardware configurations.

	Update Duration (ms)			
Device	Mean	Min.	Max.	STD
PC	5.06	3.03	22.86	2.58
CPU	47.5	40.92	88.6	6.95
CPU+GPU	51.16	44.86	93.06	5.04

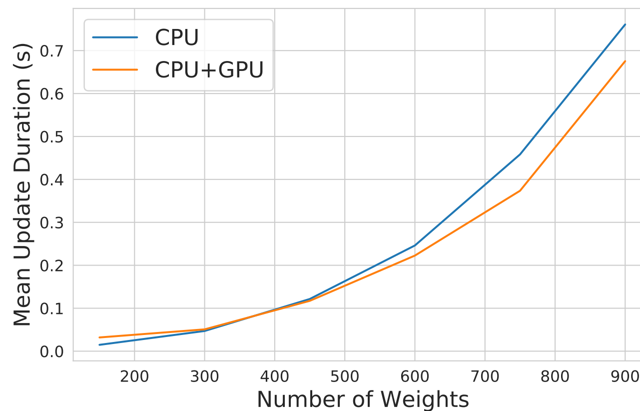


Figure 6.12: Trend in update times for different network sizes with and without a GPU on edge hardware.

6.4 Summary

This chapter introduced an approach for updating Q-networks online via EQLM, which is an alternative to gradient-based updates. The results demonstrate that EQLM can improve the performance of an agent from entirely offline updates. However, as with many other RL approaches, the performance of EQLM is sensitive to several random variables, such as the data for initialisation and updating. Therefore, this approach requires careful offline testing over several random seeds to find the best model. In the event of a changing environment, where the distribution of random disturbances are different to those used in training, online updates can compensate for these changes. In both cases with or without environment changes, the agent's performance in terms of the soft landing requirement is further improved by using an optimal trajectory over the final 10m of descent. Results shown on flight suitable hardware indicate that the online updates could indeed run onboard a spacecraft. However, to allow sufficient time for model updating and multiple training episodes in practice may require improvements in the update times.

As discussed, the approach described here relied heavily on running over multiple random seeds to achieve desirable performance. This is the case both in training the agents and when testing their performance. Although this approach can perform well even in changing environments, its stochastic nature makes it less likely to be used in practice. Further research is necessary to reduce the agent's sensitivity to random seeds in training. In addition, the assumption of an accurate environment model for online updating is non-trivial to achieve. Future work should combine the approach proposed here with methods of model identification to better assess its feasibility.

Chapter 7

Conclusions

This thesis explored the application of Intelligent Control for onboard spacecraft GNC. Methods from reinforcement learning show promise as an approach for training control systems that can also learn online, which can be beneficial in highly uncertain environments. This has been demonstrated using the spacecraft powered descent problem as a test case. The following outlines the thesis' contributions with respect to each of the objectives introduced previously, along with a discussion of limitations and recommendations for future work.

7.1 Summary of Objectives

The first objective was to create a system of classification for IC. Chapter 3 presents the results of this effort. This was achieved by analysing literature to extract 3 primary dimensions of IC in terms of uncertainty: environment, control system, and goals. IC systems deal with uncertainties in one or more of these dimensions and the ability to deal with greater levels of uncertainty indicates a higher level of intelligence. As discussed in this chapter, adaptive or learning capabilities are key components of IC that are explored in later chapters.

Chapter 4 demonstrated the applicability of RL methods to the spacecraft powered descent problem. However, it also showed a potential failure mode of RL in this context that is commonly referred to as "reward-hacking". The second objective that is relevant

to this was to combine RL with conventional control methods to exploit the benefits of both. Chapter 5 introduced the proposed approach to achieve this. In this application, a convex optimisation procedure was the conventional control method used to generate demonstration data for a RL agent. Making these optimal control solutions suitable for use in demonstrations required additional steps of discretising actions and adjusting the states in the solutions. Including a small quantity of demonstration data in the agent's replay memory successfully mitigated the reward-hacking behaviour seen when training an agent without demonstrations.

It was noted above that online adaptation and learning are important characteristics of IC. Therefore, another key objective of this research was to demonstrate online updates using RL. Furthermore, since this work is interested in IC running onboard spacecraft, an additional objective was to demonstrate this method applied on flight suitable hardware. The proposed method of online updates, "EQLM", was introduced in Chapter 6. EQLM uses theories from ELMs to give sufficiently fast weight updates online. Compared to a RL agent that does not update, online updates can better adapt the agent's policy to changes in the environment. This chapter also gave further contribution to the objective of combining conventional control methods with RL. By using a RL agent to handle uncertainties for most of the powered descent and a convex optimisation procedure to land the spacecraft, this improves the agent's performance with respect to the soft-landing requirement. Experiments on flight suitable hardware showed that this method can be implemented onboard spacecraft and allow for sufficiently fast updates to run online.

The original question posed in the title of this thesis is: can spacecraft think? As previously acknowledged, a better question might be: can spacecraft mimic human intelligence in a useful manner? The taxonomy introduced in Chapter 3 gives us a means of answering this question by considering the level of intelligence of a spacecraft's control system. Human intelligence allows us to deal with many types of uncertainty, which can be replicated in a control system to some degree. The online updating control system described in Chapter 6 can be classed as E-2, C-1, G-1 for the application of spacecraft powered descent shown here. The reasons for this are as follows:

- *E-2*: The control system operates in an environment where the magnitude of disturbance forces can change. The dynamics governing the 3-DOF problem formulation used here are well established, but subject to uncertainties.

- *C-1*: The parameters of the DNN that govern the agent's policy can update online. Although the proposed method could in theory handle uncertainties in sensor and actuator performance, which would class it as C-2, this was not specifically tested in this work.

- *G-1*: The goals of a soft pinpoint landing and minimal fuel consumption are defined implicitly in the reward function. In the powered descent problem, these goals are unlikely to change. However, if the agent were required to select a landing site, which might change during the descent, this would result in a higher level of goal uncertainty.

In a hierarchical IC scheme with three main layers, the proposed approach can serve as the middle layer of this system. This middle layer adapts the control system according to both observations passed up from the lower level and plans given by the higher level. At the lowest level are conventional controllers for actuators with fast response times. In the case of spacecraft powered descent, this is the attitude control system. At the highest level is activity planning, which passes plans down the hierarchy to be executed by lower-level systems. This planning level may not be necessary for powered descent GNC, but this level could be more relevant in applications such as robotic exploration.

7.2 Limitations and Future Work

The work described here is not without limitations—some of which are discussed in the relevant chapters. While the proposed approach is sufficiently general to be applicable to many spacecraft GNC problems, this work only investigated the application to spacecraft powered descent. In theory, other problems where a suitable state space, action space, and reward function can be defined could be solved with the proposed method of online updating RL. Future work should investigate the application of this

method to other GNC problems that can be formulated as RL problems. Examples of such GNC problems with significant uncertainties that could incorporate this approach include docking with a non-cooperative target for active debris removal, GNC around small bodies, or autonomous extra-terrestrial rover exploration. In addition, applications where existing methods can generate demonstrations under nominal conditions can help in training the agent. However, the process of creating demonstration data is problem specific since, in this case, it required adjusting the action space and trajectories. If an off-policy RL algorithm that allows continuous actions were used instead of DQN, this process of adjusting demonstrations could be avoided, but potentially at the expense of longer training times.

The types of uncertainties considered here were limited to the initial conditions and environmental disturbance forces. While these disturbances can account for various uncertainties in both the environment and control system, these should be analysed individually to validate the fault-tolerant performance of the controller. For example, considering cases where actuators behave in physically realistic but unexpected ways. Furthermore, the results shown assume the agent observes its true state using an exact model of the environment. Including realistic sensor measurements even without the presence of faults would lead to some uncertainty in the spacecraft's state. How well the proposed approach performs with uncertainties in these aspects remains to be seen.

