Evaluating policy learning methods on the powered descent rocket landing problem.

Learning feasible policies for powered descent.

MSc Thesis

Jonathan van Zyl

Delft University of Technology

TU Delft

Evaluating policy learning methods on the powered descent rocket landing problem.

Learning feasible policies for powered descent.

by

Jonathan van Zyl

Cover Image Source:	Generated using DALL-E.
Faculty†:	Aerospace Engineering.
Supervisors:	Dr.ir. E van Kampen, Associate professor, Control & Simulation†.
Thesis Defence Committee Chair:	Dr. X Wang, Assistant professor, Control & Simulation†.
Thesis Defence External Examiner:	Dr.ir. E Mooij, Associate Professor, Space Engineering†.
Project Duration:	November 2024 - July 2025

TU Delft

Preface

To make space more accessible, launch costs must be reduced. Consequently, companies are moving towards reusable launch vehicles. Consider, for example, how impractical it would be to scrap an aircraft after every flight—flying would become prohibitively expensive. However, research into applying reinforcement learning to address this issue remains limited, hence the motivation for this thesis.

I would like to thank my supervisor, Dr. ir. Erik-Jan van Kampen for his support and advice, without which I would have been lost. Also, I would like to extend my thanks to all the lecturers who have taught me.

Thank you to my fellow Aerospace Engineering students for helping me along the way, from studying for exams to guidance. In particular, Alexander Kiselev, Andrea Speculari, Antonio Minafra, Anton Hendrickx, Kerem Er, Pierluigi Rinaldi Meneses, and Ugo Coveliers-Munzi. To Aral de Moor and Dani Rogmans, from Computer Science. To Juan Luca Kemps from Mechanical Engineering and Maria Brosens from Chemical Engineering for the late-night library work sessions. And finally, thank you to my parents, Robert and Elizabeth van Zyl, for inspiring me to become an engineer.

Jonathan van Zyl
Delft, June 2025

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
ACS	Aerodynamic Control Surfaces
AFLS	Away From Launch Site
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CoG	Centre of Gravity
CoP	Centre of Pressure
CPSO	Cooperative Particle Swarm Optimisation
CVXGEN	A code generator for convex optimisation
DDPG	Deep Deterministic Policy Gradient
DGPS	Differential Global Positioning System
DMPO	Distributed MPO
DoF	Degree(s)-of-Freedom
DP	Dynamic Programming
DQN	Deep Q-learning
DRL	Deep Reinforcement Learning
D4PG	Distributed Distributional Determinsitic Deep Policy Gradient
EA	Evolutionary Algorithms
ES	Evolutionary Strategies
FADS	Flush Air Data System
GA	Genetic Algorithm
GAIL	Generative Adversarial Imitation Learning
GNC	Guidance, Navigation and Control
GNSS	Global Navigation Satellite System
HER	Hindsight Experience Replay
ICM	Intrinisic Curiosity Module
IGA	Island Genetic Algorithm
IMU	Inertial Measurement Unit
ISA	International Standard Atmopsher
JIT	Just-In-Time
KL	Kullback-Leibler
LEO	Low-Earth Orbit
LQR	Liner Quadratic Regulator
LVLH	Local Vertical-Local Horizontal
MDP	Markov Decision Process
MOOP	Multiple-Objection Optimisation Problem
MPC	Model Predictive Control
MPO	Maximum a posteriori Policy Optimisation
MSE	Mean-Squared Error
MVM	Main Valve Management
NE	Neuroevolution
NSGA-II	Non Sorting Genetic Algorithm - II
OU	Ornstein-Uhlenbeck
PD	Proportional Derivative
PER	Prioritised Experience Replay
PID	Proportional-Integral-Derivative

Abbreviation	Definition
PPO	Proximal Policy Optimisation
PSN	Parameter Space Noise
PSO	Particle Swarm Optimisation
RCS	Reaction Control System
RL	Reinforcement Learning
RND	Random Network Distillation
RQ	Research Question
SARSA	State-Action-Reward-State-Action
SAC	Soft Actor-Critic
SGD	Stochastic Gradient Descent
RTLS	Return To Launch Site
SCvx	Successive Convexification
SI	Swarming Intelligence
SLSQP	Sequential Least Squares Programming
SOCP	Second Order Cone Programming
TD	Temporal Difference
TD3	Twin Delayed Deep Deterministic Policy Gradient
TRPO	Trust Region Policy Optimisation
TWR	Thrust-to-Weight Ratio
TVC	Thrust Vector Control
WBS	Work Breakdown Structure
WP	Work Package

Symbols, diacritical marks and operators

Modelling

Symbols

Symbol	Definition
a	Semi-major axis
A_e	Nozzle exit area
C_D	Drag coefficient
C_L	Lift coefficient
d	Distance
D	Drag
f	Fill level
F	Force
g	Gravitational acceleration
g_0	Gravitational acceleration at sea-level
I	Moment of inertia
I_{sp}	Specific impulse
J	Cost
L	Lift force
m	Mass
m_p	Propellant mass
m_{pay}	Payload mass
m_s	Structural mass
M	Mach number
M_z	Pitch moment
$\dot{m}$	Mass flow
n^e	Number of engines
p_a	Atmospheric pressure
p_e	Nozzle exit pressure
q	Dynamic pressure

Symbol	Definition
Q	MPC's cost weight matrix
r	Radius
r_e	Radius of Earth
S	Aerodynamic reference area
R	MPC's control effort weight matrix
t	Time
t_{fairing}	Fairing wall thickness
T	Thrust
TWR	Thrust-to-Weight ratio
v_{ex}	Exhaust velocity
V	Speed
x	East position from launch site
y	Altitude from launch site
y_t	Terminal altitude
z	Position along meridian from launch site
α	Angle of attack
β	Side-slip angle
γ	Flight path angle
δ	Deflection
ΔT_p	Pressure losses
Δv	Velocity increment
ϵ	Structural coefficient
θ	Pitch angle
θ^g	Gimbal pitch angle
θ^{gf}	Grid fin pitch angle
κ	Lagrangian multiplier
Λ_{ul}	Factor of upper and lower section mass
μ	Gravitational constant for Earth
ρ	(Air) density
τ	Throttle
ψ^g	Gimbal roll angle
ψ^{gf}	Grid fin roll angle
ω	Angular velocity

Common diacritical marks

Mark	Definition
x_a	Ascent phase or axial force when after F like F_a
x_d	Descent phase
x_n	Normal
x^*	Optimal value
x^{l^*}	Optimal value considering losses

Reinforcement learning

Symbols

Symbol	Definition
a	Action
$A(s,a)$	Advantage stream for duelling Q-networks
$\hat{A}$	Advantage estimate (PPO)
d	Done
$g(\eta)$	Dual objective function used to optimise the MPO temperature parameter η

Symbol	Definition
G	N-step returns, or Monte Carlo returns
$\mathcal{H}_{\text{target}}$	Target entropy used in SAC
I	Identity matrix
$\mathcal{L}$	Loss
M	Threshold
N	Number of samples in replay buffer
N_b	Batch size
$\mathcal{N}$	Normal distribution
p	Priority, used in PER buffer
P	Probability
$Q(s, a)$	Action-value function
r	Importance sampling correction (PPO)
$R(s, a)$	Reward
$\mathcal{R}$	Replay buffer
s	State
$V_d(s)$	Value stream for duelling Q-networks
$V\ w$	Weight
y	Output
α	Learning rate
α_{per}	Degree of prioritisation in PER buffer
β	Importance sampling correction used in a PER buffer
β^{PPO}	KL penalty coefficient in the PPO objective; scales the KL divergence between the old and new policies to constrain policy updates
γ	Discount factor
δ	Temporal difference error
Δt	Time step
ϵ	Noise
$\zeta(\cdot)$	Actor network
$\zeta_{\text{ref}}(s)$	Reference policy derived from exponentiated Q-values during the MPO E-step; used as target for KL projection onto the parametric actor
η_{ICM}	Intrinsic reward scaling factor in a ICM
η	Temperature parameter controlling the sharpness of the softmax weighting in MPO; minimised via dual optimisation
θ	Parameters
κ^{η}	KL divergence constraint in the MPO dual formulation; controls weight variance
λ	Lagrange multiplier scaling the KL regularisation term in the dual objective
μ	Mean
ν	Temperature parameter used in SAC
σ	Standard deviation
τ	Polyak averaging coefficient
$\phi(x)$	Encoded feature of x

Common diacritical marks

Mark	Definition
x_t	Value at time t
x^A	Advantage network, used in Dueling Q-networks
x^F	Forward network of a ICM
x^I	Inverse network of a ICM
x^Q	Critic network
x_S	Gaussian network
x^{V_d}	Value network, used in Dueling Q-networks
$x^{(y)}$	Value at y number of steps, used in N-step retuns

Mark	Definition
x^{ζ}	Actor network
x^{-}	Target value
x^{*}	Optimal value
$\bar{x}$	Normalised variable
$\tilde{x}$	Noise perturbed parameter
$\hat{x}$	Prediction

Operators

Operator	Definition
$\log P(x\|a)$	Log probability
$\|\|x\|\|^2$	L2 norm, otherwise known as Euclidean norm.
$\mathcal{D}(P\|\|Q)$	KL divergence of probability distribution P with a reference distribution Q
$f_x(y)$	Probability density function of x
$\propto$	Proportional

Evolutionary Algorithms

Here all symbols, with their diacritical marks are in a single table.

Symbol	Definition
c_1	Cognitive acceleration coefficient (attraction to personal best)
c_2	Social acceleration coefficient (attraction to neighbourhood/global best)
w	Inertia weight controlling the influence of the previous velocity
$\mathbf{v}_i$	Velocity of particle i
$\mathbf{x}_i$	Position of particle i in the search space
$\mathbf{x}_i^\star$	Personal best position found by particle i
$\mathbf{x}_s^\star$	Best position found by the swarm (or subswarm)
$\boldsymbol{\xi}_1$	Vector of independent uniform random numbers in $[0,1]$ for the cognitive term
$\boldsymbol{\xi}_2$	Vector of independent uniform random numbers in $[0,1]$ for the social term
$\circ$	Element-wise (Hadamard) product

1 Introduction

1.1. Reusable Launch Vehicle Landing and Staging

In this section, a comprehensive overview of the historical evolution of vertical spacecraft landings, culminating in reusable launch vehicles, is provided in Subsection 1.1.1, highlighting the control methods used in industry, such as lossless convex optimisation, and proposals for replacing this with Reinforcement Learning (RL). Then, Subsection 1.1.2 delves into the optimal staging procedure applied to reusable launch vehicles and how it requires guidance and control systems to generate velocity increment inputs for the procedure. Finally, Subsection 1.1.3 discusses policy optimisation methods of RL and NeuroEvolution (NE) with a concise description of the history of these methods.

1.1.1. Autonomous Landing of Reusable Rocket Stages

Autonomous vertical space landings attempts began in 1965 with the USSR's Luna 5 mission to the Moon, but a failure within its guidance system caused a crash landing. A year later, the Luna 9 mission performed a successful soft-landing on the Moon and was closely followed by the American Surveyor 1 with a 50 km landing ellipse. Another year later, Surveyor 3 landed 2.76 km from its target [1]. This paved the way for the Apollo Lunar Module's Moon landing in 1969, utilising the Apollo Guidance Computer for closed-loop control over attitude and throttle for the majority of the flight [2], with a quartic polynomial in time used for the mission profile [3], [4]. Successful soft landings on Venus and Mars followed, validating the autonomous system's capability to achieve soft landings on various planetary bodies.

The following missions, conducted between 1997 and 2011, advanced autonomous landing technology through rover missions to Mars. The Mars Pathfinder[1], Mars Exploration Rover[2], and the 2011 Curiosity rover[3] landed within 150, 35, and 10 km of the launch site [5], showing a gradual increase in landing accuracy as technology progressed. However, landing stages of launch vehicles back on Earth require much greater precision, as the landing pad constrains its available size. For instance, an accuracy of 10 or 30 metres is the minimal requirement for returning to the launch site or landing on a drone ship downrange [6], respectively.

The Falcon 9[4], successfully landed by SpaceX in 2015, marked the start of a revolutionary shift in launch vehicle design, transitioning from traditional expendable stages to reusable ones. This resulted in reduced launch costs and allowed for more economically viable space initiatives due to lower barriers to entry. Not only does cost efficiency boost private industry, but it also allows public bodies like NASA to consider planetary exploration seriously.

For the first stage's descent, the stage begins with a flip-over manoeuvre to orient the stage horizontally towards its landing site. Then, a boostback burn provides the stage with the horizontal velocity

[1] https://science.nasa.gov/mission/mars-pathfinder/. Accessed: 18/06/2025.
[2] https://science.nasa.gov/mission/mars-exploration-rovers-spirit-and-opportunity/. Accessed: 18/06/2025.
[3] https://science.nasa.gov/mission/msl-curiosity/. Accessed: 18/06/2025.
[4] https://www.spacex.com/vehicles/falcon-9/. Accessed: 18/06/2025.

increment required to reach the landing pad. A ballistic arc follows this and, finally, a powered-descent phase [7]. The powered-descent phase presents a challenging problem in terms of trajectory optimisation, as failure to meet the landing site can result in a crash, which, in the case of a human mission, could lead to astronaut fatalities, necessitating a robust solution. In particular, a leading approach is convex optimisation, which works by applying lossless convexification to nonconvex constraints, allowing for the finding of an optimal solution in real-time [5], [6]. The resulting output is a near-fuel-optimal trajectory that satisfies the constraints. The Falcon 9 utilises CVXGEN [5] [8] to generate fast code for quadratic program representable convex optimisation problems to provide a convex-optimisation solution in provably bounded solve-times, allowing for real-time trajectory alteration under uncertainty and atmospheric disturbances. Alternatively, convex Model Predictive Control (MPC) [9] utilises a finite time horizon to provide real-time bounded solutions, which can be extended to be robust to modelling uncertainties and disturbances [10].

Behçet Açıkmeşe has co-authored several foundational papers on applying convex optimisation to the powered descent guidance problem, particularly considering the Mars landing scenario. Convex optimisation can provide guarantees of global optimum convergence within real-time computing constraints. His work demonstrates that the inherently nonconvex trajectory optimisation problem subject to state constraints can be transformed into a convex problem, specifically a Second-Order Cone Programming (SOCP) program. This allows it to be solved in polynomial time, making it scalable with problem size growth, and to be solved ”very efficiently” through primal-dual interior-point algorithms [3], providing a solution to guidance determination that can occur onboard. Pontryagin’s maximum principle was used to losslessly convexify the lower bound thrust constraint. Lars Blackmore extended this work to provide a solution for when no feasible trajectory is apparent [11]. Following this, additional constraints were added, including thrust bounds and pointing [5]. These models use a lumped mass rigid body model, decoupling it from rotational dynamics, as attitude control is argued to be at a much higher bandwidth.

The Successive Convexification (SCvx) algorithm has been used by Michael Szmuk, Behçet Açıkmeşe et al.[6] on the powered descent problem, providing a different approach to applying the trust region method to non-convex constraints and nonlinear dynamics [12]. The model had a constant atmosphere, drag coefficient, centre of mass, and inertia were assumed, with no lift. Following work introduced a ”tractable aerodynamic model” that approximated the aerodynamic forces and velocity vector relationship through a spherical aerodynamic model, but still with constant aerodynamic coefficients [13]. A six-degree-of-freedom (DoF) implementation by Szmuk utilised a single gimbaled thruster, providing coupled rotational and horizontal motion.

Despite the success of SCvx, a growing research area is applying policy learning methods to the powered descent problem, motivated by the ability to execute a policy faster than optimisation solvers and to handle sensor noise and disturbances by incorporating them into the environment from which it learns or optimises a policy. Gaudet applied the Deep Reinforcement Learning (DRL) method of Proximal Policy Optimisation (PPO) to a six DoF powered descent guidance problem to guide the rocket within a five-metre landing ellipse, motivating reinforcement learning as a method to be independent to the flaws of lossless convexification or optimal control requiring independently optimised systems for guidance and control [14]. Specifically, lossless convexification research applies simplifications to the model to make it solvable, such as linearisation. Gaudet’s work included rotational motion, but with fixed inertia and centre of mass. Moreover, aerodynamics was neglected with only a normally distributed vector for wind and ”variations in atmospheric density”.

1.1.2. Optimal Staging of Reusable Launch Vehicle

A launch vehicle’s efficiency is crucial to minimise costs by increasing payload capacity. The optimal staging procedure, based on the Tsiolkovsky rocket equation, is performed to maximise payload efficiency by removing unnecessary mass during flight, thereby improving the rocket’s mass ratio—the ratio of initial to final mass—and enabling more effective propulsion over flight [15]. Early work staged the launch vehicle in the absence of velocity losses, before later work incorporated an iterative loop with trajectory optimisation to include these [16]. Jo and Ahn [17] extended this method for reusable launch vehicles, which require extra velocity increments during the descent phase.

[5] `https://ee.stanford.edu/news/2021/jan/stephen-boyd-cvxgen-guides-spacex-falcon?`. Accessed 03-06-2025.

[6] In collaboration with Blue Origin LLC, a developer of reusable launch vehicles.

A trajectory optimiser with inner loop controls generates these velocity increments in an iterative loop with the staging procedure. However, specific launch vehicle design parameters should not be neglected, including the number of engines, moment of inertia and aerodynamic reference area, which should change with each iteration. Additionally, generating the descent velocity increments requires a feasible trajectory constrained by the respective stage's aero-mechanical-thermal constraints [7] and actuator control authority.

Policy optimisation methods provide a unique solution to descent trajectory generation by directly learning strategies that satisfy constraints on each stage [18], [19], [20]. Classical control methods, such as Proportional-Integral-Derivative (PID), for actuation or guidance typically require gain retuning for each new configuration, which is often done manually or requires human inspection. In contrast, policy optimisation methods require a fixed set of hyperparameters that are kept constant throughout iterations, thereby avoiding the need for complex optimisation solvers, constraint convexification, or model linearisation, and provide a more accessible solution. Through interaction with a simulation, policies can be learnt to enable robust control and feasible descent solutions that can generate velocity increments to support the iterative staging procedure.

1.1.3. Policy Optimisation Methods

A Markov Decision Process (MDP) provides decision-making for a stochastic problem through states, actions, rewards and transition probabilities; such that the next state depends only on the current state and action [19]. The powered descent problem can be viewed as an MDP, allowing a policy to map continuous state vectors (position, velocity, and attitude) to actions (throttle, gimbal, and deflections) to maximise expected cumulative rewards under uncertainty. A Markovian problem allows policy-based methods to directly optimise the action distribution, which can generalise high-dimensional functions through the use of a neural network acting as a function approximator [18].

In the 1950s, Dynamic Programming (DP) [21] was developed by Richard Bellman. Here, a complex problem is transformed into a sequence of simpler sub-problems, recording intermediate results to update the global optimum incrementally. It is widely applicable to combinatorial optimisation problems, where the previously computed outcomes affect the decisions at later stages. The Bellman equation of DP finds the optimal value of each state recursively as the sum of the maximum expected immediate reward and the discounted value of subsequent states. DP applied to MDPs constituted the first examples of RL, first through the value-based Temporal Difference (TD) algorithm in 1988 [22]. Here, the value function estimates the expected cumulative reward from a particular state. The functions are fitted by minimising the error between the predicted value and a bootstrapped Bellman value, known as the TD error. Later, in 1989, Q-learning was introduced, which moved from a state-value function to a state-action value function, predicting the expected cumulative reward from taking a particular action in a state [23]. Secondly, DP suffers from the need to cover the entire search space to find an optimal result, making it infeasible for large state spaces. As a result, Q-learning stores state-action values in a Q-table, where the agent selects actions to maximise expected cumulative rewards, guiding it to an optimal solution more quickly.

Q-learning provides discrete actions; transforming to continuous actions would lead to an infeasibly large action dimension. As such, the direct policy method of REINFORCE was introduced by Ronald Williams in 1992 [24]. This Monte Carlo method, which relies on transitions — sequences of states, actions, and rewards obtained from environment interaction — updates policy parameters through gradient ascent on the log-policy, with each action's gradient scaled by the trajectory's cumulative return.

Q-learning suffers from the *curse of dimensionality*, where the neural network becomes increasingly more complicated to fit as the number of features (inputs or outputs) increases. As a result, Deep Q-learning (DQN) was introduced by Volodymyr Mnih and his team at DeepMind in 2013 [18], replacing the Q-table with a neural network that serves as a function approximator and adding an experience buffer to enable the resampling of transitions. Mnih's work created an agent capable of achieving human-level performance on Atari games from learnt policies based on pixel data. This combination of neural networks and RL ushered in a new era in RL, focusing on DRL. Early work on Actor-Critic methods, which combined value-based and direct policy search methods [25] through the separation of action selection and value function estimation, were then extended to include neural network function approx-

imators through Trust Region Policy Optimisation (TRPO) in 2015 [26], and Timothy Lillicrap and his team at DeepMind introduced the Deep Deterministic Policy Gradient (DDPG) method in 2015; applied to continuous action spaces on Atari games, breaking DQN's curse of dimensionality on continuous problems.

Schulman introduced PPO as an extension to TRPO in 2017 [27], and later the Soft Actor Critic (SAC) algorithm was presented by Haarnoja et al. in 2018 [28]. SAC is an off-policy actor-critic deep RL method suitable for continuous action spaces, performing direct policy search on a stochastic policy which aims to maximise expected cumulative reward and entropy to encourage exploration through state-dependent Gaussian noise.

Neural networks in RL - a policy-gradient method - update their parameters through backpropagation [29], propagating a loss backwards through each layer via the chain rule, yielding a partial derivative for each parameter to adjust the weights to minimise the loss function. However, policy-gradient methods, when optimising a non-convex objective, can get stuck in local optima or produce no gradients in flat regions. Also, value function learning is hindered when non-informative gradients, such as those arising from a non-smooth reward function, cause high variance. This corrupts the policy's gradient updates, which leads to catastrophic forgetting and noisy learning. Additionally, problems with a long time horizon increase this variance, as distant rewards propagate further with the discount factor.

In 1950, Alan Turing conceived the first programmable computer as a "learning machine" whose development — via an initial state, education and experiential modification—he explicitly likened to evolutionary processes. Turing suggested that instead of trying to program an adult mind, it would be more effective to program a child's mind and then subject it to a process of education and experience [30]. Twenty-five years later (1975), John Holland presented an overview of the adaptation of artificial systems based on evolutionary methods [31], formalising the introduction of Genetic Algorithms (GAs) [32]–[34]. In 1994, a GA was used on a toy lander simulation to fit the weights of a neural network in a process the authors called *Neuro-Genetic control* [35], to evaluate its feasibility for trajectory planning, building on work from References [36], [37]. Neuroevolution (NE) has become the standardised term for using Evolutionary Algorithms (EAs) [38] to augment an artificial neural network's topology, parameters or to optimise learning coefficients. Although augmenting topologies can provide a more efficient network, it adds extra policy search overhead, which is unnecessary for simple problems [39].

In terms of NE as an alternative to gradient-based backpropagation methods, Salimans reviewed its performance by comparing Covariance Matrix Adaptation Evolution Strategy (CMA-ES) against modern DRL algorithms, such as TRPO and Advantage Actor-Critic (A3C), on Atari games and MuJoCo humanoid control problems [20], [40]. It was demonstrated that CMA-ES outperformed the DRL algorithms in specific environments, particularly in those with long time horizons. Additionally, it was shown to be invariant to RL's issues with reward distribution and non-informative gradients, which are caused by the use of a value function approximator. Furthermore, a GA performing neuroevolution was compared on Atari games and showed competitive performance, outperforming sceanairos where RL would get stuck in local optima, as GA is not gradient-based [41].

Swarming Intelligence (SI) is a group of algorithms inspired by the collective behaviour of animals, such as insect colonies, to explore parameter search. When used to optimise the parameters of neural networks, it can be referred to as NE or *neural swarm optimisation*. In SI, a particle represents a candidate solution; for the NE case, the parameters of a neural network are iteratively updated based on individual experiences and the success of other members within the swarm. Ant Colony Optimisation was proposed in 1996 [42], and Particle Swarm Optimisation (PSO) [43] in 1995, with each having additional variations [44], [45]. PSO has been used with Artificial Neural Networks to learn their weights and parameters, for example, on the exclusive OR (XOR) problem [43]. Comparison studies have been performed between backpropagation and PSO, showing contrasting results, with PSO converging faster or only faster on problems with a high number of local minima [46], [47]; the latter provides a more critical analysis.

1.2. Research Gaps and Project Scope

The previous section introduced background information on the autonomous rocket landing control and staging problem. Here, research gaps from this background will be identified, and the project scope

will be constrained within the limitations.

1.2.1. Research Gaps

Subsection 1.1.1 showed a need for highly accurate and precise landings for reusable launch vehicles, and that the descent procedure for a first stage landing is made up of multiple flight phases. Furthermore, current guidance and control methods have been shown to utilise trajectory optimisers like SCvx, MPC, and lossless convex optimisation, accelerated through fast code generation [7], combined with inner loop controls. The outer loop guidance simplifies to linear models by assuming fixed aerodynamic coefficients, a constant moment of inertia, and a constant atmosphere. As a result, Gaudet argued that an inner-loop controller is always required to correct for linearization errors and model fidelity reduction, culminating in the proposal of DRL through PPO as an alternative solution.

However, Gaudet's work also had many simplifications:

- Fixed inertia and centre of mass.
- A wet mass of 2000kg, much less than SuperHeavy's 250 t and Falcon 9's 25.6 t dry mass.
- Fixed thrusters with no gimballing.
- Neglected aerodynamics, and instead used a normally distributed force based on Mars' wind.
- A starting altitude of 2.5km, much lower than for Earth landing, requiring avoidance of dynamic pressure constraints [7].

To summarise, there is limited work in applying policy optimisation to the powered descent guidance problem on Earth, under representative conditions.

Subsection 1.1.2 introduced the optimal staging procedure for reusable launch vehicles, noting that it requires trajectory optimisation and parameter sizing to function effectively. Current trajectory optimisers require an outer loop to set the trajectory, followed by an inner loop that uses classical control methods, often necessitating manual gain tuning for each new staging iteration. Policy optimisation methods, on the other hand, considering PSO and RL, require only hyperparameters.

Finally, Subsection 1.1.3 introduced policy gradient and non-policy gradient methods, in the form of RL and NE, respectively. NE has been demonstrated to perform better on long-term horizons and high-variance reward landscapes [20], similar to the powered descent problem. The powered descent problem action affects the long-term success of the episode. Secondly, it will be constrained by aerothermal-mechanical constraints [7], causing a high-variance reward landscape. As such, NE may be better suited to solve the problem, but to the author's knowledge, no such analysis has been undertaken.

From the findings above, six research gaps are presented in Table 1.1.

Table 1.1: Research gaps identification.

Research Gap	Description
1.	Compare the performance of a policy optimisation method with a convex optimisation solver on the powered descent problem.
2.	Imitation of a simple convex optimisation process through DRL, followed by policy improvement.
3.	Validate a policy optimisation method's ability to generate velocity increments to perform optimal staging without needing to retune hyperparameters.
4.	Compare the performance of NE and DRL in solving the powered descent problem, in terms of convergence time, accuracy and optimality.
5.	Validate a policy optimisation method's ability to perform powered descent on Earth with higher fidelity models discussed in Subsection 1.1.1.
6.	Proposal of a low-fidelity method for sizing parameters of a launch vehicle following its staging, to facilitate incorporation with guidance and control.

[7] Falcon 9 uses CVXGEN.

1.2.2. Project Limitations

The research gaps have been identified, but they must be aligned with the project's limitations before the research objective can be defined. The limitations are as follows:

- Time: The project must be completed within 28 weeks (excluding defence), as per the MSc Thesis guidelines of the Faculty of Aerospace Engineering at TU Delft.
- Resources: Computational resources are limited to a laptop with 16 GB of RAM and a 4 GB NVIDIA Quadro P1000 GPU. There is an option to access the Delft Blue supercomputer[8] or utilise cloud computing; however, this comes with additional uncertainty and reduces flexibility.
- Existing knowledge: The author's starting knowledge is limited to the BSc and MSc Aerospace Engineering, Control and Simulation curriculum at TU Delft[9].

The limitations are analysed in Table 1.2, as applied to the research gaps proposed in Subsection 1.2.1.

Table 1.2: Analysing the limitations of answering the research gaps of Table 1.1.

Research Gap	Main limiting factor(s)
1.	Time, to implement a convex optimiser and a policy optimisation method requires two custom solutions.
2.	Resources, to generate solutions of a convex optimiser before imitating them would increase the learning-step time of a policy optimisation agent.
3.	Time, descent requires four flight phases, requiring a solution for each.
4.	Resources, to fully compare the implementation, an extensive ablation study must be performed to ensure the worst-performing solution is not due to incorrect hyperparameters.
5.	Time and existing knowledge, higher fidelity models can encompass many improvements from the inclusion of accurate aerodynamics to fuel sloshing. Generating a validated model of a launch vehicle without access to proprietary data or models is infeasible within the project timeframe.
6.	None.

Research gaps one and two require a lossless convex optimiser or MPC implementation for comparison, which introduces additional complexity and later requires additional computational resources. Research gap three requires trajectory generators and control systems for each descent phase of the flight. However, these can be simplified by optimising policy only during the final powered descent phase and utilising simple PID controllers and references for the preceding flight phases, with the ability to be automatically tuned through SI or EAs. Additionally, for research gap four, the ablation study can be performed in parallel to writing tasks.

Regarding research gap five, generating a complete launch vehicle model is infeasible; however, incorporating specific improvements is possible. For example, complex dynamics like hypersonic aerodynamics [48], aerodynamic heating, fuel sloshing [49], and launch vehicle bending [50]. Finally, research gap six has no notable limiting factors. As a result, research gaps three, four, five and six best suit the project under its limitations, with the simplifications presented in this section.

Additionally, reusable launch vehicles are generally two-stage rockets (Falcon 9, Starship[10], New Glenn[11]), with only Starship providing a reusable second stage. To reduce the scope of the study, only a reusable first stage shall be considered.

1.2.3. Research Objectives and Hypotheses

The identified research gaps of three and four aim to test the validity of using policy optimisation methods to generate velocity increments for descent, and compare the performance of gradient and non-

[8] `https://www.tudelft.nl/dhpc/system`. Accessed: 18/06/2025.
[9] `https://www.tudelft.nl/en/ae`. Accessed: 18/06/2025.
[10] `https://www.spacex.com/vehicles/starship/`. Accessed: 18/06/2025
[11] `https://www.blueorigin.com/new-glenn`. Accessed: 18/06/2025.

gradient (DRL and NE) methods applied to this problem. This leads to the following **main research objective**:

To investigate the performance of gradient and non-gradient based policy optimisation methods for first-stage landing during the powered descent phase.

The hypotheses of this study are presented below in Table 1.3.

Table 1.3: Hypotheses for the primary research objective.

Hypothesis	Description	References
1.	Reinforcement learning algorithms can successfully control the powered descent phase of a launch vehicle's first stage, achieving a stable landing.	[14]
2.	The selected neuroevolution algorithm will require fewer training episodes to learn an effective control policy for powered descent, compared to the selected reinforcement learning method, thereby demonstrating greater learning efficiency.*	[20], [41], [47]
3.	Policy optimisation methods can be effectively applied to determine velocity increments for optimal staging in reusable launch vehicles, yielding consistent performance when hyperparameters are held constant across optimisation iterations.	[17]

* - There are many RL and NE algorithms; for the purposes of this study, one model-free RL algorithm and one NE shall be chosen to be compared. Model-free RL[12] is selected, as it is comparable to Gaudet's work [14], which used PPO, and Salimans' work comparing NE to A3C and TRPO [20]. Moreover, NE shall only be conducted for policy parameter search, not for the network topology, to decrease the computational complexity of the work and maintain consistency with the work of Salimans and Such [41].