As discussed in Chapter 6, adjusting the agent's policy online in response to environment changes requires an updated model of the environment. This was assumed to occur separately from the agent's online training, such that any online updates occur with a "correct" model. A more realistic scenario would be where both the network weights and the environment model adapt online in response to new observations. In the powered descent problem, updates are highly time constrained. The results in Chapter 6 showed that an agent requires many simulated interactions with the environment to update its policy. Therefore, an open question remains as to whether an IC system trained in this manner can simultaneously update an environment model and its control policy over a realistic timescale.

The main barrier to incorporating IC onboard spacecraft in practice is the lack of formal Verification and Validation (V&V) procedures. Typically, for conventional GNC systems, V&V consists of conducting mathematical analyses to give certain guarantees on the controller's performance combined with extensive simulation studies. The complex, nonlinear nature of DNN-based controllers makes them difficult to analyse in the same manner as conventional controllers. Given the stochasticity inherent in RL agents, this also presents further challenges in generating empirical guarantees on their performance from simulations. The results in Chapter 6 demonstrated how many training runs can be necessary in RL problems since an agent's performance depends heavily on random seeds. For this method to be of practical application for spacecraft GNC, new approaches of V&V will be necessary. As well as being used in model evaluation, new V&V procedures could also improve the RL training process by finding ways to guarantee a certain level of performance without requiring many different training runs.

This research represents a step towards greater levels of intelligence onboard spacecraft. While conventional control approaches will remain essential in this field, as the scope of space missions become more ambitious, control systems will accordingly require increased levels of intelligence to realise these missions. At the highest level of intelligence, an IC system would be able to operate in an unexplored environment about which very little is known. During its operation, it would design its control laws from the ground up as required by the environment in which it operates. Its initial goals could be very vague, but from this it would create its own goals and plans to achieve these. At this point, a spacecraft's control system may well deceive a human operator into believing the spacecraft can think.

Appendix A

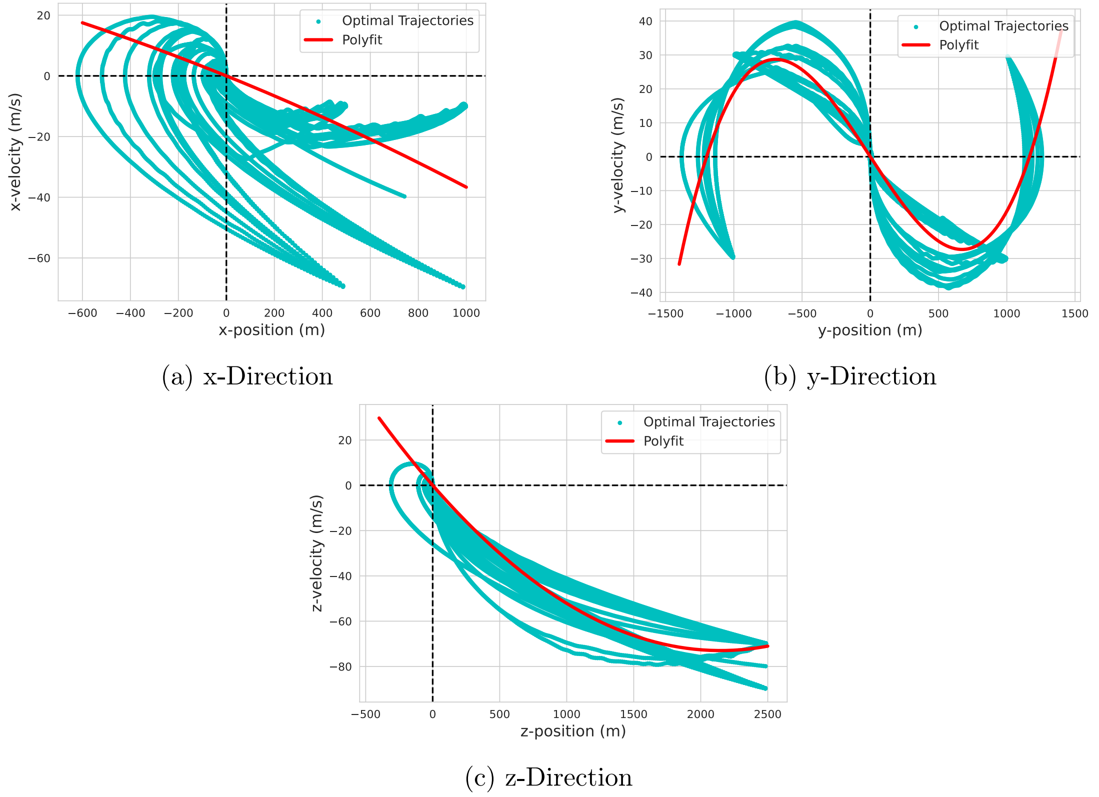
Optimal Demonstrations for Reward Shaping

The target velocity, $\mathbf{v}_{targ}$ of Equation 4.6 in the shaped reward function was derived in previous works using a heuristic approach [23]. This appendix presents preliminary results from attempting to leverage optimal demonstrations in the reward function as well as agent training.

A.1 Reward Shaping Method

Previous work described an approach for generating optimal control demonstrations via nonlinear optimisation methods [32]. The output of these optimisations are control points that can be interpolated to give a full trajectory of commanded thrusts and spacecraft states. Figure A.1 shows the positions and velocities in each direction along the optimal trajectories starting from 33 different initial conditions. Although some of these trajectories give negative altitudes, indicating crashes, these were still suitable for use as demonstrations following action discretisation. These positions and velocities can be used to create a target velocity profile for reward shaping.

The data for fitting target velocities in each direction are the positions r_i and velocities v_i at each timestep in all trajectories. The lander's position is the independent variable with velocity as the dependent variable. A polynomial function of the position



(a) x-Direction

(b) y-Direction

(c) z-Direction

Figure A.1: Positions and velocities from optimal trajectories and the polynomial fit target velocities.

can serve as a very simple model for the relationship between position and velocity. The target velocity in the i-direction can then be expressed as

$$v_i^{targ} = c_{i3} \cdot r_i^3 + c_{i2} \cdot r_i^2 + c_{i1} \cdot r_i, \tag{A.1}$$

where c_{i1}, c_{i2}, and c_{i3} are coefficients determined for each direction using a nonlinear least-squares fit. Table A.1 shows the values of these coefficients from this fit. In the x- and z-directions, a second degree polynomial was used while in the y-direction a third degree polynomial was used. These were chosen to best capture the shapes of the target velocities as shown in Figure A.1. Since this target velocity does not account for the spacecraft's initial velocity, its profile can differ substantially from the optimal trajectories—especially in the x-direction. Nevertheless, as will be shown even this very simple velocity profile can be suitable as a target velocity for reward shaping.

Table A.1: Polynomial coefficients for target velocity.

Direction	c_{i3}	c_{i2}	c_{i1}
x	0	-4.688×10^{-6}	-3.199×10^{-2}
y	4.399×10^{-8}	1.459×10^{-6}	-6.153×10^{-2}
z	0	1.582×10^{-5}	-6.796×10^{-2}

A.2 Simulation Setup

The powered descent environment formulation and training procedure used here was the same as in previous work with a different shaped reward function [32]. The ranges of initial conditions used in training are shown in Table A.2, from which the initial states are uniformly sampled. As well as uncertainties in initial conditions, this environment includes disturbance forces sampled from a normal distribution with mean $0N$ and standard deviation $100N$. 32 training runs of 10,000 episodes were carried out with different random seeds.

Table A.2: Range of initial conditions for training the agent.