The **secondary research objective** relating to research gaps five and six is:

To develop improvements in the iterative process of optimal staging and trajectory optimisation for reusable launch vehicles, with a focus on the powered descent trajectory prediction and the estimation of key design parameters, to support more efficient and reliable launch vehicle design.

Table 1.4: Hypothesis for the secondary research objective.

Hypothesis	Description	References
4.	The developed methodology will estimate the inertia of a launch vehicle stage when provided with a specified stage mass at staging.	-

1.3. Research Questions

The research questions are formulated to address the chosen research gaps of three to six, test the hypotheses presented in Table 1.3 and Table 1.4, and are based around the two research objectives.

RQ.1 How can policy optimisation methods be applied to find effective high-level trajectory planning and mid-level guidance control strategies for the powered descent of a reusable launch vehicle's first stage?

- RQ.1.1 What policy optimisation algorithms are most suitable for performing powered descent?
- RQ.1.2 To what extent can the selected algorithms achieve a successful landing under representative powered descent scenarios on Earth?
- RQ.1.3 How do the learning efficiency and stability of the different algorithms compare in this context?

[12] The alternative is model-based RL, which is briefly introduced in Subsection 3.2.4 [51].

RQ.2: What model fidelity is required to accurately represent powered descent flight for a reusable launch vehicle's first stage as a feasibility study for policy optimisation's effectiveness?

- RQ.2.1 How can a representative initial condition for powered descent be generated based on a realistic mission profile?
- RQ.2.2 What are the key mathematical equations necessary for an adequate simulation model of powered descent?
- RQ.2.3 Which model extensions (e.g., aerodynamics, fuel sloshing) could most improve fidelity for future research?

RQ.3: Do changes in launch vehicle parameters necessitate re-tuning of hyperparameters for learning a policy to perform powered descent?

- RQ.3.1 How do the hyperparameters change the learning/optimisation stability and efficiency regarding the chosen RL algorithm?
- RQ.3.2 How can policy optimisation be incorporated into the optimal staging procedure to handle variations in launch vehicle parameters?
- RQ.3.3 How can a low-fidelity methodology estimate key physical parameters (such as inertia and actuator sizes) for a launch vehicle following a staging procedure?
- RQ.3.4 How similar are the launch vehicle parameter estimations compared to the *Starship* launch vehicle?

1.4. Report Structure

The report aims to answer the research questions presented in Section 1.3, which evaluate the effectiveness of gradient and non-gradient policy optimisation methods in performing powered descent, with the ultimate goal of integrating these methods with an optimal staging procedure for reusable launch vehicles in future work. The report begins with an academic article in Chapter 2, focused on comparing SAC and Particle SubSwarm Optimisation (PSSO) - the chosen gradient and non-gradient methods, respectively - for the powered descent problem and evaluating the SAC's sensitivity to hyperparameter changes. Following this, a literature study in Chapter 3 provides the reader with background information on rocket landing guidance, navigation and control systems, policy optimisation choices, RL, launch vehicle sizing and simulation, and finally NE through EA and SI methods.

Chapter 4 presents additional results that complement the article, including verification, validation, and the generation of an initial condition for powered descent following the outcome of the optimal staging procedure. Chapter 5 evaluates the report's findings, directly tying them to the research questions and hypotheses presented, and then details recommendations and avenues for future work. Additionally, Section A.1 presents the project planning and explains the deviations from the initial Gantt charts and Work Breakdown Structures (WBS).

2

Scientific Article

Throttle Optimisation for Powered Descent: A Comparative Study between Soft Actor Critic and Particle Subswarm Optimisation Performing Neuroevolution

J.R. van Zyl*
Delft University of Technology, Netherlands, 2628 HS

This study explored the application of particle subswarm optimisation - a non-gradient-based swarming intelligence method - in neuroevolution for solving the powered descent landing problem. It was compared to a gradient-based Soft Actor-Critic (SAC) reinforcement learning algorithm under sparse and non-sparse reward settings. A simulation environment was developed incorporating aerodynamic modelling, grid fins, thrust vector control, and variable inertia. Results show that a Soft Actor Critic struggled with non-sparse rewards due to its inability to generalise across the long-horizon powered descent task due to non-informative gradients from high-variance state-action values. In contrast, learning with sparse rewards became trapped in a local optimum due to a weak reward signal. Particle subswarm optimisation achieved rapid convergence to feasible solutions and reduced propellant consumption, outperforming SAC due to an episodic gated reward structure, local minima avoidance, and gradient independence. This study establishes particle subswarm optimisation neuroevolution as a viable alternative to reinforcement learning for the assignment of throttle during powered descent of a launch vehicle's first stage.

Nomenclature

A_e	=	Nozzle exit area $[m^2]$
d	=	Distance from rocket base $[m]$
C_A, C_D, C_L, C_N	=	Grid fin's axial force, lift, drag and grid fin's normal force coefficients
D	=	Drag force $[N]$
f^l	=	Fill level
F	=	Force $[N]$
g	=	Gravitational acceleration $[m/s^2]$
I	=	Moment of inertia $[kg \cdot m^2]$
L	=	Lift force $[N]$
m	=	Mass $[kg]$
M	=	Mach number
M_z	=	Pitch moment $[N \cdot m]$
p_a	=	Atmospheric pressure $[Pa]$
p_e	=	Nozzle exit pressure $[Pa]$
q	=	Dynamic pressure $[Pa]$
R_E	=	Radius of the Earth $[m]$
S	=	Launch vehicle's frontal area $[m^2]$
T_e	=	Engine thrust without losses $[N]$
T^g	=	Gimballed thrust $[N]$
v	=	Speed $[m/s]$
y	=	Altitude $[m]$
α	=	Angle of attack [rad]
δ	=	Grid fin deflection [rad]
γ	=	Flight path angle [rad]

*MSc Student, Control and Simulation, Faculty of Aerospace Engineering, Kluyverweg 1, Delft, 2629 HS, Netherlands.

θ = Pitch angle [rad]
θ^g = Gimbal angle (pitch) [rad]
ρ = Air density $[kg/m^3]$
τ = Throttle

Particle Swarm Optimisation Variables

c_1, c_2 = Cognitive and social coefficients
f_i = Fitness of particle i
$f_i^\star$ = Personal best fitness of particle i
$f_s^\star$ = Best fitness within subswarm s
f^{global} = Global best fitness
G = Number of generations
N = Total number of particles
p_{comm} = Probability threshold to communicate
S = Number of subswarms
w = Inertia weight
$\mathbf{v}_i$ = Velocity vector of particle i
$\mathbf{x}_i$ = Position vector of particle i
$\mathbf{x}_i^\star$ = Personal best position of particle i
$\mathbf{x}_s^\star$ = Best position within subswarm s
$\mathbf{x}^{\text{global}}$ = Best position across all subswarms
α_{comm} = Soft influence coefficient for best solution communication.
$\boldsymbol{\xi}_1, \boldsymbol{\xi}_2$ = Random vectors sampled from $\mathcal{U}(0,1)^d$
$\Pi_{\mathcal{X}}$ = Projection operator onto bounded domain $\mathcal{X}$
$\tau_{\text{comm}}, \tau_{\text{mig}}$ = Communication and migration intervals in terms of generations

Key Reinforcement Learning Variables

$\mathcal{H}_{\text{target}}$ = Target entropy
N = Buffer capacity
N_b = Batch size
N_β = Importance-sampling exponent number of annealing steps.
p_i = Priority of transition i
$P(i)$ = Sampling probability for transition i
w_k = Importance sampling weight for transition k
α_t = Prioritization exponent
$\alpha_\theta, \alpha_\phi, \alpha_\nu$ = Actor, critic and temperature learning rates
β_t = Importance-sampling exponent
θ = Actor (policy) network parameters
τ = Polyak averaging coefficient
π_θ = Policy (actor) network
ϕ_1, ϕ_2 = Critic network parameters
γ = Discount factor
ν = Temperature parameter

I. Introduction

Surveyeror 1 landed with a 50km landing ellipse on the Moon in 1966, paving the way for planetary soft-landing systems. In the 70s, the Soviet Union's Mars 3 and Venera landers achieved soft landings on Mars and Venus, respectively. From the 1990s to the 2010s, the landing accuracy on Mars increased significantly, from 150 km for the 1997 Mars Pathfinder mission to 10 km for the Curiosity rover [1]. However, greater accuracy is needed for launch pad landing to the resolution of metres or tens of metres, and with a lower touchdown speed due to the greater mass [2]. Additionally, although spacecraft have performed successful soft landings on multiple planetary bodies, including comets and asteroids, rocket landing provides a challenging task due to a much larger mass onboard and a small margin for failure.

The final phase of descent for a reusable stage is the powered descent arc, which presents a significant challenge, constrained by dynamic pressure, terminal conditions of the landing pad position and touchdown velocity, and the

desire to minimise fuel consumption for more efficient flight. Lossless convex optimisation and its extensions are a leading solution applied to this problem, as they allow optimal solutions to be found in real-time [1, 3–6]. Secondly, it has been used to minimise the final distance to the landing site when the stage is flying in a region with no feasible solution [4]. Furthermore, neural networks have been used to generate an initial guess for a convex optimiser to reduce computational demands and increase real-time feasibility [7]. Falcon 9 utilises CVXGEN* to rapidly speed up its optimisation procedure through the generation of fast code [8], removing doubts about the real-time feasibility of lossless convex optimisers.

Despite the success of convex optimisation, Deep Reinforcement Learning (DRL) has been explored as an alternative solution to guiding and controlling powered descent [9]. DRL can provide integrated guidance and control, whereas a convex optimiser would provide a solution based on a linearised and simplified model, requiring an inner loop control to minimise tracking error. Although work has been done to minimise the usage of tracking error controls through a convex Model Predictive Controller (MPC) [10], a finite-time-horizon optimisation solver. DRL can provide a solution without the need for separate systems, due to optimal guidance that takes into account non-linear model features.

Neuroevolution (NE) provides an alternative policy optimisation method to reinforcement learning. NE's non-gradient black-box optimisation nature has benefits for specific problems, such as those with long time horizons and non-smooth rewards [11]. Furthermore, it has shown competitive performance against RL on benchmark problems and outperforms RL on certain tasks. A Genetic Algorithm (GA) performing NE has been shown to outperform RL agents on benchmark problems, due to a GA's ability to escape local optima easily [12].

The main contributions of this article include a three degree of freedom rocket model incorporating variable aerodynamic coefficients and thruster pressure losses, to be used in future studies. Furthermore, an analysis is provided into the feasibility of using Soft Actor Critic (SAC) and a custom Particle SubSwarm Optimisation (PSSO) algorithm to learn a policy to provide a feasible powered descent throttle profile under a $65kPa$ dynamic pressure constraint. To ensure a fair comparison, an extensive ablation study is performed on the SAC, trialling different actor, critic, and temperature learning rates, as well as the Polyak averaging coefficient. The SAC was tested in both sparse and non-sparse reward settings, both failing to learn a feasible policy within 1000 and 2000 episodes, respectively. The PSSO employed a gated reward structure and found a viable solution within 600 episodes across the swarm. An evaluation, backed by literature, is provided on why the PSSO outperforms SAC for this problem.

This article starts by providing background to reinforcement learning and neuroevolution approaches applicable to this problem in Subsection II.A and Subsection II.B, respectively. The background section is finalised with a description of a launch vehicle's flight to powered descent and the constraints it faces in Subsection II.C. Following this, the problem is formulated by presenting the PSSO and SAC algorithms in Subsection III.A, the environment description in Subsection III.B, and the experiments performed in Subsection III.C. The results of the experiments are presented in Section IV, before being concluded, and future work directions discussed in Section V.

II. Background

This section provides background on the problem, focusing on Reinforcement Learning (RL) for continuous environments, Neuroevolution as an alternative to backpropagation and the launch vehicles' flight to the powered descent phase.

A. Reinforcement Learning

Reinforcement learning (RL) is a topic within the field of machine learning, where an agent learns an optimal policy to provide state-dependent actions through interaction with an environment, aiming to maximise the cumulative reward over an episode [13]. RL is a field with many algorithms to choose from; to perform algorithm selection between a few, it can be reduced through the consideration of algorithm characteristics. First, the learning strategy determines whether the policy will update dynamically as it progresses through the episode, known as online learning, or offline learning, which occurs through a pre-collected dataset. Online learning enables the policy to adapt to changes within the episode, but it can also lead to instability. Next, transition reusability is permitted if an off-policy method is chosen; in this case, transitions are stored in a replay buffer to be sampled in a batch format during later updates. In contrast, on-policy methods learn an optimal policy based on transitions generated by the active policy, thereby reducing sample inefficiency due to the lack of data reusability. A final consideration is to choose a model-based or model-free method. Model-based methods attempt to learn a dynamical model of the environment to support planning [14]. On the other

*https://ee.stanford.edu/news/2021/jan/stephen-boyd-cvxgen-guides-spacex-falcon?. Accessed 03-06-2025.

hand, model-free methods learn policies and value functions directly from the collected transitions.

Replay buffers store the transition data for off-policy learning. First, a uniform replay buffer was used to play Atari games, working in a first-in-first-out manner with random transition sampling, thereby breaking the temporal correlation of consecutive steps [15]. After, the prioritised replay buffer (PER) was tested on Atari games and showed a learning efficiency increased by a factor of two [16]. PER works by sampling transitions with a higher learning potential, based on their temporal difference (TD) error, which quantifies the difference between the prediction and the updated estimation of the state-action value pair, given the received reward and a bootstrapped estimation of the value received for the next state. The Hindsight Experience Replay (HER) buffer is an alternative buffer based on goal-orientation, where failed trajectories are relabeled as alternative goals that the agent did not achieve [17]. This motivates the agent to improve sample efficiency in sparse reward environments by learning from unsuccessful attempts as if they were successful.

Focusing on sample-efficient methods (online and off-policy) on a continuous domain, without the additional complexity of model-based applications, several algorithms have been presented in the literature based on the actor-critic framework. Here, an actor learns an optimal policy mapping the states to actions to maximise expected cumulative rewards. Meanwhile, the critic learns a mapping of states and actions to expected cumulative returns, thereby reducing the variance of training on raw rewards through smoothed action-value estimations. The mapping is learnt through TD error minimisation.

Deep Q-learning (DQN) has successfully learned optimal policies for playing continuous Atari games [15] by using neural networks to approximate the action-value function, thereby breaking the curse of dimensionality, as the neural networks act as function approximators. However, as this algorithm works on a discrete action space, applying it to a continuous action space would lead to a large action dimension, reducing learning efficiency, leading to the proposal of the Deep Deterministic Policy Gradient (DDPG) method [18]; a model-free off-policy online learning method in an actor-critic framework to learn mappings of state-action value functions and an optimal policy of state to actions. DDPG uses action space noise for exploration to break the deterministic nature of the policy, and updates target networks through Polyak averaging for enhanced learning stability. Twin Delayed DDPG (TD3) was proposed as an extension to DDPG, utilising double Q-learning through double critics to reduce value function overestimation bias, and clipped Gaussian noise for action-space exploration, reducing sharp action changes [19]. SAC was proposed as an alternative to DDPG and TD3 to overcome the sensitivity of convergence to hyperparameter choices [20]. SAC maximises expected returns and policy entropy to ensure sufficient exploration, with a learnable *temperature* parameter controlling the policy update to maintain a target entropy.

B. Neuroevolution

Neuroevolution (NE) is an alternative to gradient-based methods for optimising neural network architectures and parameters. Here, Evolutionary Algorithms (EAs) are employed as meta-heuristic black-box optimisers inspired by nature [21]. In terms of tuning the parameters of a neural network, they start by initialising a *population* of bounded candidate solutions, random weights and bias within a range, which are evaluated through a whole episode's returns via a *fitness function*; alike a reward function seen in RL. The fitness is then used to apply a nature-inspired operator to change the candidate solution, before the process is iterated until convergence.

NE has been successfully applied to RL benchmark problems, including Atari games and MuJoCo's humanoid walking problem, using the covariance matrix adaptation evolution strategy (CMA-ES) [11]. In specific scenarios, NE outperformed traditional RL algorithms of Asynchronous Advantage Actor-Critic (A3C) and Trust Region Policy Optimisation (TRPO) due to NE working well on problems with long time horizons, and black-box optimisers not being affected by reward distributions, allowing them to work well on tasks with sparse rewards where RL can be slow to converge. Additionally, RL can struggle when presented with non-informative gradients of policy performance, stemming from areas such as non-smooth rewards, which cause high variance in the value function and make it difficult to learn. As NE avoids using a value function estimator, it circumvents this difficulty.

Genetic Algorithms (GA) have also been compared to RL for training deep neural networks to play Atari games, showing competitive performance [12], with specific environments showing the GA outperforming traditional RL methods theorised to be due to GA's ability to escape local minima, whereas gradient methods can remain trapped. The GA is a popular EA where a population of candidate solutions are called *genes* with each parameter being a *chromosome*. For a neural network, each chromosome corresponds to a weight or bias. For each gene, an environment is initialised and run, with the episodic reward evaluated through an *objective function* that returns the *fitness* of the solution, indicating how well the gene performed. *Parents* are selected from the most promising solutions, with higher fitness. These parents

share genes through a process called *crossover*, which is used to maintain diversity for exploration while exploiting promising solutions, to create an *offspring* for the next population. A selection of the best-performing genes will be kept in the new population (*elitism*). Finally, a small number of genes will be *mutated* to unlock new parameter combinations.

Swarming Intelligence (SI) is an alternative population-based search method, sometimes included under the umbrella term of EAs, due to their similarity, but does not involve Darwinian-like mechanisms. As such, when used to tune a neural network's parameters, they can be argued to be included under NE. Ant Colony Optimisation (ACO) [22] and Particle Swarm Optimisation (PSO), inspired by the social behaviour of birds [23], are popular SI algorithms, each having additional variations [24, 25]. PSO initialises a *swarm* of randomly bounded *particles*. Like a GA, they are evaluated on episodic return through an objective function that returns the solution's fitness. The particles and the swarm's best positions (network parameters) are remembered and used to update the PSO. The PSO updates through Equation 1. The particle's velocities $\mathbf{v}$ update its positions $\mathbf{x}$ representing the parameters of the neural network. The velocity updates through three mechanisms:

1) An inertia term encourages particles to search the environment through a velocity that moves the particle. The inertia weight w decays over time, decreasing exploration.
2) The cognitive term brings the particle back to its best position. Here a random vector ξ_1 is multiplied by the particle's distance to it's personal best position $\mathbf{x}^\star$, weighted by the cognitive coefficient c_1.
3) The social term pulls the particle towards the swarm's best position. A random vector ξ_2 is multiplied by the particle's distance to the swarm's best position $\mathbf{x}_s^\star$, weighted by the social coefficient c_2.

$$\begin{aligned} \mathbf{v}_i \leftarrow & \underbrace{w\mathbf{v}_i}_{\text{inertia}} + \underbrace{c_1 \boldsymbol{\xi}_1 \circ (\mathbf{x}_i^\star - \mathbf{x}_i)}_{\text{cognitive}} + \underbrace{c_2 \boldsymbol{\xi}_2 \circ (\mathbf{x}_s^\star - \mathbf{x}_i)}_{\text{social}} \\ \mathbf{x}_i \leftarrow & \mathbf{x}_i + \mathbf{v}_i \end{aligned} \tag{1}$$

Backpropagation, the traditional RL update method, has been compared to PSO for updating a neural network's parameters. It was found that PSO converges faster on specific problems, attributed to problems with a high number of local minima [26, 27]. To improve its optimisation quality, Corporate PSO (CPSO) was developed to split the swarm into multiple *subswarms* [28], with periodic migration of particles and sharing of solutions.

C. Powered descent problem

A launch vehicle's mission profile can be divided into segments, each bounded by guidance objectives, an unchanged vehicle configuration and actuators. Considering a two-stage launch vehicle, ascent begins with a first burn that propels the launch vehicle vertically upwards to clear the launch tower. After the launch vehicle pitches over, it performs a *gravity turn*. The launch vehicle ascends through the dense lower atmosphere, causing dynamic pressure to increase, which requires engine throttling to avoid exceeding dynamic pressure limits. For SpaceX missions, this is commonly seen as $30 - 35kPa$. The launch vehicle pitches over through Thrust Vectored Control (TVC); as it pitches over, its horizontal speed increases, which is required for circularisation in orbit. Following this, the stages separate, with the first stage initiating its landing procedure and the second stage carrying the payload into orbit.

The first stage ascent burn is constrained by a horizontal and vertical velocity flight envelope, determining its ability to perform a Return to Launch Site (RTLS) or Away From Launch Site (AFLS) landing scenario [29]. When travelling to Low Earth Orbit (LEO) orbit, RTLS is available due to a steeper ascent than for Geostationary Orbit (GEO)[29], which is desirable as the first stage lands directly back at the launch site, allowing for rapid reusability. For RTLS, after stage separation, the first stage performs a flip-over manoeuvre through TVC to orient the first stage near the horizontal position. Following this, a boostback burn occurs to provide the first stage with a horizontal speed, allowing it to return to the landing pad.

When the desired horizontal speed is reached, the first stage coasts on a ballistic arc past its apogee, utilising cold gas thrusters from its reaction control system to orient for a low angle of attack. This is crucial when it descends into the denser atmosphere and vertical speed increases, as this procedure minimises aerodynamic moments. The ballistic arc comes to an end when the first stage needs to reduce its speed to manage aero-thermal-mechanical load constraints, through a burn. This marks the beginning of the powered descent phase, which RETALT, a EU-funded project to develop and validate reusable vertical-landing launch vehicle technologies, divides into two burns with a ballistic arc in between as depicted in Figure 1 [30].

1) Re-entry burn: to manage the velocity to within aero-thermal-mechanical load constraints. These loads occur due to high dynamic pressure, aerodynamic heating, and structural stresses. The RETALT1 first stage has a

maximum dynamic pressure of 100 kPa.

2) Low altitude ballistic arc: slows the stage down while providing trajectory correction from uncertainties in previous flight phases and this ballistic arc. The desired outcome is to slow the vehicle to meet set initial conditions for the landing burn.
3) Landing burn: aims to precisely land the stage on the landing pad with a zero touchdown velocity.

The Falcon 9 first stage also performs this split burn scenario. However, the Super Heavy booster performs a single burn, as its design can handle the aero-thermal-mechanical loads present during re-entry, providing different options based on the rocket's configuration and launch vehicle trajectory.

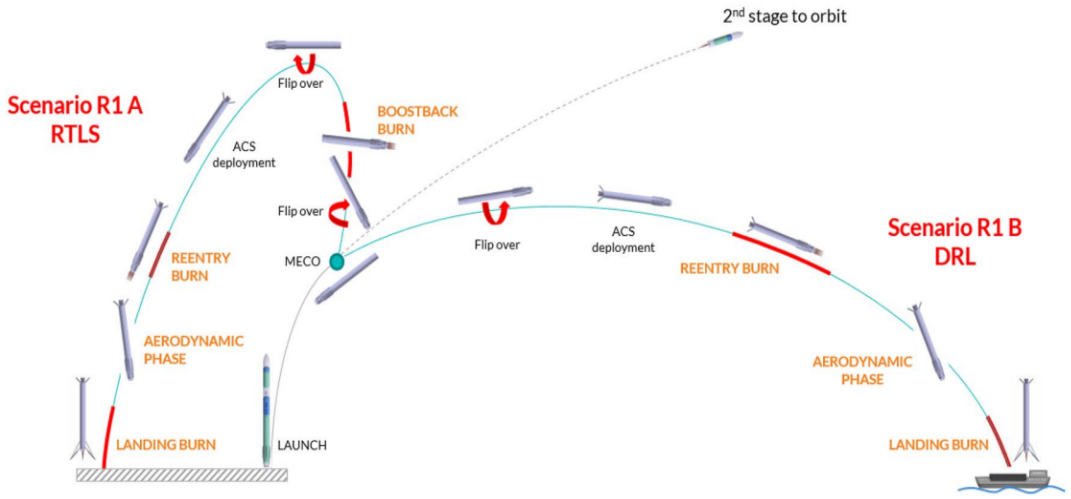


Fig. 1 RETALT1 return mission profile [30]

III. Problem Formulation

This section formulates the problem of comparing PSSO and SAC's performance in learning to perform powered descent. First, the SAC and PSSO algorithms are explained in Subsection III.A. Then, the environment is developed in Subsection III.B. Finally, the experiments are described in Subsection III.C.

A. Policy optimisation algorithms

This section presents the pseudo-code for the two algorithms used, first the PSSO in Algorithm 1, and then the SAC in Algorithm 2. First, the PSSO utilises independent subswarms to explore separate search trajectories. Each subswarm updates as in vanilla PSO and with inertia weight decay. Then, after a set number of generations, several random particles (n_{mig}) are transferred between the subswarms, allowing underperforming swarms to be reinvigorated for improved convergence. Moreover, the best subswarm has the opportunity to communicate its optimal position through soft influence if a communication threshold is passed, by a random number.

The SAC is an actor-critic method utilising a Gaussian actor and double critics. The Gaussian actor outputs a mean and standard deviation to provide state-dependent exploration. Entropy regularisation is used to encourage the policy to explore, which is controlled through its temperature ν, which the policy sets to ensure a target entropy is reached.

The experiments of the actor first utilised a uniform replay buffer; afterwards, a PER buffer was tried due to its increase in sample efficiency [16]. The PER buffer uses Algorithm 3 to sample transitions with high TD errors. Importance sampling weights w were used to weight the square TD error component of the critic loss. This augments the critic loss function to Equation 2, with N_b as the batch size, and the output y and state-action values Q_ϕ from Algorithm 2.

$$\mathcal{L}_\phi = \frac{1}{N_b} \sum_{k=1}^{N_b} w_k \left(Q_\phi(s_{i_k}, a_{i_k}) - y_{i_k} \right)^2 \tag{2}$$

Algorithm 1: Particle Subswarm Optimisation (PSSO)

Input: Generations G, population size N, number of subswarms S, inertia bounds $w_{\min}, w_{\max}$

Initialise S subswarms with random particle positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i = \mathbf{0}$;
Set $w \leftarrow w_{\max}$;
Initialise global best fitness $f^{\text{global}} \leftarrow -\infty$, position $\mathbf{x}^{\text{global}} \leftarrow$ None;
foreach $s \in \{1, \ldots, S\}$ **do**
    Set subswarm best $f_s^\star \leftarrow -\infty$, $\mathbf{x}_s^\star \leftarrow$ None;
for $g = 1$ **to** G **do**
    foreach *subswarm* $s \in \{1, \ldots, S\}$ **do**
        foreach *particle* i *in subswarm* s **do**
            Evaluate fitness $f_i \leftarrow f(\mathbf{x}_i)$;
            if $f_i > f_i^\star$ **then**
                $\mathbf{x}_i^\star \leftarrow \mathbf{x}_i, \quad f_i^\star \leftarrow f_i$;
            if $f_i > f_s^\star$ **then**
                $\mathbf{x}_s^\star \leftarrow \mathbf{x}_i, \quad f_s^\star \leftarrow f_i$;
    Update global best: $\mathbf{x}^{\text{global}} \leftarrow \arg\max_s f_s^\star$;
    foreach $s \in \{1, \ldots, S\}$ **do**
        foreach *particle* i *in subswarm* s **do**
            Sample $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2 \sim \mathcal{U}(0,1)^d$;

$$\mathbf{v}_i \leftarrow w\mathbf{v}_i + c_1\,\boldsymbol{\xi}_1 \circ (\mathbf{x}_i^\star - \mathbf{x}_i) + c_2\,\boldsymbol{\xi}_2 \circ (\mathbf{x}_s^\star - \mathbf{x}_i)$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$$

            Project $\mathbf{x}_i$ to domain: $\mathbf{x}_i \leftarrow \Pi_{\mathcal{X}}(\mathbf{x}_i)$;
    Update inertia: $w \leftarrow w_{\max} - \frac{w_{\max} - w_{\min}}{G} \cdot \dot{w}$;
    if $g \bmod \tau_{comm} = 0$ **then**
        Identify the top-performing sub-swarm:

$$s_{\text{best}} \leftarrow \arg\max_s f_s^\star$$

        foreach $s \neq s_{best}$ **do**
            if *rand()* $< p_{comm}$ **then**
                Soft influence $\mathbf{x}_s^\star \leftarrow (1 - \alpha_{\text{comm}})\,\mathbf{x}_s^\star + \alpha_{\text{comm}}\,\mathbf{x}_{s_{\text{best}}}^\star$;
                Re-evaluate new best performer $f_s^\star \leftarrow f(\mathbf{x}_s^\star)$;
    if $g \bmod \tau_{mig} = 0$ **then**
        Migrate n_{mig} random particles across subswarms;
return *Best position* $\mathbf{x}^{global}$ *and fitness* f^{global}

Algorithm 2: Soft Actor–Critic (SAC) with a uniform replay buffer

Input : Actor (policy network) parameters θ, critic (Q-value network) parameters ϕ_1, ϕ_2, target critic parameters ϕ_1^-, ϕ_2^-, replay buffer $\mathcal{R}$, intial temperature ν_0, batch size N_b, Polyak averaging coefficient τ, discount factor γ, target entropy $\mathcal{H}_{\text{target}}$, maximum number of episodes M, actor learning rate α_θ, critic learning rate α_ϕ and temperature learning rate α_ν

Initialize $\pi_\theta, Q_{\phi_1}, Q_{\phi_2}$;
Set $\phi_i^- \leftarrow \phi_i$, $\mathcal{R} \leftarrow \emptyset$, $\nu \leftarrow \nu_0$ $n_e \leftarrow 0$;
repeat
  Reset environment to obtain initial state s_0;
  $s_t \leftarrow s_0$;
  while *not* d_t **do**
    $(\mu_t, \sigma_t) \leftarrow \pi_\theta(s_t)$;
    $\varepsilon_t \sim \mathcal{N}(0, I)$;
    $a_t \leftarrow \mu_t + \sigma_t \odot \varepsilon_t$;
    Execute a_t;
    Observe (r_t, s_{t+1}, d_t);
    Store transition $(s_t, a_t, r_t, s_t', d_t)$ in $\mathcal{R}$;
    if $|\mathcal{R}| \geq N_b$ **then**
      sample $\{(s_i, a_i, r_i, s_i', d_i)\}_{i=1}^{N_b}$;
      for $i = 1 \ldots N_b$ **do**
        $(\mu_i', \sigma_i') \leftarrow \pi_\theta(s_i')$;
        $\varepsilon_i \sim \mathcal{N}(0, I)$;
        $a_i' \leftarrow \mu_i' + \sigma_i' \odot \varepsilon_i$;
        $y_i = r_i + \gamma\big(\min_j Q_{\phi_j^-}(s_i', a_i') - \nu \log \pi_\theta(a_i' \mid s_i')\big)$
      end
      for $j = 1, 2$ **do**
        $\phi_j \leftarrow \phi_j - \alpha_\phi \cdot \nabla_{\phi_j} \frac{1}{N_b} \sum_{i=1}^{N_b} \big(Q_{\phi_j}(s_i, a_i) - y_i\big)^2$;
      end
      $\theta \leftarrow \theta + \alpha_\theta \cdot \nabla_\theta \frac{1}{N_b} \sum_i \big(\nu \log \pi_\theta(a_i \mid s_i) - \min_j Q_{\phi_j}(s_i, a_i)\big)$;
      $\nu \leftarrow \nu + \alpha_\nu \cdot \nabla_\nu \frac{1}{N_b} \sum_i \big(-\nu \log \pi_\theta(a_i \mid s_i) - \nu \cdot \mathcal{H}_{\text{target}}\big)$;
      for $j = 1, 2$ **do**
        $\phi_j^- \leftarrow \tau\phi_j + (1 - \tau)\phi_j^-$
      end
    end
    $s_t \leftarrow s_{t+1}$;
    $n_e \leftarrow n_e + 1$;
  end
until $n_e = M$;
return θ

Algorithm 3: Prioritised Experience Replay (PER)

Input : Capacity N, prioritisation exponent α_t, small constant ε, batch size N_b, importance-initial sampling exponent schedule β_0, final sampling exponent schedule β_1, number of β annealing steps N_β

Initialize empty buffer $\mathcal{R}$ with capacity N;
Initialize priorities $p_i \leftarrow 0\ \forall i$;
while *training* **do**
  Observe transition $\mathcal{T} = (s_t, a_t, r_t, s_{t+1}, d_t)$;
  Compute temporal-difference error δ_t for $\mathcal{T}$;
  Set $p_t \leftarrow (|\delta_t| + \varepsilon)^{\alpha_t}$;
  if $|\mathcal{R}| < N$ **then**
    Append $\mathcal{T}$ to $\mathcal{R}$ and p_t to priorities;
  else
    Overwrite oldest entry with $\mathcal{T}$ and its priority p_t;
  end
  if $|\mathcal{R}| \geq N_b$ **then**
    $P(i) \leftarrow \dfrac{p_i}{\sum_j p_j}\ \forall i$;
    Sample indices $\{i_k\}_{k=1}^{N_b} \sim P(i)$;
    for $k = 1, \ldots, N_b$ **do**
      $w_k \leftarrow \left(|\mathcal{R}| \cdot P(i_k)\right)^{-\beta_t}$;
    end
    Normalise $w_k \leftarrow w_k / \max_j w_j\ \forall k$;
    return $\{e_{i_k}\}$, $\{i_k\}$, $\{w_k\}$
  end
  $\beta_t \leftarrow \beta_0 + (\beta_1 - \beta_0) \cdot \min(1, t/N_\beta)$;
end

B. Policy learning environment

This section describes the simulation model used as the learning environment, with the initial conditions defined in Subsection VI.B and the model parameters in Subsection VI.A.