Parameter	Min. Value	Max. Value
Downrange position, r_x (m)	400	1100
Crossrange position, r_y (m)	-1100	1100
Elevation position, r_z (m)	2400	2600
Downrange velocity, v_x (m/s)	-75	-5
Crossrange velocity, v_y (m/s)	-35	35
Elevation velocity, v_z (m/s)	-95	-65
Mass, m (kg)	2000	2000

The reward function takes the same form as Equation 4.4 with the target velocity calculated according to Equation A.1. The state representation in this case was not shaped to the target velocity and used the “raw” form of Equation 4.13. The agent’s action space has 7 discrete magnitudes with a maximum of $10kN$ in the x- and y-directions and 3 discrete magnitudes with a maximum of $13kN$ in the z-direction.

A.3 Results

Results are shown for the best performing agent in terms of mean cumulative reward over the final 100 episodes of training. Figure A.2 shows the learning curve and number of steps over the training episodes for this agent. There are a few occasions where the cumulative reward drops sharply during training, but then recovers and continues to increase. The trend in the number of steps shows that the agent often has episodes that last for close to the maximum duration of 500 timesteps. This was also observed when testing the trained agent as shown below.

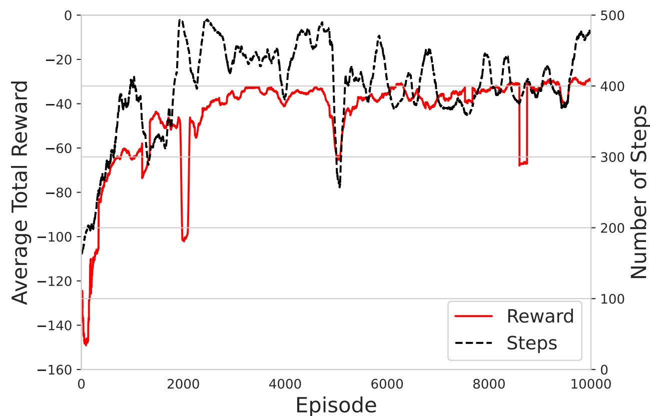
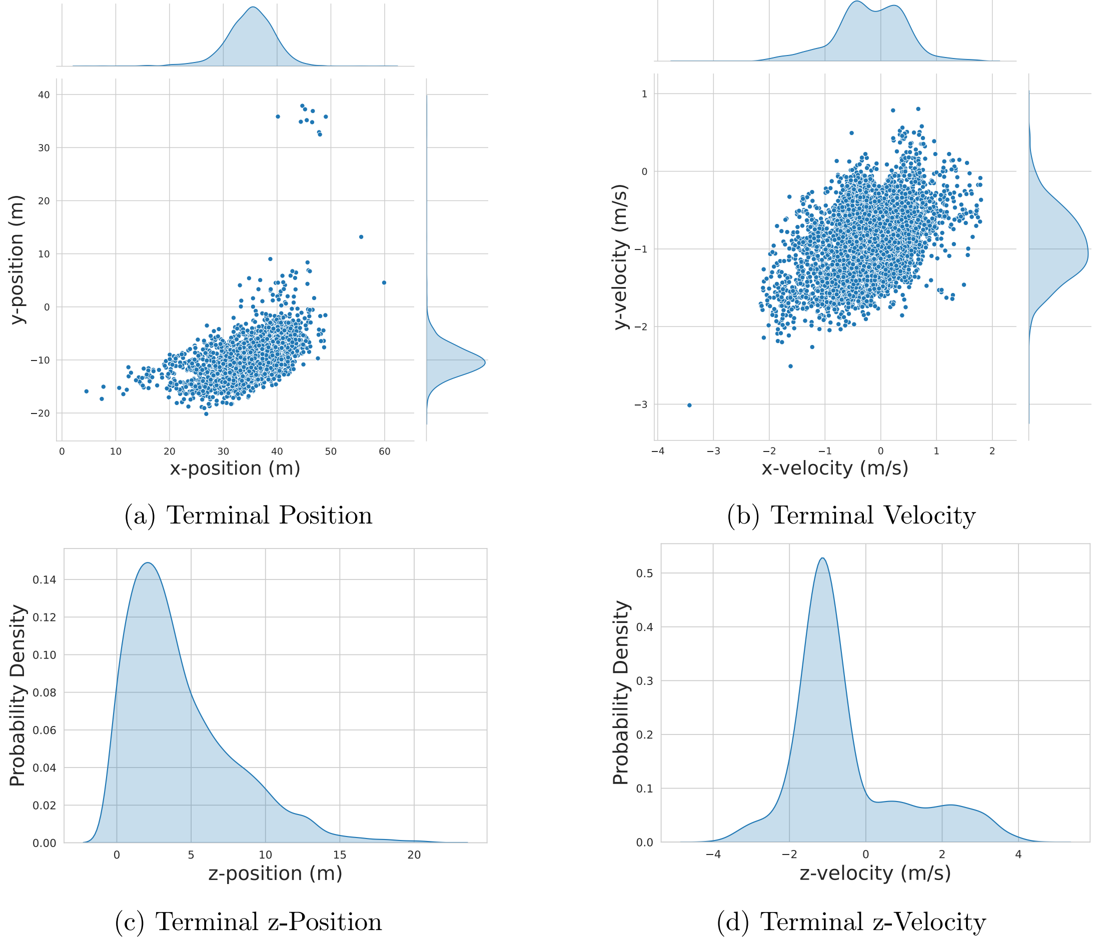


Figure A.2: Average reward and steps per episode over a training run with a new shaped reward. Uniformly filtered (average) over 150 episodes for clarity.

The best performing agent carried out 5000 test episodes. These episodes all used the same initial conditions of $\mathbf{r}_0 = \begin{bmatrix} 1 & 1 & 2.5 \end{bmatrix} km$, $\mathbf{v}_0 = \begin{bmatrix} -60 & 30 & -90 \end{bmatrix} m/s$, and $m_0 = 2000kg$ with uncertainties only present in the disturbance forces. Figure A.3 shows the distribution of terminal states over the testing episodes. Note that the terminal altitude is often greater than 0 as shown in Figure A.3c due to the episode terminating after 500 timesteps. The terminal z-velocity also has a large range of values—some of which are positive. In total, 283 of the 5000 trajectories reached a terminal landing state of $r_z = 0$, with the other trajectories keeping the lander hovering until the episode terminates. This suggests a different type of reward-hacking to that of the agent in Chapter 4 where, instead of abruptly terminating the episode, the agent aims to remain at a low velocity near the desired landing point without reaching it. This is highlighted

in Table A.3, which shows summary statistics of terminal states and fuel. Note that the z-velocity statistics are taken for the absolute values of velocity magnitude, whereas the z-position (altitude) is relative to $z = 0$. Due to the episodes often lasting for the maximum duration, the fuel consumption is also relatively high.