1. Reference frame definitions and transformations

Four reference frames were used in this study:

1) The body frame had y'' pointing through the rocket's nose, and x'' perpendicular through the right-hand side; eastward when vertically upright. The pitch (θ) and flight path (γ) angles went counter-clockwise from x'',
2) To obtain local vertical-local horizontal (LVLH) frame forces ($F_{x'}, F_{y'}$), the body frame forces ($F_{x''}, F_{y''}$) were rotated around the pitch angle by Equation 3. The LVLH originated at the rocket's centre of gravity, and x' pointed horizontally eastward and y' vertically upwards.

$$F_{x'} = F_{y''} \cos\theta + F_{x''} \sin\theta \tag{3a}$$
$$F_{y'} = F_{y''} \sin\theta - F_{x''} \cos\theta \tag{3b}$$

3) The landing site centred vertical-horizontal frame (x, y) was the same as the LVLH frame, but centred around the launch tower/landing pad.
4) The aerodynamic reference frame for descent was defined as (x^a, y^a) originating from the centre of pressure, having y^a aligned with the the velocity vector and x^a $\frac{\pi}{2}$ radians counter-clockwise. The transformation of the aerodynamic forces (F_{x^a}, F_{y^a}) to the body frame is given in Equation 4, with the effective angle of attack (α_{eff}) being the angle of attack for a rocket in descent, as defined by Equation 5.

$$F_{x''} = F_{y^a} \cos\alpha_{\text{eff}} + F_{x^a} \sin\alpha_{\text{eff}} \tag{4a}$$
$$F_{y''} = F_{y^a} \sin\alpha_{\text{eff}} - F_{x^a} \cos\alpha_{\text{eff}} \tag{4b}$$

$$\alpha_{\text{eff}} = \gamma - (\theta + \pi) \tag{5}$$

2. Aerodynamics

The International Standard Atmosphere (ISA) defines atmospheric pressure, air density, and the speed of sound as a function of altitude, enabling the calculation of dynamic pressure (q) and Mach number (M). The dynamic pressure is dependent on the speed (V) and air density (ρ), which itself is a function of altitude (y). The lift and drag coefficients (C_L, C_D) were functions of Mach number and effective angle; from this, the lift and drag forces (L, D) were calculated in Equation 6b and Equation 6c respectively using the dynamic pressure and frontal area (S). Lift and drag act in the $-y^a$ and $-x^a$ axes, respectively. Finally, as the aerodynamic forces act at the centre of pressure, they generate a moment (M_z^a) as defined by Equation 7. Here, d was the distance from the base of the launch vehicle.

$$q = \tfrac{1}{2}\rho(y)V^2 \tag{6a}$$
$$L = qSC_L(\alpha_{\text{eff}}, M) \tag{6b}$$
$$D = qSC_D(\alpha_{\text{eff}}, M) \tag{6c}$$

$$M_z^a = (d_{cg} - d_{cp}) \cdot F_{x''}^a \tag{7}$$

3. Gravitational acceleration

The inverse-square law was used to model gravitational acceleration (g) as a function of altitude, as shown in Equation 8. This assumed a spherically symmetric and non-rotating Earth. Effects due to Earth's oblateness, rotation, and local anomalies were neglected. The gravitational acceleration acts in the negative y'' body axis.

$$g = g_0 \cdot \left(\frac{R_E}{R_E + y}\right)^2 \tag{8}$$

4. Thrust vector controlled engines

Powered descent uses the gimbaled engines[†] aboard the first stage to control its velocity, lateral motion and orientation. The engines will experience pressure losses when the exhaust gases expand less efficiently than at their optimised conditions, due to the nozzle exit pressure (p_e) not matching atmospheric pressure (p_a), leading to an under- or over-expanded flow. Equation 9 models this converting the loss free thrust per engine (T_e) to the actual thrust per engine (T) by adding pressure losses with A_e being the nozzle exit area. Moreover, to reduce the action space and learning complexity for this benchmark study, the engines had uniform throttle and gimbal angles, similar to literature on convex optimisation for powered descent guidance [3]. The gimbal angle was a counter-clockwise pitch angle that determined the thrust applied in the body frame, as shown in Equation 10.

$$T = T_e + (p_e - p_a) \cdot A_e \tag{9}$$

$$F_{x''} = -T^g \cdot \sin(\theta^g) \tag{10a}$$
$$F_{y''} = T^g \cdot \cos(\theta^g) \tag{10b}$$
$$M_z^t = -(d_{cg} - d_t) \cdot T^g \cdot \sin(\theta^g) \tag{10c}$$

[†]No fixed engines are typically active during powered descent.

5. Grid fins

Grid fins provide excellent control in both angular and lateral directions, while also providing an additional drag component to the vehicle. This study did not require angular control, as the aerodynamics effectively damped out the angle of attack; only a throttle profile was desired, and no disturbances were present. The simplified grid fin model of Equation 11 was used to provide an extra variable drag component through the grid fin's axial force coefficient (C_a) modelled as a function of Mach number.

$$F_{y''} = 2 \cdot q \cdot S \cdot C_a(M) \cdot \sin \alpha_{\text{eff}} \tag{11}$$

6. Dynamics

The explicit first-order forward Euler numerical integration scheme was used to integrate the kinematics. Newton's 2nd law, Equation 12, took the cumulative LVLH frame forces of gravity, thrust, grid fins and aerodynamic forces to find the acceleration (a) of the rocket with respect to the landing site centred vertical-horizontal frame (x,y), by dividing the force with mass (m). This was integrated to find velocity and position.

$$F = m \cdot a \rightarrow \ddot{x} = a = \frac{F}{m} \tag{12}$$

Euler's rotational motion equation, Equation 13, was used to enact the moment's change on the orientation (pitch) of the rocket through an enacted moment on the stage's inertia (I). As only pitch rotation was considered, no non-linear coupling terms were included. This was acceptable as many studies regarding the powered descent problem only take a three DoF model [1, 3–5], considering translational motion only, although a few do consider six DoF dynamics [6, 9].

$$\ddot{\theta} = \frac{M_z}{I} \tag{13}$$

C. Experiment description

The landing burn will impose a dynamic pressure constraint of $65kPa$[‡], reduced from RETALT's $100kPa$ limit, to demonstrate how policy learning can effectively manage heavy constraints that significantly affect the trajectory. Secondly, this heavy constraint combines the re-entry and landing burns into a single phase. Three experiments were conducted to demonstrate the performance of RL and NE methods, specifically SAC and PSSO NE. The experiments aim to perform a successful landing with minimal touchdown speed. Each experiment used the same *done* condition of speed beneath $5m/s$ between 0 and 1m, where an independent landing controller can take over. Early truncation was employed to reduce the learning time of unsuccessful episodes, following the logic outlined in Table 1.

The observable action space was (y, v_y), with the policy network outputting an action normalised through a tanh activation function to (-1,1), before being scaled to (0,1). Input normalisation was used to promote learning stability by normalising the initial conditions by their maximum [31].

Table 1 Truncation logic for a policy network performing powered descent.

Condition	Description
$y < -10$	The rocket has crashed into the ground.
$m_{\text{propellant}} \leq 0$	Propellant is depleted.
$q > 65\,\text{kPa}$	Dynamic pressure constraint.
$v_y > 0$	Rocket moves upwards, away from the landing pad.
$v_x > 0.01$	Rocket moves away from the landing pad.
$g_{1\text{s}} > 6$	Sustained 1-second g-load exceeds $6g$.

EXP 1: PSSO-NE. A relatively small swarm of 150 particles split into two subswarms following the PSSO algorithm of Subsection III.A using the hyperparameters of Table 6 was used. The sparse and gated fitness function of Equation 14 was applied to enable sequential optimisation. First, if the episode truncates above the ground, a penalty proportional to

[‡]Similar to the Falcon 9.

altitude was applied ($r = -|y|$). Then, after a ground impact, a terminal reward was given based on touchdown speed ($r = 200 - |V|$). Finally, when a successful landing occurred, the reward was the remaining propellant mass ($r = m_p$).

$$r(s,a) = \underbrace{H(y-0.1)\cdot H(\text{truncated})\cdot |y|}_{\text{Early termination penalty}} + \underbrace{H(0.1-y)\cdot H(\text{truncated})\cdot (200-|V|)}_{\text{Crash landing serverity reward}} + \underbrace{H(\text{done})\cdot m_p}_{\text{Landing reward}} \tag{14}$$

EXP2: SAC with non-sparse rewards. A non-sparse reward function of Equation 15 was shaped to encourage the agent to steer away from early truncation using quadratic normalised penalties. These were activated by heavyside step functions $H(\cdot)$ acting as if statements. This was accompanied by a descent progress term, and a slowdown reward for the last 100 meters. A terminal reward of remaining propellant or normalised altitude was given for *done* and early termination conditions, respectively. Utilising the knowledge gained from previous tests, coefficients were altered as shown in Subsubsection VI.C.2.

$$\begin{aligned} r(s,a) = \underbrace{\Pi_{-r_{\max},r_{\max}}}_{\text{Clipping}} \Bigg\{ & \underbrace{-H(q-q_{\text{thres}})\cdot W_Q \cdot \min\left[\left(\frac{q-q_{\text{thres}}}{q_{\max}-q_{\text{thres}}}\right)^2, 1\right]}_{\text{Dynamic pressure penalty}} \underbrace{- H(g_{1s}-g_{\text{thres},1s})\cdot W_G \cdot \min\left[\left(\frac{g_{1s}-g_{\text{thres},1s}}{g_{\max}-g_{\text{thres},1s}}\right)^2, 1\right]}_{\text{G-load penalty}} \\ & \underbrace{+ W_P\left(1-\frac{y}{y_0}\right)}_{\text{Descent progress}} \underbrace{+ H(100-y)\cdot W_S\cdot\left(1-\frac{|v_y|}{50}\right)}_{\text{Slowdown near surface}} \underbrace{- H(\text{truncated})\cdot W_C\cdot\frac{|y|}{y_0}}_{\text{Early Termination}} \Bigg\} \underbrace{+ H(\text{done})\cdot W_D\cdot\frac{m_p}{m_{p,0}}}_{\text{Successful landing reward}} \end{aligned} \tag{15}$$

EXP3: SAC with sparse rewards. The sparse reward scenairo gave rewards bound between $(-6, 10)$, with a $(-6, 6)$ reward scale when excluding the sucessfull episode reward. Equation 16 contains four rewards, first a normalised altitude penalty for early truncation before the landing pad, then a normalised velocity penalty for crash landing. The arguably non-sparse reward component was the slowdown near the surface reward. Finally, there was a propellant consumption fraction terminal reward on completed episodes, encouraging successful landing.

$$\begin{aligned} r(s,a) = & \underbrace{-6\cdot H(\text{truncated})\cdot H(y-0.1)\cdot\frac{y}{y_0}}_{\text{Early truncation before reaching pad}} \underbrace{+ H(\text{truncated})\cdot H(0.1-y)\cdot\left(6-\frac{|v|}{200}\right)}_{\text{Early truncation at pad}} \\ & \underbrace{+ 2\cdot H(100-y)\cdot\left(1-\frac{|v|}{200}\right)}_{\text{Slowdown near surface}} \underbrace{+ 10\cdot H(\text{done})\cdot\frac{m_p}{m_{p,0}}}_{\text{Successful landing}} \end{aligned} \tag{16}$$

IV. Results

This section presents and analyses the results of the study. First, Subsection IV.A presents the PSSO results, then Subsection IV.B the ablation study for a SAC under a non-sparse reward setting, and finally Subsection IV.C the SAC results with the sparse reward setting.

A. Particle subswarm optimisation performing neuroevolution

PSSO-NE uses non-gradient-based updates to optimise the policy network for powered descent. Figure 2 shows the fitness of the subswarms, with the best global fitness and the average fitness. Generation 4 provided the first initial solution satisfying the done condition. Following this, there were two successive steps at 18 and 34 generations, which provided a more optimal trajectory and resulted in a 90-ton reduction in propellant used. Subswarm 1 found six feasible solutions, whereas subswarm 2 found one over 47 generations, demonstrating the algorithm's sensitivity to the initial composition of the swarm. This indicated that multiple subswarms are beneficial for exploring different search trajectories. The average fitness showed a mean increase over time, but with a flattening trend at 35 generations. The fitness often followed an oscillatory trend, indicating high velocity changes in the particle, causing overshooting. The

resulting trajectory of Figure 3 shows a feasible trajectory, but a non-optimal trajectory, as the optimal trajectory would touch 65 kPa to minimise the flight time and hence the total propellant usage.

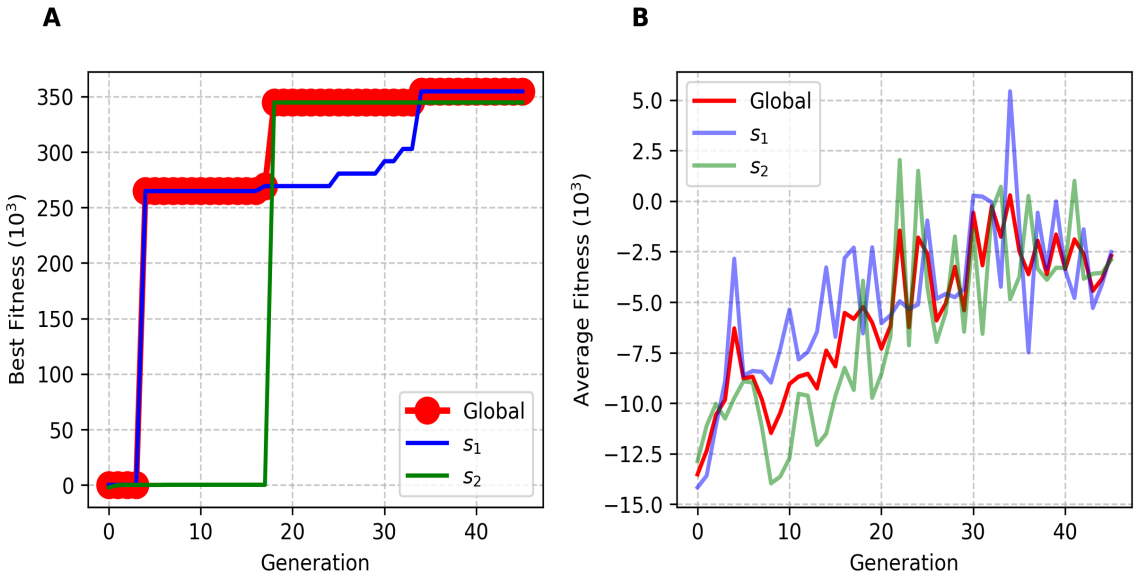


Fig. 2 Best fitness and average of two subswarms performing PSO to fit a neural network to perform powered descent. The left plot (A) displays the best fitness values, with the global best indicated by a red dot. The right plot (B) shows the average fitness of the subswarms, with the average value indicated in red.