(a) Terminal Position (b) Terminal Velocity

(c) Terminal z-Position (d) Terminal z-Velocity

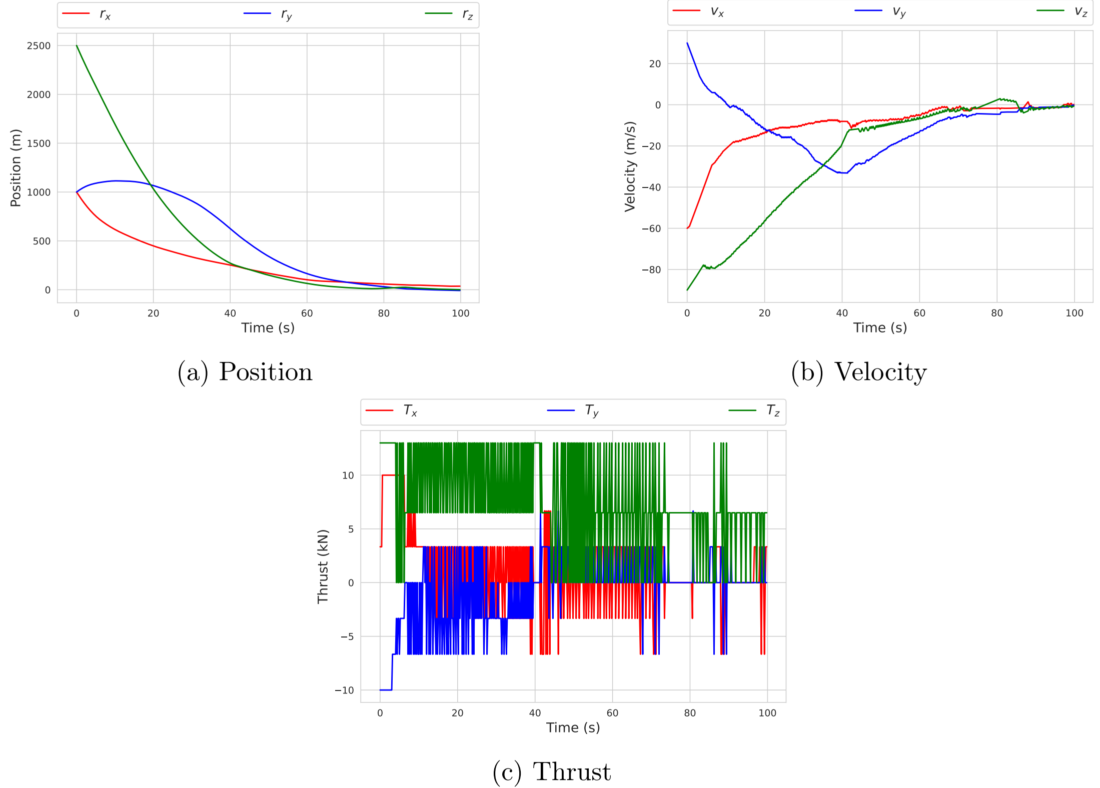
Figure A.3: Scatter and KDE plots showing distributions of terminal states for the best agent trained with new shaped reward.

Figure A.4 shows an example trajectory of this agent that lasts for the maximum duration of 500 timesteps, or $100s$. Starting from the initial conditions shown above, at the end of the episode its terminal state is $\mathbf{r} = \begin{bmatrix} 36.376 & -8.079 & 0.934 \end{bmatrix} m$ and $\mathbf{v} = \begin{bmatrix} 0.169 & -0.244 & -0.658 \end{bmatrix} m/s$ with a fuel consumption of $608.3kg$. Although the spacecraft's altitude is close to zero after around $75s$, instead of landing the agent keeps the lander near this state until the episode terminates. This shows that the proposed

Table A.3: Test results from the agent trained with new shaped reward. Statistics shown for terminal state over 5000 test episodes.

	Mean	Min.	Max.	STD
xy-position (m)	36.539	16.552	60.742	4.068
xy-velocity (m/s)	1.133	0.081	4.565	0.444
z-position (m)	4.240	-0.309	21.663	3.560
z-velocity (m/s)	1.355	3.213×10^{-4}	4.566	0.767
Fuel (kg)	611.5	585.7	647.1	7.8

reward shaping via demonstrations can motivate an agent towards the desired landing point, but might still result in some undesirable behaviours. These results used a highly simplified form of target velocity that could be improved to better approximate optimal trajectories. In addition, an approach similar to that introduced in Chapter 6 of using optimal control for the final descent could also improve the performance of an agent trained in this manner.



(a) Position

(b) Velocity

(c) Thrust

Figure A.4: Trajectory of the agent over a sample episode with commanded thrusts.

Bibliography

[1] V. I. Moroz, W. T. Huntress, and I. L. Shevalev, "Planetary Missions of the 20th Century*," *Cosmic Research*, vol. 40, no. 5, pp. 419–445, 2002.

[2] C. A. Graves and J. C. Harpold, "Apollo experience report: Mission planning for Apollo entry," National Aeronautics and Space Administration, Tech. Rep. NASA TN D-6725, Jun. 1972, nTRS Author Affiliations: NASA Lyndon B. Johnson Space Center, NTRS Document ID: 19720013191, NTRS Research Center: Legacy CDMS (CDMS).

[3] A. Longobardo, *Sample Return Missions: The Last Frontier of Solar System Exploration.* Elsevier, 2021.

[4] L. Harra and D. Müller, "Solar orbiter: a short review of the mission and early science results," *Astrophysics and Space Science*, vol. 370, no. 2, p. 12, 2025.

[5] K. A. Capova, "Introducing Humans to the Extraterrestrials: the Pioneering Missions of the Pioneer and Voyager Probes," *Frontiers in Human Dynamics*, vol. 3, 2021.

[6] T. Tolker-Nielsen, "EXOMARS 2016 - Schiaparelli Anomaly Inquiry," European Space Agency, Tech. Rep. DG-I/2017/546/TTN, 2017.

[7] G. N. Saridis, "Toward the realization of intelligent controls," *Proceedings of the IEEE*, vol. 67, no. 8, pp. 1115–1133, 1979.

[8] P. M. Krafft, M. Young, M. Katell, K. Huang, and G. Bugingo, "Defining AI in Policy versus Practice," in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society.* Association for Computing Machinery, 2020, pp. 72–78.

[9] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A Comprehensive Overview of Large Language Models," *ACM Transactions on Intelligent Systems and Technology*, vol. 16, no. 5, pp. 1–72, 2025.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, 2nd ed., ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: MIT Press, 2018.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 2013, arXiv:1312.5602. [Online]. Available: https://arxiv.org/abs/1312.5602

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, 2016.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[14] M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang, "Autonomous navigation of stratospheric balloons using reinforcement learning," *Nature*, vol. 588, no. 7836, pp. 77–82, 2020.

[15] Y. Wang, H. Tang, L. Huang, L. Pan, L. Yang, H. Yang, F. Mu, and M. Yang, "Self-play reinforcement learning guides protein engineering," *Nature Machine Intelligence*, vol. 5, no. 8, pp. 845–860, 2023.

[16] K. Reuer, J. Landgraf, T. Fösel, J. O'Sullivan, L. Beltrán, A. Akin, G. J. Norris, A. Remm, M. Kerschbaum, J.-C. Besse, F. Marquardt, A. Wallraff, and C. Eichler, "Realizing a deep reinforcement learning agent for real-time quantum feedback," *Nature Communications*, vol. 14, no. 1, p. 7138, 2023.

[17] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[18] J. Seo, S. Kim, A. Jalalvand, R. Conlin, A. Rothstein, J. Abbate, K. Erickson, J. Wai, R. Shousha, and E. Kolemen, "Avoiding fusion plasma tearing instability with deep reinforcement learning," *Nature*, vol. 626, no. 8000, pp. 746–751, 2024.

[19] D. Izzo, M. Märtens, and B. Pan, "A survey on artificial intelligence trends in spacecraft guidance dynamics and control," *Astrodynamics*, vol. 3, no. 4, pp. 287–299, 2019.

[20] B. Gaudet and R. Furfaro, "Adaptive pinpoint and fuel efficient mars landing using reinforcement learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 1, no. 4, pp. 397–411, 2014.

[21] R. Furfaro and R. Linares, "Waypoint-based Generalized ZEM/ZEV Feedback Guidance for Planetary Landing via a Reinforcement Learning Approach," in *3rd International Academy of Astronautics Conference on Dynamics and Control of Space Systems, DyCoSS 2017*. Univelt Inc., 2017, pp. 401–416.

[22] X. Jiang, S. Li, and R. Furfaro, "Integrated guidance for Mars entry and powered descent using reinforcement learning and pseudospectral method," *Acta Astronautica*, vol. 163, pp. 114–129, 2019.

[23] B. Gaudet, R. Linares, and R. Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," *Advances in Space Research*, vol. 65, no. 7, pp. 1723–1741, 2020.

[24] R. Furfaro, A. Scorsoglio, R. Linares, and M. Massari, "Adaptive generalized ZEM-ZEV feedback guidance for planetary landing via a deep reinforcement learning approach," *Acta Astronautica*, vol. 171, pp. 156–171, 2020.

[25] L. Federici and R. Furfaro, "Improving reinforcement learning performance in spacecraft guidance and control through meta-learning: a comparison on planetary landing," *Neural Computing and Applications*, pp. 17 249–17 271, 2024.

[26] A. M. Turing, "Computing Machinery and Intelligence," in *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*. Springer Netherlands, 2009, pp. 23–65.

[27] C. Wilson, F. Marchetti, M. Di Carlo, A. Riccardi, and E. Minisci, "Classifying Intelligence in Machines: A Taxonomy of Intelligent Control," *Robotics*, vol. 9, no. 3, p. 64, 2020.

[28] C. Wilson and A. Riccardi, "Enabling intelligent onboard guidance, navigation, and control using reinforcement learning on near-term flight hardware," *Acta Astronautica*, vol. 199, pp. 374–385, 2022.

[29] ——, "Improving the efficiency of reinforcement learning for a spacecraft powered descent with Q-learning," *Optimization and Engineering*, vol. 24, no. 1, pp. 223–255, 2023.

[30] C. Wilson, F. Marchetti, M. Di Carlo, A. Riccardi, and E. Minisci, "Intelligent Control: A Taxonomy," in *8th International Conference on Systems and Control (ICSC)*. IEEE, 2019, pp. 333–339.

[31] C. Wilson, A. Riccardi, and E. Minisci, "A Novel Update Mechanism for Q-Networks Based On Extreme Learning Machines," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[32] C. Wilson and A. Riccardi, "Leveraging Optimal Control Demonstrations in Reinforcement Learning for Powered Descent," in *8th International Conference on Astrodynamics Tools and Techniques (ICATT)*, 2021.

[33] ——, "Enabling intelligent onboard guidance, navigation, and control using near-term flight hardware," in *72nd International Astronautical Congress*, 2021.

[34] R. Bellman, "Dynamic Programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[35] C. J. C. H. Watkins, "Learning from Delayed Rewards," PhD Thesis, King's College, May 1989.

[36] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[37] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.

[38] L. Baird and A. Moore, "Gradient Descent for General Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 11. MIT Press, 1998.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[40] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," 2016, arXiv:1511.05952. [Online]. Available: https://arxiv.org/abs/1511.05952

[41] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[42] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE transactions on neural networks and learning systems*, vol. 27, no. 4, pp. 809–821, 2015.

[43] S. Pang and X. Yang, "Deep Convolutional Extreme Learning Machine and Its Application in Handwritten Digit Classification," *Computational Intelligence and Neuroscience*, vol. 2016, no. 1, 2016.

[44] V. Dwivedi and B. Srinivasan, "Physics Informed Extreme Learning Machine (PIELM)-–A rapid method for the numerical solution of partial differential equations," *Neurocomputing*, vol. 391, pp. 96–118, 2020.

[45] X. Liu, W. Yao, W. Peng, and W. Zhou, "Bayesian physics-informed extreme learning machine for forward and inverse pde problems with noisy data," *Neurocomputing*, vol. 549, p. 126425, 2023.

[46] V. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.

[47] W. Deng, Q. Zheng, and L. Chen, "Regularized Extreme Learning Machine," in *2009 IEEE Symposium on Computational Intelligence and Data Mining*, 2009, pp. 389–395.

[48] L. Guo, J.-h. Hao, and M. Liu, "An incremental extreme learning machine for online sequential learning problems," *Neurocomputing*, vol. 128, pp. 50–58, 2014.

[49] K. Fu, "Learning control systems and intelligent control systems: An intersection of artificial intelligence and automatic control," *IEEE Transactions on Automatic Control*, vol. 16, no. 1, pp. 70–72, 1971.

[50] P. J. Antsaklis, "Intelligent Learning Control," *IEEE Control Systems Magazine*, vol. 15, no. 3, pp. 5–7, 1995.

[51] G. N. Saridis, "Identification Methods for Intelligent Control Systems," *IFAC Proceedings Volumes*, vol. 12, no. 8, pp. 145–149, 1979.

[52] P. J. Antsaklis, "Defining Intelligent Control: Report of the Task Force on Intelligent Control," IEEE Control Systems Society, Tech. Rep., Dec. 1993.

[53] D. Linkens and H. Nyongesa, “Learning systems in intelligent control: an appraisal of fuzzy, neural and genetic algorithm control applications,” *IEE Proceedings - Control Theory and Applications*, vol. 143, no. 4, pp. 367–386, 1996.

[54] K. Krishnakumar and N. Kulkarni, “Inverse adaptive neuro-control of a turbo-fan engine,” in *Guidance, Navigation, and Control Conference and Exhibit*, 1999.

[55] D. B. Lavallee, C. Olsen, J. Jacobsohn, and J. Reilly, “Intelligent Control For Spacecraft Autonomy-An Industry Survey,” in *Space 2006*, 2006.

[56] J. Mökander, M. Sheth, D. S. Watson, and L. Floridi, “The Switch, the Ladder, and the Matrix: Models for Classifying AI Systems,” *Minds and Machines*, vol. 33, no. 1, pp. 221–248, 2023.

[57] S. Bennett, “A brief history of automatic control,” *IEEE Control Systems Magazine*, vol. 16, no. 3, pp. 17–25, 1996.

[58] J. C. Maxwell, “I. On governors,” *Proceedings of the Royal Society of London*, no. 16, pp. 270–283, 1868.

[59] O. Mayr and L. Bryant, “The origins of feedback control,” *Scientific American*, vol. 223, no. 4, pp. 110–119, 1970.

[60] H. Nyquist, “Regeneration Theory,” *Bell System Technical Journal*, vol. 11, no. 1, pp. 126–147, 1932.

[61] S. Bennett, “Development of the PID controller,” *IEEE Control Systems Magazine*, vol. 13, no. 6, pp. 58–62, 1993.

[62] R. Bellman, “The Theory of Dynamic Programming,” *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[63] A. Bryson, “Optimal control-1950 to 1985,” *IEEE Control Systems Magazine*, vol. 16, no. 3, pp. 26–33, 1996.

[64] K. J. Hunt, D. Sbarbaro, R. Zbikowski, and P. J. Gawthrop, “Neural networks for control systems—A survey,” *Automatica*, vol. 28, no. 6, pp. 1083–1112, 1992.

[65] P. Fleming and R. Purshouse, "Evolutionary algorithms in control systems engineering: a survey," *Control Engineering Practice*, vol. 10, no. 11, pp. 1223–1241, 2002.

[66] K. M. Passino and S. Yurkovich, *Fuzzy Control.* Menlo Park, CA: Addison-Wesley, 1998, vol. 42.

[67] Y. Ichikawa and T. Sawa, "Neural network application for direct feedback controllers," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 224–231, 1992.

[68] J. S. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 665–685, 1993.

[69] Y. Wang, W. Zhou, J. Luo, H. Yan, H. Pu, and Y. Peng, "Reliable Intelligent Path Following Control for a Robotic Airship Against Sensor Faults," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 6, pp. 2572–2582, 2019.

[70] Q. Wu, C. M. Lin, W. Fang, F. Chao, L. Yang, C. Shang, and C. Zhou, "Self-Organizing Brain Emotional Learning Controller Network for Intelligent Control System of Mobile Robots," *IEEE Access*, vol. 6, pp. 59 096–59 108, 2018.

[71] Y. Xu, B. Jiang, G. Tao, and Z. Gao, "Fault Tolerant Control for a Class of Nonlinear Systems with Application to Near Space Vehicle," *Circuits, Systems, and Signal Processing*, vol. 30, no. 3, pp. 655–672, 2011.

[72] S. Li and Y. M. Peng, "Neural network-based sliding mode variable structure control for mars entry," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 226, no. 11, pp. 1373–1386, 2012.

[73] D. A. Handelman, S. H. Lane, and J. J. Gelfand, "Integrating neural networks and knowledge-based systems for intelligent robotic control," *IEEE Control Systems Magazine*, vol. 10, no. 3, pp. 77–87, 1990.