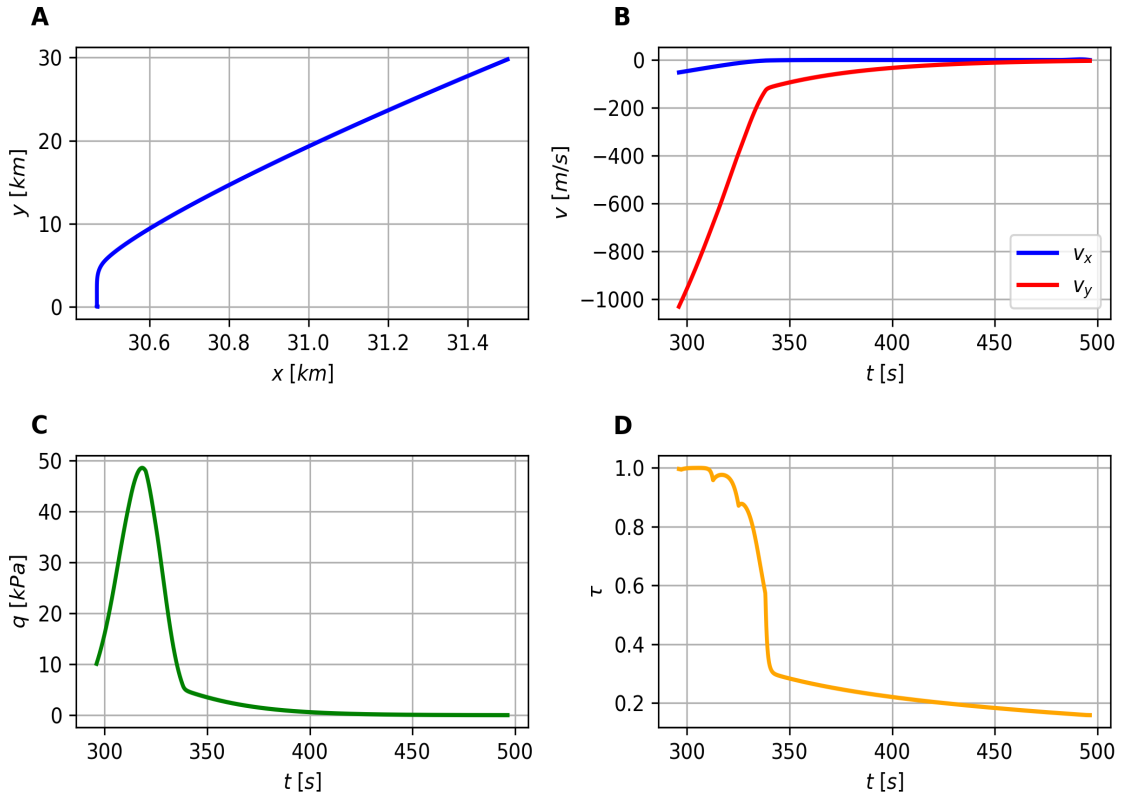


Fig. 3 Trajectory of the final candidate solution from the PSSO performing NE of Figure 2.

B. Non-sparse reward soft actor critic

1. Temperature learning rate

The ablation study started with the temperature learning rate. The policy update was under a fixed target entropy equal to the negative action dimension, $\mathcal{H}_{\text{target}} = -\dim\{a\}$ [20]. Temperature controls the exploration-exploitation trade-off in SAC. An increase in temperature shows less exploitation. The temperature loss and learning rate control the update of the temperature, with a lower loss indicating that the policy's entropy is closer to the desired target entropy, thereby balancing exploration and exploitation. Three temperature learning rates were tested across 1000 episodes, under the fixed hyperparameters presented in the Appendix table of Table 4. The results are shown in Figure 4.

All temperature learning rates exhibited a convergence of the temperature loss to zero, with the intermediate learning rate (0.0004) displaying the smoothest mean trend. In each case, the temperature dropped to approximately 0.02 by 27,500 steps. After the highest learning rate rose, a sharp increase was followed by a decline, indicating instability. In contrast, the lowest learning rate also experienced a steep rise, but nears a plateau around 0.14. The intermediate learning rate exhibited a steady increase in temperature, albeit with a high standard deviation, indicating consistent dynamic updates to the policy's entropy.

In terms of performance, all learning rates ended at similar training rewards. However, for the deterministic policy evaluation shown by the evaluation rewards, the lowest learning rate led to a minor improvement. In contrast, the highest learning rate had a sharp rise before plateauing. The intermediate learning rate rose to its highest evaluation reward at 700 episodes, with a sustained variance indicating a changing policy and ongoing improvements. Consequently, the intermediate temperature learning rate of 0.0003 was selected, as it balanced entropy adaptation with exploitation, resulting in a continuously changing policy with a mean trend of growth.

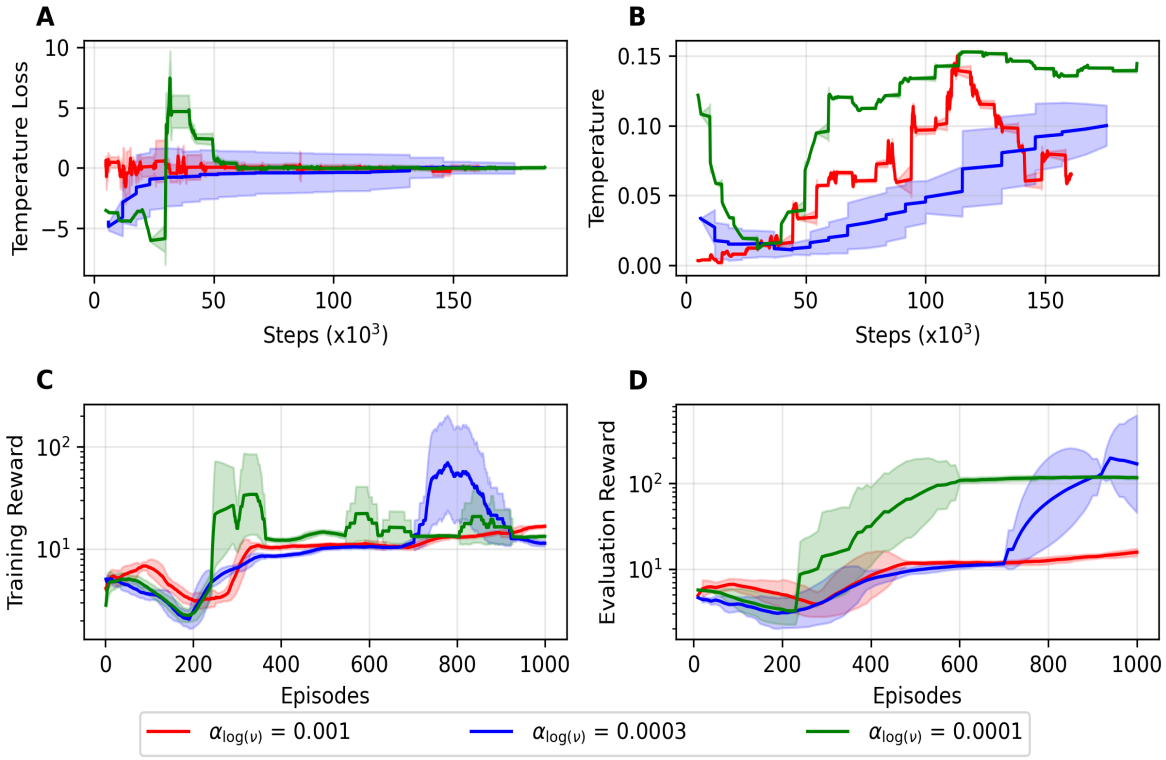


Fig. 4 Temperature adaptation in SAC RL for the first 1000 episodes of training. The figure presents, left to right, top to bottom: (A) temperature loss, (B) temperature, (C) training rewards, and (D) evaluation rewards for three different temperature learning rates: $\alpha_{\log(\alpha)}$ = 0.001 (red), 0.0003 (blue), and 0.0001 (green). Shaded regions represent $\pm 1\sigma$, calculated using a moving window of 50 timesteps for plots A and C, and 20 timesteps for plots B and D. Reward values in plots C and D are displayed using a logarithmic scale.

2. Critic learning rate

The critic's learning rate controls the update magnitude of the predicted state-action value (Q-value) network - the critic. A low critic loss and rising Q-values were desired, as the former indicates a reduction in the difference between the prediction and target state-action values, and the latter indicates an increase in expected cumulative rewards. Figure 5 presents an ablation study of four critic learning rates from $5e-3$ to $1e-5$. The highest critic learning rate showed the best performance. First, the critic loss gradually decreased, albeit with high variance, indicating slight instability. Second, the Q-values rose steadily above the others, and finally, both the evaluation and training rewards were the highest. The critic's dominating loss was from the TD errors, with a mean critic loss of 0.4 at a 37 Q-value, which gave an approximate relative error of 1.7%.

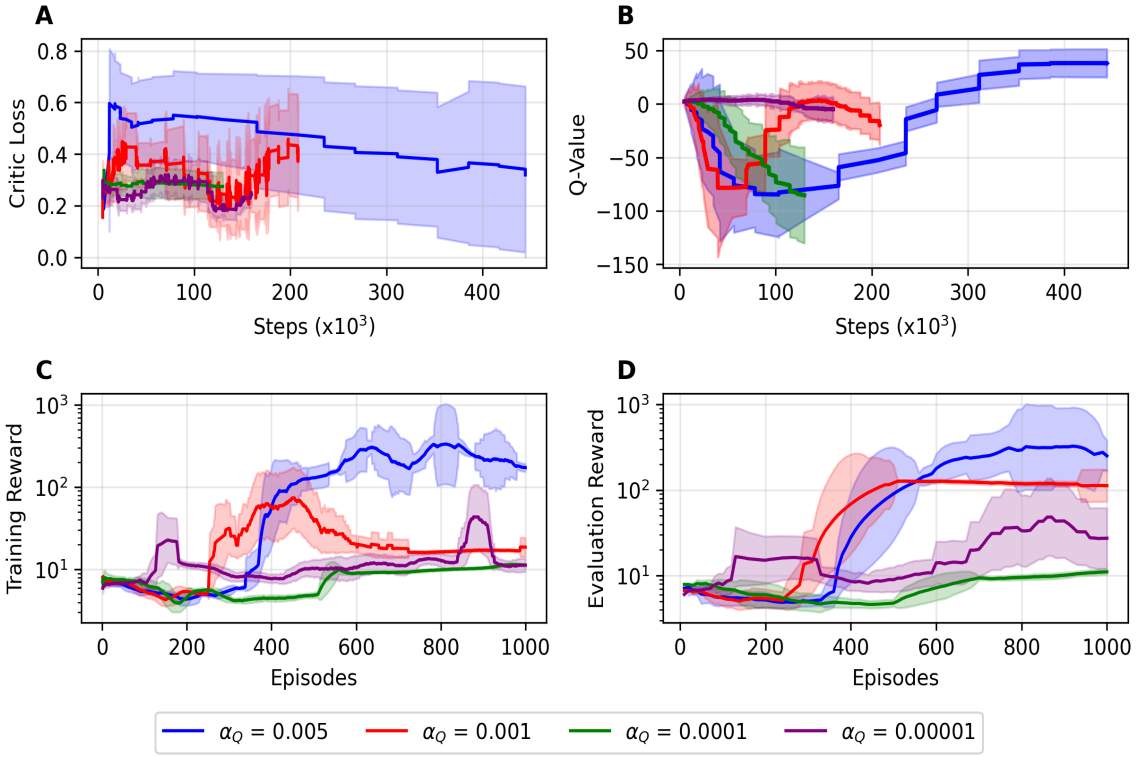


Fig. 5 Critic performance comparison in SAC RL for the first 1000 episodes of training. The figure presents, left to right, top to bottom: (A) critic loss, (B) Q-values, (C) training rewards, and (D) evaluation rewards for three different critic learning rates: $\alpha_Q = 0.005$ (blue), 0.001 (red), 0.0001 (green), and 0.00001 (purple). Shaded regions represent $\pm 1\sigma$. For plots A, C, and D, this was calculated using a moving window of 50 time-steps for plots A and C, and 20 time-steps for plot D. For plot B, the shaded region represents the batch Q-value σ deviation. Reward values in plots C and D are displayed using a logarithmic scale.

3. Polyak averaging coefficient

The target network update was controlled through the Polyak averaging coefficient, in turn affecting policy improvement through smoothed target network updates. A smaller coefficient should lead to a slower update, promoting stability. Conversely, a larger coefficient should facilitate faster adaptation, resulting in faster learning at the expense of increased risk from faster-moving targets. This, in turn, can lead to noisy critic updates, ultimately degrading actor performance due to poorly estimated Q-values. Figure 6 trials four Polyak averaging coefficients from 0.1 to 0.001, revealing $\tau = 0.005$ as arguably the most favourable candidate. First, the two intermediate coefficients exhibited the highest Q-value rise, and both converged to a near-zero critic loss, indicating good value prediction. The low coefficient did not converge to 0 critic loss within 100 episodes, as it takes fewer steps overall. Likewise, with the highest coefficient, but in this case attributed to a rise in temperature and not critic instability. The temperature increased exploration and led to earlier episode termination, stemming from the actor producing lower entropy than the target in early training, promoting the temperature to rise. The critic had a higher loss, causing worse actor updates, influencing the policy's

entropy. As a result, the lower intermediate value was chosen as the rewards end high with an early convergence in the critic loss.