[74] C. H. Chiang, "A genetic programming based rule generation approach for intelligent control systems," in *2010 International Symposium on Computer, Communication, Control and Automation (3CA)*, vol. 1, 2010, pp. 104–107.

[75] F. Marchetti, E. Minisci, and A. Riccardi, "Towards Intelligent Control via Genetic Programming," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–8.

[76] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davis, D. Mandl, S. Frye, B. Trout, S. Shulman, and D. Boyer, "Using Autonomy Flight Software to Improve Science Return on Earth Observing One," *Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 4, pp. 196–216, 2005.

[77] M. Vasile, M. Massari, and G. Giardini, "WISDOM: an Advanced Intelligent, Fault-Tolerant System for Autonomy in Risky Environments," ESA ITI Contract 18693/04/NL/MV, Tech. Rep., 2004.

[78] W. He, Y. Chen, and Z. Yin, "Adaptive Neural Network Control of an Uncertain Robot with Full-State Constraints," *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 620–629, 2016.

[79] H. Chaoui and P. Sicard, "Adaptive Fuzzy Logic Control of Permanent Magnet Synchronous Machines With Nonlinear Friction," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 2, pp. 1123–1133, 2012.

[80] T. Orlowska-Kowalska and K. Szabat, "Control of the Drive System With Stiff and Elastic Couplings Using Adaptive Neuro-Fuzzy Approach," *IEEE Transactions on Industrial Electronics*, vol. 54, no. 1, pp. 228–240, 2007.

[81] K. Salahshoor, M. Kordestani, and M. S. Khoshro, "Fault detection and diagnosis of an industrial steam turbine using fusion of SVM (support vector machine) and ANFIS (adaptive neuro-fuzzy inference system) classifiers," *Energy*, vol. 35, no. 12, pp. 5472–5482, 2010.

[82] H. A. Talebi, K. Khorasani, and S. Tafazoli, "A Recurrent Neural-Network-Based Sensor and Actuator Fault Detection and Isolation for Nonlinear Systems With Application to the Satellite's Attitude Control Subsystem," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 45–60, 2009.

[83] M. Wu, J.-H. She, and M. Nakano, "An expert control system using neural networks for the electrolytic process in zinc hydrometallurgy," *Engineering Applications of Artificial Intelligence*, vol. 14, no. 5, pp. 589–598, 2001.

[84] R. Braun and R. Manning, "Mars exploration entry, descent and landing challenges," in *2006 IEEE Aerospace Conference.* IEEE, 2006, p. 18.

[85] M. B. Quadrelli, L. J. Wood, J. E. Riedel, M. C. McHenry, M. Aung, L. A. Cangahuala, R. A. Volpe, P. M. Beauchamp, and J. A. Cutts, "Guidance, Navigation, and Control Technology Assessment for Future Planetary Science Missions," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 7, pp. 1165–1186, 2015.

[86] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning," 2016, arXiv:1611.02779. [Online]. Available: https://arxiv.org/abs/1611.02779

[87] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning That Matters," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[88] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011.

[89] Z.-y. Song, C. Wang, S. Theil, D. Seelbinder, M. Sagliano, X.-f. Liu, and Z.-j. Shao, "Survey of autonomous guidance methods for powered planetary landing," *Frontiers of Information Technology & Electronic Engineering*, vol. 21, no. 5, pp. 652–674, 2020.

[90] A. R. Klumpp, "Apollo lunar descent guidance," *Automatica*, vol. 10, no. 2, pp. 133–146, 1974.

[91] P. Lu, "Augmented Apollo Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 447–457, 2019.

[92] ——, "Theory of Fractional-Polynomial Powered Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 3, pp. 398–409, 2020.

[93] Y. Guo, M. Hawkins, and B. Wie, "Applications of Generalized Zero-Effort-Miss/Zero-Effort-Velocity Feedback Guidance Algorithm," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 3, pp. 810–820, 2013.

[94] ——, "Waypoint-Optimized Zero-Effort-Miss/Zero-Effort-Velocity Feedback Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 3, pp. 799–809, 2013.

[95] Y. Zhang, Y. Guo, G. Ma, and T. Zeng, "Collision avoidance ZEM/ZEV optimal feedback guidance for powered descent phase of landing on Mars," *Advances in Space Research*, vol. 59, no. 6, pp. 1514–1525, 2017.

[96] Y. Guo, M. Hawkins, and B. Wie, "Optimal feedback guidance algorithms for planetary landing and asteroid intercept," in *AAS/AIAA astrodynamics specialist conference.* AAS, 2011, p. 588.

[97] J. Meditch, "On the problem of optimal thrust programming for a lunar soft landing," *IEEE Transactions on Automatic Control*, vol. 9, no. 4, pp. 477–484, 1964.

[98] U. Topcu, J. Casoliva, and K. D. Mease, "Minimum-Fuel Powered Descent for Mars Pinpoint Landing," *Journal of Spacecraft and Rockets*, vol. 44, no. 2, pp. 324–331, 2007.

[99] S. P. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, 2004.

[100] B. Açikmeşe and S. R. Ploen, "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007.

[101] L. Blackmore, B. Açikmeşe, and D. P. Scharf, "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 4, pp. 1161–1171, 2010.

[102] B. Açikmeşe, J. M. Carson, and L. Blackmore, "Lossless Convexification of Non-convex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, 2013.

[103] X. Liu, Z. Shen, and P. Lu, "Entry Trajectory Optimization by Second-Order Cone Programming," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 2, pp. 227–241, 2016.

[104] D. Dueri, B. Açikmeşe, D. P. Scharf, and M. W. Harris, "Customized Real-Time Interior-Point Methods for Onboard Powered-Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 197–212, 2017.

[105] D. P. Scharf, B. Açikmeşe, D. Dueri, J. Benito, and J. Casoliva, "Implementation and Experimental Demonstration of Onboard Powered-Descent Guidance," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 213–229, 2017.

[106] M. Sagliano, "Pseudospectral Convex Optimization for Powered Descent and Landing," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 2, pp. 320–334, 2018.

[107] M. Szmuk, B. Açikmeşe, and A. W. Berning, "Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints," in *AIAA Guidance, Navigation, and Control Conference.* American Institute of Aeronautics and Astronautics, 2016.

[108] Y. Mao, M. Szmuk, and B. Açikmeşe, "Successive convexification of non-convex optimal control problems and its convergence properties," in *IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 3636–3641.

[109] M. Szmuk, T. P. Reynolds, and B. Açıkmeşe, "Successive Convexification for Real-Time Six-Degree-of-Freedom Powered Descent Guidance with State-Triggered Constraints," *Journal of Guidance, Control, and Dynamics*, vol. 43, no. 8, pp. 1399–1413, 2020.

[110] U. Lee and M. Mesbahi, "Constrained Autonomous Precision Landing via Dual Quaternions and Model Predictive Control," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 2, pp. 292–308, 2017.

[111] J. Ridderhof and P. Tsiotras, "Uncertainty Quantication and Control During Mars Powered Descent and Landing using Covariance Steering," in *2018 AIAA Guidance, Navigation, and Control Conference*, 2018.

[112] ——, "Minimum-fuel Powered Descent in the Presence of Random Disturbances," in *AIAA SciTech 2019 Forum*, 2019.

[113] A. Zavoli and L. Federici, "Reinforcement Learning for Robust Trajectory Design of Interplanetary Missions," *Journal of Guidance Control and Dynamics*, vol. 44, no. 8, pp. 1440–1453, 2021.

[114] C. J. Sullivan and N. Bosanac, "Using Reinforcement Learning to Design a Low-Thrust Approach into a Periodic Orbit in a Multi-Body System," in *AIAA SciTech 2020 Forum*, 2020.

[115] N. B. LaFarge, D. H. Miller, K. C. Howell, and R. Linares, "Autonomous closed-loop guidance using reinforcement learning in a low-thrust, multi-body dynamical environment," *Acta Astronautica*, vol. 186, pp. 1–23, 2021.

[116] S. Boone, S. Bonasera, J. W. McMahon, N. Bosanac, and N. R. Ahmed, "Incorporating Observation Uncertainty into Reinforcement Learning-Based Spacecraft Guidance Schemes," in *AIAA SciTech 2022 Forum*, 2022.

[117] L. Federici, A. Scorsoglio, A. Zavoli, and R. Furfaro, "Autonomous Guidance Between Quasiperiodic Orbits in Cislunar Space via Deep Reinforcement Learning," *Journal of Spacecraft and Rockets*, vol. 60, no. 6, pp. 1954–1965, 2023.

[118] K. Hovell and S. Ulrich, "Deep Reinforcement Learning for Spacecraft Proximity Operations Guidance," *Journal of Spacecraft and Rockets*, vol. 58, no. 2, pp. 254–264, 2021.

[119] H. Holt, R. Armellin, A. Scorsoglio, and R. Furfaro, "Low-Thrust Trajectory Design Using Closed-Loop Feedback-Driven Control Laws and State-Dependent Parameters," in *AIAA SciTech 2020 Forum*, 2020.

[120] H. Holt, R. Armellin, N. Baresi, Y. Hashida, A. Turconi, A. Scorsoglio, and R. Furfaro, "Optimal Q-laws via reinforcement learning with guaranteed stability," *Acta Astronautica*, vol. 187, pp. 511–528, 2021.

[121] A. Scorsoglio, R. Furfaro, R. Linares, and M. Massari, "Relative motion guidance for near-rectilinear lunar orbits with path constraints via actor-critic reinforcement learning," *Advances in Space Research*, vol. 71, no. 1, pp. 316–335, 2023.

[122] E. M. Zucchelli, D. Wu, J. Briden, C. Hofmann, V. Rodriguez-Fernandez, and R. Linares, "Fine-tuned language models as space systems controllers," in *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference*, 2024.

[123] A. Carrasco, V. Rodriguez-Fernandez, and R. Linares, "Large language models as autonomous spacecraft operators in kerbal space program," *Advances in Space Research*, vol. 76, no. 6, pp. 3480–3497, 2025.

[124] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," 2016, arXiv:1606.01540. [Online]. Available: https://arxiv.org/abs/1606.01540

[125] M. Towers, A. Kwiatkowski, J. K. Terry, J. U. Balis, G. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. J. S. Tan, and O. G. Younis, "Gymnasium:

A Standard Interface for Reinforcement Learning Environments," Jun. 2025. [Online]. Available: https://zenodo.org/records/15753105

[126] T. Navarro, A. Stroescu, D. Izzo, S. Galvez Rojas, and F. Lopez Valverde, "Decision making for planetary landing applications using ai agents and reinforcement learning," in *Proceedings of SPAICE2024: The First Joint European Space Agency/IAA Conference on AI in and for Space*, 2024, pp. 128–133.

[127] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[128] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *Journal of Machine Learning Research*, vol. 18, no. 185, pp. 1–52, 2018.

[129] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems*, vol. 25. Curran Associates, Inc., 2012.

[130] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential Model-Based Optimization for General Algorithm Configuration," in *Learning and Intelligent Optimization*. Springer Berlin Heidelberg, 2011, pp. 507–523.

[131] J. Bergstra, D. Yamins, and D. Cox, "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 28, no. 1. PMLR, 2013, pp. 115–123.

[132] TensorFlow Developers, "TensorFlow," Aug. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.16852354

[133] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., 2000, pp. 663–670.

[134] C. Sánchez-Sánchez and D. Izzo, "Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems," *Journal of Guidance, Control, and Dynamics*, vol. 41, no. 5, pp. 1122–1135, 2018.

[135] S. You, C. Wan, R. Dai, and J. R. Rea, "Learning-Based Onboard Guidance for Fuel-Optimal Powered Descent," *Journal of Guidance, Control, and Dynamics*, vol. 44, no. 3, pp. 601–613, 2021.

[136] J. Briden, T. Gurga, B. J. Johnson, A. Cauligi, and R. Linares, "Improving Computational Efficiency for Powered Descent Guidance via Transformer-based Tight Constraint Prediction," in *AIAA SciTech 2024 Forum*, 2024, p. 1760.

[137] L. Cheng, Z. Wang, Y. Song, and F. Jiang, "Real-time optimal control for irregular asteroid landings using deep neural networks," *Acta Astronautica*, vol. 170, pp. 66–79, 2020.

[138] L. Cheng, Z. Wang, F. Jiang, and J. Li, "Fast Generation of Optimal Asteroid Landing Trajectories Using Deep Neural Networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 56, no. 4, pp. 2642–2655, 2020.

[139] L. Federici, B. Benedikter, and A. Zavoli, "Deep Learning Techniques for Autonomous Spacecraft Guidance During Proximity Operations," *Journal of Spacecraft and Rockets*, vol. 58, no. 6, pp. 1774–1785, 2021.

[140] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming Exploration in Reinforcement Learning with Demonstrations," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6292–6299.

[141] S. Ross, G. Gordon, and D. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15. PMLR, 2011, pp. 627–635.

[142] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. PMLR, 2017, pp. 3309–3318.

[143] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Leibo, and A. Gruslys, "Deep Q-learning From Demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[144] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards," 2018, arXiv:1707.08817. [Online]. Available: https://arxiv.org/abs/1707.08817

[145] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations," 2018, arXiv:1709.10087. [Online]. Available: https://arxiv.org/abs/1709.10087

[146] A. Rubinsztejn, K. Bryan, R. Sood, and F. E. Laipert, "Using Reinforcement Learning to Design Missed Thrust Resilient Trajectories," in *2020 AAS/AIAA Astrodynamics Specialist Conference*, 2020.

[147] R. Vinter, "Optimal Control and Pontryagin's Maximum Principle," in *Encyclopedia of Systems and Control.* Springer, 2021, pp. 1578–1584.

[148] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight Experience Replay," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[149] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned Language Models are Zero-Shot Learners," in *International Conference on Learning Representations*, 2022.

[150] Lockheed Martin, "Lockheed Martin and University of Southern California Build Smart CubeSats, La Jument," Aug. 2020. [Online]. Available: https://news.lockheedmartin.com/news-releases?item=128962

[151] T. Pultarova, "SpaceX to launch 1st space-hardened Nvidia AI GPU on upcoming rideshare mission," *Space.com*, Aug. 2024. [Online]. Available: https://www.space.com/ai-nvidia-gpu-spacex-launch-transporter-11

[152] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[153] F. Zenke, B. Poole, and S. Ganguli, "Continual Learning Through Synaptic Intelligence," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. PMLR, 2017, pp. 3987–3995.

[154] D. Lopez-Paz and M. A. Ranzato, "Gradient Episodic Memory for Continual Learning," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[155] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual Learning with Deep Generative Replay," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[156] J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell, "Progress & Compress: A scalable framework for continual learning," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 2018, pp. 4528–4537.

[157] R. S. Sutton, A. Koop, and D. Silver, "On the role of tracking in stationary environments," in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 871–878.

[158] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable Deep Rein-

forcement Learning for Vision-Based Robotic Manipulation," in *Proceedings of The 2nd Conference on Robot Learning*, vol. 87. PMLR, 2018, pp. 651–673.

[159] T. Xie, N. Jiang, H. Wang, C. Xiong, and Y. Bai, "Policy Finetuning: Bridging Sample-Efficient Offline and Online Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 27 395–27 407.

[160] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-Learning for Offline Reinforcement Learning," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 1179–1191.

[161] M. Nakamoto, S. Zhai, A. Singh, M. Sobol Mark, Y. Ma, C. Finn, A. Kumar, and S. Levine, "Cal-QL: Calibrated Offline RL Pre-Training for Efficient Online Fine-Tuning," in *Advances in Neural Information Processing Systems*, vol. 36. Curran Associates, Inc., 2023, pp. 62 244–62 269.