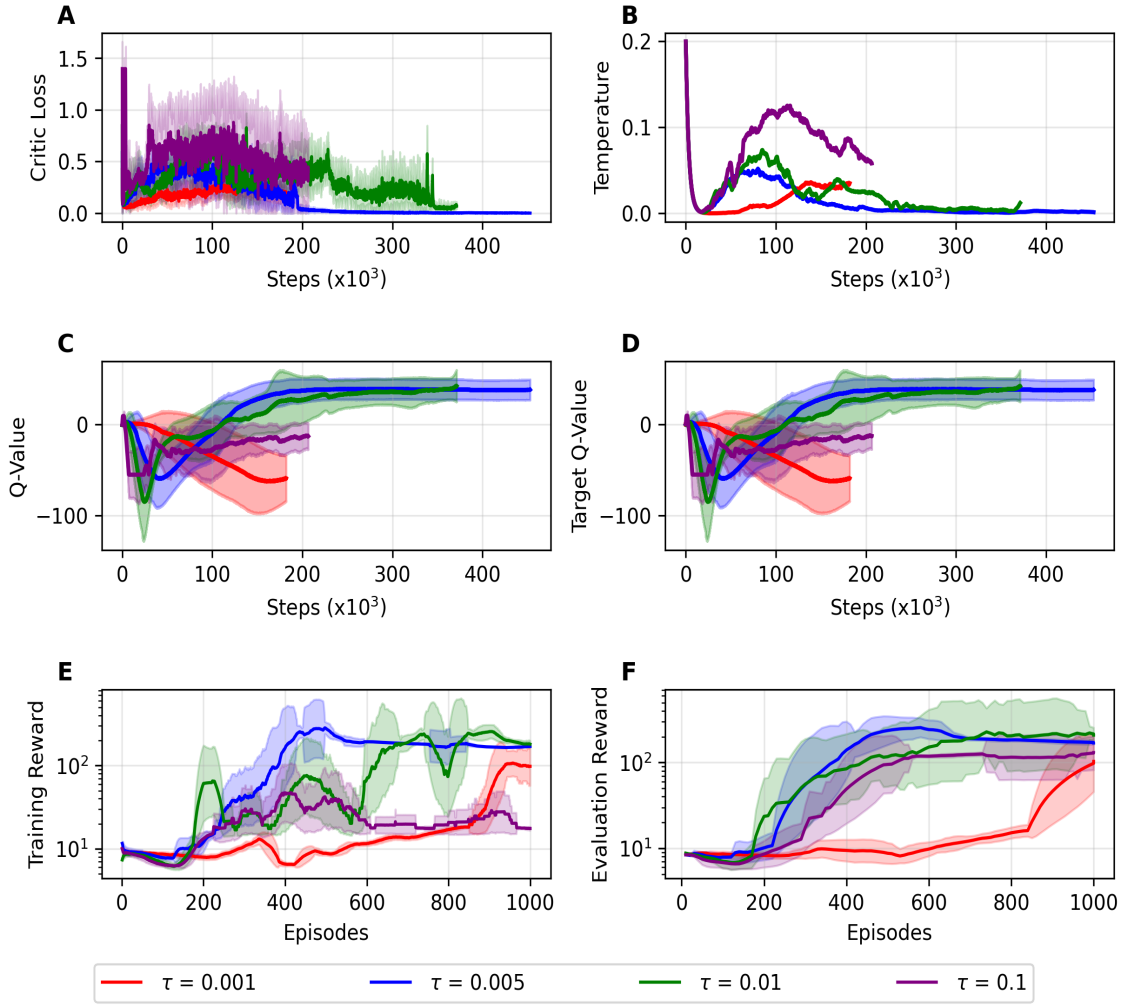


Fig. 6 Polyak averaging coefficient (τ) comparison in SAC RL for the first 1000 episodes of training. The figure presents (A) critic loss, (B) temperature, (C) Q-values, (D) target Q-values, (E) training rewards, and (F) evaluation rewards for four different Polyak averaging coefficients: τ = 0.001 (red), τ = 0.005 (blue), τ = 0.01 (green), and τ = 0.1 (purple). Shaded regions represent $\pm 1\sigma$. For critic loss and rewards, this was calculated using a moving window of 50 timesteps for critic loss and training rewards, and 20 timesteps for evaluation rewards. For Q-values and target Q-values, the shaded region represents the batch standard deviation σ. Reward values are displayed using a logarithmic scale.

4. Actor learning rate

The actor learning rate controls the effect of the gradient-based update of the policy network within the SAC. A higher learning rate provides more aggressive updates, but can lead to instability, resulting in a balance that needs to be found between learning stability and efficiency. Figure 7 shows the rewards, actor loss, and temperature for four different actor learning rates, all of which converged to similar training and evaluation rewards. All actor losses tended to approach negative 50, indicative of the Q-values of Figure 6. Still, the highest learning rate ($4e-3$) caused oscillations around a negative 50 loss, which was attributed to an overly large update over-correcting the policy as the critic stabilised. The rest of the policies exhibited a smooth rise in evaluation rewards, along with the temperature

decreasing to near zero, showing a well-balanced exploration-exploitation trade-off. This ablation study demonstrates that the actor learning rate is of secondary importance to the critic hyperparameters after the maximum bound has been found, as each learning rate yields a similar final performance. Resulting in the lowest actor learning rate of $\alpha_Q = 1e-5$ being chosen, as it should give the most stable updates.

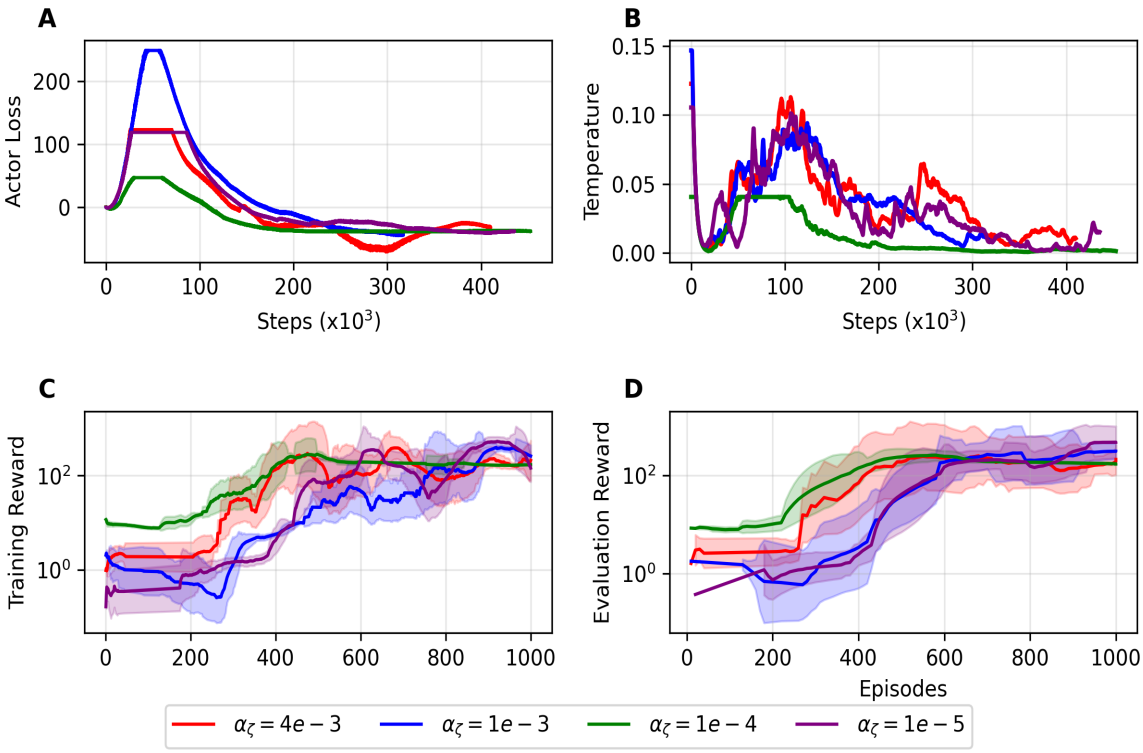


Fig. 7 Actor loss and performance comparison in SAC RL for the first 1000 episodes of training. The figure presents, from left to right and top to bottom: (A) actor loss, (B) temperature, (C) training rewards, and (D) evaluation rewards for four different actor learning rate configurations. Shaded regions in plots C and D represent $\pm 1\sigma$ standard deviation, calculated using a moving window of 50 timesteps for plot C, and 20 timesteps for plot D. Plot B shows the smoothed mean temperature values for each configuration. Reward values in plots C and D are displayed using a logarithmic scale.

5. Long run analysis

A longer run, of 2000 episodes, using a uniform replay was performed, and the learning results are shown in Figure 8. Colour-coded dashed vertical lines highlight key events. Additional graphs of the temperature and log probability, along with the estimated Q-values, are shown in Figure 13.

- 0 → 390 episodes: The rewards remained near zero as the policy didn't make it past the dynamic pressure constraint. The actor loss rose sharply to peak at 200, and the critic loss was noisy, rising above zero. The agent was untrained, and the critic loss was highly noisy because the critic had not yet been fitted to meaningful estimates; as can be seen from Figure 13, which showed a significant decline in Q-values to a mean of -200, with a trough just before 390 episodes. The actor loss rise is attributed to a low log probability and a high temperature, as shown in Figure 12, where the mean log probability settled at one, equal to the action dimension, and the temperature increased at episode 390.
- 390 → 730 episodes: The critic loss dropped to zero, and the actor loss decreased, as the critic's Q-values rose from better value estimation and increased expected cumulative rewards. The agent was learning well, so both the evaluation and training rewards increased, while the temperature decreased as the policy's entropy approached the target entropy. Here, the policy started overcoming the dynamic pressure constraint but terminated early as the

stage began to fly upwards. Near the end, the policy made it to the slowdown phase causing rewards to rise.

- 730 → 1070 episodes: The buffer was now filled with good experiences. However, the rewards showed a mean decrease as the policy terminated early, flying upwards before the slowdown phase. The actor loss had declined, and the critic loss had moved slightly from zero, as the Q-values exhibited increased variance and minimum and maximum values that were nearing positive and negative 200. This poor value estimation, stemming from new transitions downrange, led to inaccurate Q-value estimations, resulting from the dominant TD errors, which in turn worsened the critic's performance and, consequently, the actors.
- 1070 → 1420 episodes: The Q-value variance decreased, causing a low critic loss and only minor changes to the actor loss. The value estimations improved, and as a result, the rewards increased to the highest of this run, with the critic loss (un-smoothed) rising at the end of this period. The rewards increased as the policy moved into the slowdown phase, and the training transitions started to reach the landing pad, but at too high a speed.
- 1420 → 1610 episodes: The rewards dropped following the peak. As the raw critic loss was high (above 10), the actor loss decreased due to overestimated Q-values, before increasing as the Q-values estimation improved. The temperature fluctuated, rising and falling, as the maximum log probability lowered. Due to poor value estimation, from high variance q-values, the rewards decreased.
- 1610 → 2000 episodes: The variance of the Q-values decreased, accompanied by a drop in the critic loss, causing a rise in rewards as the policy escaped the dynamic pressure threshold; similar to near 390 episodes, indicating the policy was back to where it started.

The final trajectory, shown in Figure 12, indicates that the policy successfully passes the dynamic pressure constraint; however, throttling too hard afterwards resulted in a positive vertical velocity 2.5 km above the ground, before the slowdown reward was provided.

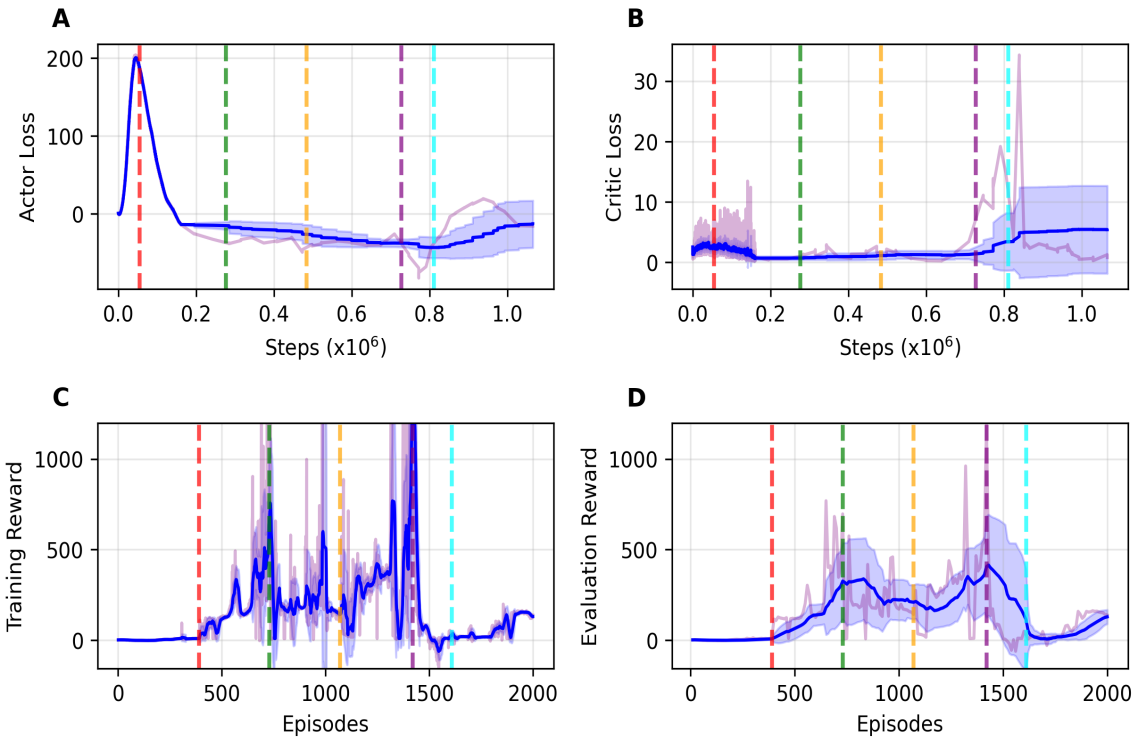


Fig. 8 SAC RL over 2000 episodes. The figure presents (A) actor loss, (B) critic loss, (C) training rewards, and (D) evaluation rewards. Raw data is shown in purple with low opacity, while smoothed trends are displayed in blue. Shaded regions represent $\pm 1\sigma$, calculated using a moving window of 50 timesteps for loss plots and 20 timesteps for reward plots. Vertical lines mark key episodes (390, 730, 1070, 1420, 1610) across all plots.

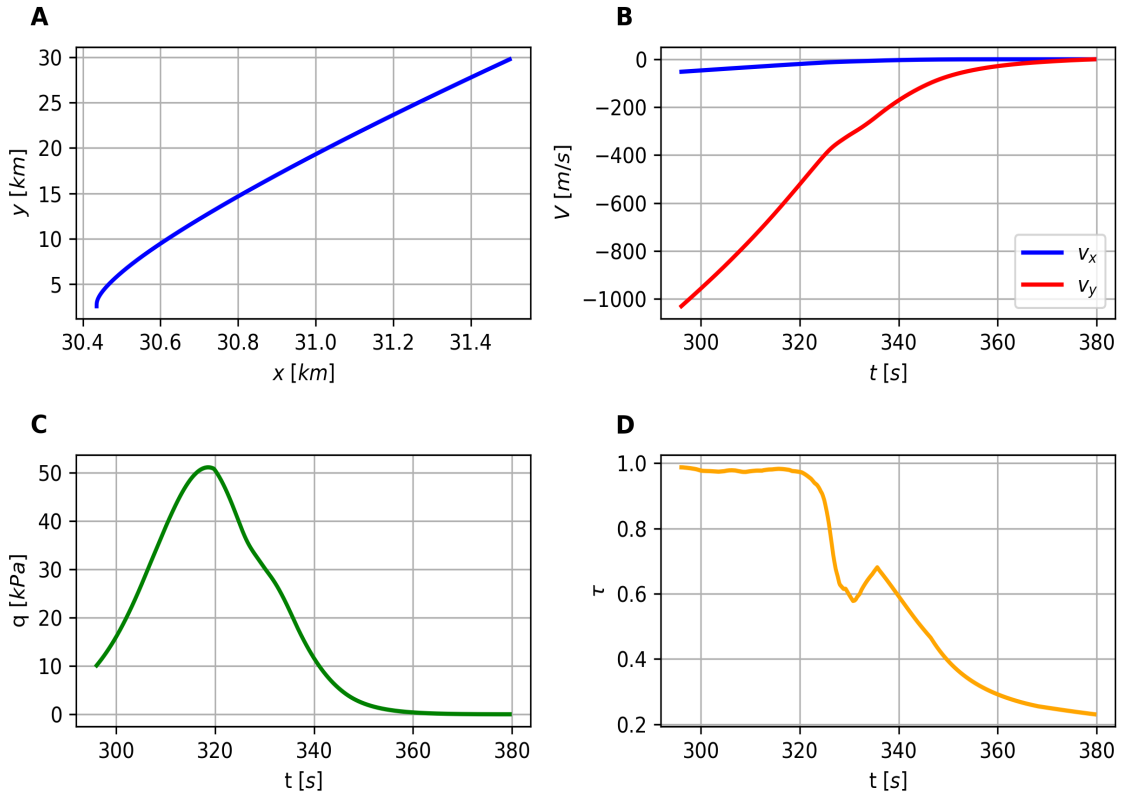


Fig. 9 The trajectory of the final SAC policy of Subsubsection IV.B.5

6. Prioritised Experience Replay buffer

The PER buffer performance was compared to that of a uniform buffer over 2000 episodes. Figure 10 shows that although the PER had higher rewards on average, with low and stable actor and critic losses at zero, it followed the same trend as the uniform buffer, as described in Subsubsection IV.B.5. Subsection VI.E, shows that the α used for the PER buffer was the best performing.