[162] H. Ju, R. Juan, R. Gomez, K. Nakamura, and G. Li, "Transferring policy of deep reinforcement learning from simulation to reality for robotics," *Nature Machine Intelligence*, vol. 4, no. 12, pp. 1077–1087, 2022.

[163] A. Scorsoglio, A. D'Ambrosio, and R. Furfaro, "Stability Certification of Reinforcement Learning-Based Control for Aerospace Applications," in *2025 AAS/AIAA Space Flight Mechanics Meeting*, 2025.

[164] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-Learning in Neural Networks: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5149–5169, 2022.

[165] C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *Proceedings of the 34th International Conference on Machine Learning*. PMLR, 2017, pp. 1126–1135.

[166] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," 2017, arXiv:1611.05763. [Online]. Available: https://arxiv.org/abs/1611.05763

[167] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments," 2018, arXiv:1710.03641. [Online]. Available: https://arxiv.org/abs/1710.03641

[168] B. Gaudet, R. Furfaro, R. Linares, and A. Scorsoglio, "Reinforcement Metalearning for Interception of Maneuvering Exoatmospheric Targets with Parasitic Attitude Loop," *Journal of Spacecraft and Rockets*, vol. 58, no. 2, pp. 386–399, 2021.

[169] L. Federici, A. Scorsoglio, A. Zavoli, and R. Furfaro, "Meta-reinforcement learning for adaptive spacecraft guidance during finite-thrust rendezvous missions," *Acta Astronautica*, vol. 201, pp. 129–141, 2022.

[170] L. Federici and A. Zavoli, "Robust interplanetary trajectory design under multiple uncertainties via meta-reinforcement learning," *Acta Astronautica*, vol. 214, pp. 147–158, 2024.

[171] G. Fereoli, H. Schaub, and P. Di Lizia, "Meta-Reinforcement Learning for Spacecraft Proximity Operations Guidance and Control in Cislunar Space," *Journal of Spacecraft and Rockets*, vol. 62, no. 3, pp. 706–718, 2025.

[172] A. Scorsoglio, A. D'Ambrosio, L. Ghilardi, B. Gaudet, F. Curti, and R. Furfaro, "Image-Based Deep Reinforcement Meta-Learning for Autonomous Lunar Landing," *Journal of Spacecraft and Rockets*, vol. 59, no. 1, pp. 153–165, 2022.

[173] B. Gaudet, R. Linares, and R. Furfaro, "Terminal adaptive guidance via reinforcement meta-learning: Applications to autonomous asteroid close-proximity operations," *Acta Astronautica*, vol. 171, pp. 1–13, 2020.

[174] ——, "Six degree-of-freedom body-fixed hovering over unmapped asteroids via LIDAR altimetry and reinforcement meta-learning," *Acta Astronautica*, vol. 172, pp. 90–99, 2020.

[175] L. Federici, A. Scorsoglio, L. Ghilardi, A. D'Ambrosio, B. Benedikter, A. Zavoli, and R. Furfaro, "Image-Based Meta-Reinforcement Learning for Autonomous Guidance of an Asteroid Impactor," *Journal of Guidance, Control, and Dynamics*, vol. 45, no. 11, pp. 2013–2028, 2022.

[176] B. Kouvaritakis and M. Cannon, *Model Predictive Control: Classical, Robust and Stochastic*. Springer International Publishing, 2015.

[177] H. Hellendoorn and D. Driankov, *Fuzzy Model Identification: Selected Approaches*. Springer Science & Business Media, 2012.

[178] V. A. Akpan and G. D. Hassapis, "Nonlinear model identification and adaptive model predictive control using neural networks," *ISA Transactions*, vol. 50, no. 2, pp. 177–194, 2011.

[179] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "Model-based Reinforcement Learning: A Survey," *Foundations and Trends® in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023.

[180] Z. Rimon, A. Tamar, and G. Adler, "Meta reinforcement learning with finite training tasks - a density estimation approach," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates Inc., 2022, pp. 13 640–13 653.

[181] B. C. da Silva, E. W. Basso, A. L. C. Bazzan, and P. M. Engel, "Dealing with non-stationary environments using context detection," in *Proceedings of the 23rd International Conference on Machine Learning*. Association for Computing Machinery, 2006, pp. 217–224.

[182] J. Wang, S. Lu, S.-H. Wang, and Y.-D. Zhang, "A review on extreme learning machine," *Multimedia Tools and Applications*, vol. 81, no. 29, pp. 41 611–41 660, 2022.

[183] J. M. Lopez-Guede, B. Fernandez-Gauna, and M. Graña, "State-action Value Function Modeled by ELM in Reinforcement Learning for Hose Control Problems," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 21, no. 2, pp. 99–116, 2013.

[184] T. Sun, B. He, R. Nian, and T. Yan, "Target following for an autonomous underwater vehicle using regularized ELM-based reinforcement learning," in *OCEANS 2015 - MTS/IEEE Washington*, 2015, pp. 1–5.

[185] P. Escandell-Montero, J. M. Martínez-Martínez, J. D. Martín-Guerrero, E. Soria-Olivas, and J. Gómez-Sanchis, "Least-squares temporal difference learning based on an extreme learning machine," *Neurocomputing*, vol. 141, pp. 37–45, 2014.

[186] D. Li, L. Li, T. Song, and Q. Jin, "Least-squares temporal difference learning with eligibility traces based on regularized extreme learning machine," in *2016 Chinese Control and Decision Conference (CCDC)*, 2016, pp. 6976–6981.

[187] J. MSV, "NVIDIA Ups The Ante On Edge AI With Jetson AGX Orin," *Forbes*, 2022. [Online]. Available: https://www.forbes.com/sites/janakirammsv/2022/03/27/nvidia-ups-the-ante-on-edge-ai-with-jetson-agx-orin/

[188] J. Wu, "Edge AI Is The Future, Intel And Udacity Are Teaming Up To Train Developers," *Forbes*, Apr. 2020. [Online]. Available: https://www.forbes.com/sites/cognitiveworld/2020/04/16/edge-ai-is-the-future-intel-and-udacity-are-teaming-up-to-train-developers/

[189] L. P. García, G. Furano, M. Ghiglione, V. Zancan, E. Imbembo, C. Ilioudis, C. Clemente, and P. Trucco, "Advancements in Onboard Processing of Synthetic Aperture Radar (SAR) Data: Enhancing Efficiency and Real-Time Capabilities," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 17, pp. 16 625–16 645, 2024.

[190] N. Ghasemi, J. A. Justo, M. Celesti, L. Despoisse, and J. Nieke, "Onboard Processing of Hyperspectral Imagery: Deep Learning Advancements, Methodologies,

Challenges, and Emerging Trends," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 18, pp. 4780–4790, 2025.

[191] G. Giuffrida, L. Fanucci, G. Meoni, M. Batič, L. Buckley, A. Dunne, C. van Dijk, M. Esposito, J. Hefele, N. Vercruyssen, G. Furano, M. Pastena, and J. Aschbacher, "The Φ-Sat-1 Mission: The First On-Board Deep Neural Network Demonstrator for Satellite Earth Observation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2022.

[192] G. Mateo-Garcia, J. Veitch-Michaelis, L. Smith, S. V. Oprea, G. Schumann, Y. Gal, A. G. Baydin, and D. Backes, "Towards global flood mapping onboard low cost satellites with machine learning," *Scientific Reports*, vol. 11, no. 1, p. 7249, 2021.

[193] J. Ridderhof and P. Tsiotras, "Minimum-Fuel Closed-Loop Powered Descent Guidance with Stochastically Derived Throttle Margins," *Journal of Guidance, Control, and Dynamics*, vol. 44, no. 3, pp. 537–547, 2021.