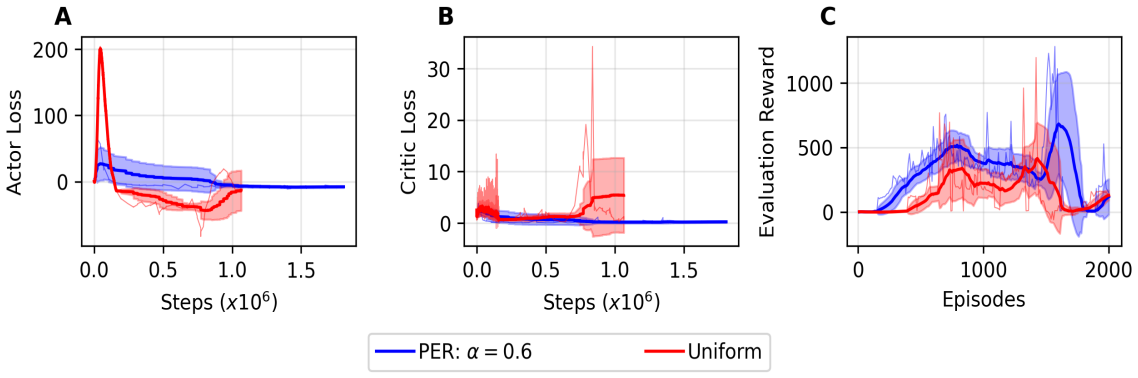


Fig. 10 Comparative analysis between Uniform and PER buffers in SAC RL over 2000 episodes. The figure presents (a) actor loss, (b) critic loss, and (c) evaluation rewards. Shaded regions represent $\pm 1\sigma$, calculated using a moving window of 50 timesteps for loss plots and 20 timesteps for evaluation rewards.

7. Final ablation study results

Figure 15 of Subsection VI.F confirms that the original discount rate of 0.99 was the best performing, due to receiving the highest evaluation rewards when compared to 0.95 and 0.9 discount rates. Furthermore, the original L2 critic loss was shown to outperform L1 critic loss in Figure 16 of Subsection VI.G.

C. Sparse reward soft actor critic

The sparse reward environment showed decreasing critic and actor losses, indicating that the SAC is improving its state-action value estimation and refining its policy accordingly. However, across 1,000 episodes, nearly 200,000 steps, the agent did not learn a policy to pass the dynamic pressure constraint.

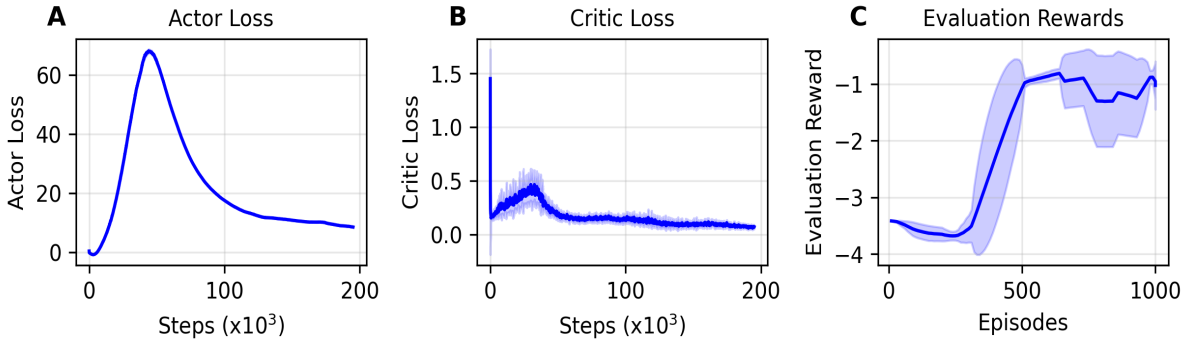


Fig. 11 Training progression of SAC with sparse rewards over the first 1000 episodes. The figure presents, left-to-right: (A) actor loss, (b) critic loss, and (c) evaluation rewards. Shaded regions represent $\pm 1\sigma$, calculated using a moving window of 50 timesteps for loss plots and 20 timesteps for evaluation rewards.

V. Conclusion

This section concludes the study, evaluates the findings and presents possibilities for future work.

A. Results summary

Of the two algorithms tested, only PSSO-NE provided a feasible solution. It found this within four generations of a 150-sized swarm with two subswarms. Before reducing mass consumption by 90 tonnes. This solution successfully avoided the dynamic pressure constraint and touchdowned with zero velocity. The subswarms exhibited different fitness trends as they explored different search trajectories.

The SAC in a sparse reward setting underwent an ablation study on its key hyperparameters: temperature, actor and critic learning rates, and the Polyak averaging coefficient. However, no feasible solutions were found in 2000 episodes, in contrast to the PSSO, which provided a viable solution within 600 episodes across the swarm. High variance Q-values and non-smooth critic loss show the SAC struggled with value estimation. Additionally, the sparse reward environment, although showing a steady decrease in actor and critic loss across 1000 episodes, did not get past the dynamic pressure constraint.

B. Reasoning neuroevolutions higher performance

The non-sparse RL agent has demonstrated an inability to generalise across states, as indicated by its temporal difference error. Although gradient clipping [31][§] and L2 regularisation [32] may help stabilise value estimation. On the other hand, PSSO demonstrated excellent convergence to a feasible solution, resulting in a successful landing in a relatively short time. PSO performs a black-box optimisation and, as such, was not affected by a reward distribution due to the absence of value estimation [11], which caused problems with the non-sparse reward SAC.

Sparse RL showed an agent that struggled to pick up a policy to overcome the dynamic pressure constraint limit. In a sparse reward environment, where feedback was scarce for most of the episode, the lack of reward gradients hindered the agent's learning efficiency. As a result, the policy got trapped behind the dynamic pressure constraint, a

[§]Andrychowicz et al. specially state that "gradient clipping is of secondary importance" compared to value function normalisation [31].

local optimum. The NE implementation easily escaped this dynamic pressure constraint by having a swarm of diverse candidate solutions. The diversity was increased through the incorporation of subswarms with migration and parameter sharing.

Furthermore, PSO works with episodic reward, so a gated reward function was easily shaped to avoid reward hacking. Additionally, episodic rewards provide the algorithm with a way to understand policies for long-term horizon problems effectively. Alternatively, for SAC, incorporating N-step returns could improve the long-term horizon performance of the SAC algorithm; however, this comes at the cost of higher reward variance [33].

C. Future work

Here, three recommendations for future research are presented in separate paragraphs.

NSGA-II is an EA that is well-suited to MOO [34]. For example, in control systems, this can be minimising rise time while minimising overshoot. NSGA-II sorts the population by Pareto dominance into layers, where a solution dominates another if it is at least as good as, or better than, the other in all objectives, presenting a helpful method for encompassing all descent flight phases. This would work by having extra genes that control the boostback burn's terminal horizontal velocity and the high-altitude ballistic arc's cutoff time, with the potential to extend this to the PD controller gains within the arc. Here, the objective would be to minimise fuel usage, minimise vertical velocity at the launch pad, and minimise the horizontal distance to the landing pad.

The horizontal wind profile is known before launch, and SpaceX now leverages machine learning for improved predictions[¶]. As a result, the trajectory generator can take into account the known horizontal wind; however, additional actuators providing pitch moments and perpendicular forces would be necessary to achieve this. For example, movable grid fins, gimbal systems, and variable throttle controls over the engines.

As model-free learning struggled to learn a policy to land the rocket, a change in algorithm or improved reward shaping is required [35]. Reference [9] suggested that a reward function could be realised through inverse reinforcement learning [36]. Alternatively, the PSSO's trajectory can provide a velocity profile guiding the reinforcement learning agent to a successful landing before applying terminal rewards to optimise mass. Alternatively, model-based reinforcement learning can support problems with long time horizons, as it learns the system's dynamics to roll out rewards that support effective planning [37].

D. Reflection

Reusable launch vehicles extend the complexity of previous planetary landings, as they require a smaller acceptable landing ellipse to meet the landing pad. As such, highly accurate methods are needed to facilitate this. The motivation for reusable launch vehicles is to reduce the cost of access to space and to increase the number of launches per year through rapid reusability. The reduced cost makes space more accessible to all, having potential economic, social and environmental benefits. For example, Earth-imaging satellites in Brazil are used to find illegal mining operations affecting the Indigenous communities[‖]. Earth imaging satellites enable the prediction of natural disasters and provide data on climate change, gathering information to inform the creation of crucial policies related to sustainability. Furthermore, easier access to space also allows space agencies like NASA and ESA to consider planetary exploration seriously. This work presents findings that motivate further research into using neuroevolutionary algorithms for trajectory generation and optimisation of the powered descent procedure, contributing to the future development of accurate and real-time feasible guidance and control systems for reusable launch vehicles.

References

[1] Açıkmeşe, B., III, J. M. C., and Blackmore, L., "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, Vol. 21, No. 6, 2013, pp. 2104–2113. https://doi.org/10.1109/TCST.2012.2237346.

[2] Blackmore, L., "Autonomous Precision Landing of Space Rockets," *The Bridge*, Vol. 46, No. 4, 2016, pp. 16–22.

[3] Açıkmeşe, B., and Ploen, S. R., "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, 2007, pp. 1353–1366. https://doi.org/10.2514/1.27553.

[¶] https://www.atmo.ai/. Accessed: 21 June 2025.

[‖] Maxar Blog, "Combating Illegal Gold Mining in the Amazon Rainforest With Maxar's High-Resolution Satellite Imagery," 15 February 2022, https://blog.maxar.com/earth-intelligence/2022/combating-illegal-gold-mining-in-the-amazon-rainforest-with-maxars-high-resolution-satellite-imagery Accessed: 26 May 2025.

[4] Blackmore, L., Açıkmeşe, B., and Scharf, D. P., "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization," *Journal of Guidance, Control, and Dynamics*, Vol. 33, No. 4, 2010, pp. 1161–1171. https://doi.org/10.2514/1.47202.

[5] Szmuk, M., Acikmese, B., and Berning, A. W., *Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints*, 2016. https://doi.org/10.2514/6.2016-0378, URL https://arc.aiaa.org/doi/abs/10.2514/6.2016-0378.

[6] Szmuk, M., and Acikmese, B., "Successive Convexification for 6-DoF Mars Rocket Powered Landing with Free-Final-Time," *2018 AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, 2018. https://doi.org/10.2514/6.2018-0617, URL http://dx.doi.org/10.2514/6.2018-0617.

[7] Shen, Z., Zhou, S., and Yu, J., "Real-time computational powered landing guidance using convex optimization and neural networks," *arXiv preprint arXiv:2210.07480*, 2022.

[8] Mattingley, J., and Boyd, S., "CVXGEN: A Code Generator for Embedded Convex Optimization," *Optimization and Engineering*, Vol. 12, No. 1, 2012, pp. 1–27.

[9] Gaudet, B., Linares, R., and Furfaro, R., "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing," *arXiv preprint arXiv:1810.08719*, 2018. URL https://arxiv.org/abs/1810.08719, accessed from arXiv.

[10] Wang, C., and Song, Z., "Convex Model Predictive Control for Rocket Vertical Landing," *Proceedings of the 37th Chinese Control Conference*, Wuhan, China, 2018, pp. 9837–9843.

[11] Salimans, T., Ho, J., Chen, X., and Sutskever, I., "Evolution Strategies as a Scalable Alternative to Reinforcement Learning," *arXiv preprint arXiv:1703.03864*, 2017.

[12] Such, F., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O., and Clune, J., "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

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

[14] Kaiser, L., Babaeizadeh, M., Milani, P., Osinski, R. H. C., Kohl, N., Nachum, O., Kurach, K., Guez, A., Rezende, D., Kingma, D. P., et al., "Model-based reinforcement learning for Atari," *International Conference on Learning Representations*, 2020.

[15] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing Atari with Deep Reinforcement Learning," , 2013.

[16] Schaul, T., Quan, J., Antonoglou, I., and Silver, D., "Prioritized Experience Replay," *arXiv preprint arXiv:1511.05952*, 2015.

[17] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W., "Hindsight Experience Replay," *Advances in Neural Information Processing Systems*, Vol. 30, 2017. URL https://arxiv.org/abs/1707.01495.

[18] Lillicrap, T. P., Hunt, J. J., Pritzel, A., et al., "Continuous Control with Deep Reinforcement Learning," *arXiv preprint arXiv:1509.02971*, 2015. URL https://arxiv.org/abs/1509.02971.

[19] Fujimoto, S., van Hoof, H., and Meger, D., "Addressing Function Approximation Error in Actor-Critic Methods," *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018, pp. 1587–1596.

[20] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018, pp. 1861–1870.

[21] Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1996.

[22] Dorigo, M., Maniezzo, V., and Colorni, A., "Ant system: optimization by a colony of cooperating agents," *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, Vol. 26, No. 1, 1996, pp. 29–41.

[23] Kennedy, J., and Eberhart, R., "Particle Swarm Optimization," *Proceedings of ICNN'95 - International Conference on Neural Networks*, IEEE, 1995.

[24] Bonabeau, E., Dorigo, M., and Theraulaz, G., *Swarm Intelligence from Natural to Artificial Systems*, Santa Fe Institute Studies in the Sciences of Complexity, Oxford University Press, 1999. URL http://www.us.oup.com/us/catalog/general/subject/LifeSciences/Neurobiology/?view=usa&ci=0195131592.

[25] Freitas, D., Lopes, L. G., and Morgado-Dias, F., "Particle Swarm Optimisation: A Historical Review Up to the Current Developments," *Entropy*, Vol. 22, No. 3, 2020. https://doi.org/10.3390/e22030362, URL https://www.mdpi.com/1099-4300/22/3/362.

[26] Gudise, V., and Venayagamoorthy, G., "Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks," *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No.03EX706)*, 2003, pp. 110–117. https://doi.org/10.1109/SIS.2003.1202255.

[27] Mendes, R., Cortez, P., Rocha, M., and Neves, J., "Particle swarms for feedforward neural network training," 2002, pp. 1895 – 1899. https://doi.org/10.1109/IJCNN.2002.1007808.

[28] Bergh, F., and Engelbrecht, A., "A Cooperative Approach to Particle Swarm Optimization," *Evolutionary Computation, IEEE Transactions on*, Vol. 8, 2004, pp. 225 – 239. https://doi.org/10.1109/TEVC.2004.826069.

[29] Botelho, A., Martinez, M., Recupero, C., Fabrizi, A., and Zaiacomo, G. D., "Design of the landing guidance for the retro-propulsive vertical landing of a reusable rocket stage," *CEAS Space Journal*, Vol. 14, 2022, pp. 551–564. https://doi.org/10.1007/s12567-022-00423-6, URL https://doi.org/10.1007/s12567-022-00423-6.

[30] De Zaiacomo, G., Blanco Arnao, G., Bunt, R., and Bonetti, D., "Mission engineering for the RETALT VTVL launcher," *CEAS Space Journal*, Vol. 14, 2022, pp. 533–549. https://doi.org/10.1007/s12567-021-00415-y.

[31] Andrychowicz, M., Raichuk, A., Stanczyk, P., Orsini, M., Harchaoui, Z., Espeholt, L., Marinier, R., Bachem, O., and Heess, N., "What Matters in On-Policy Reinforcement Learning? A Large-Scale Empirical Study," *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=r1etN1rtPB.

[32] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, Adaptive Computation and Machine Learning series, MIT Press, 2016. URL https://books.google.nl/books?id=-s2MEAAAQBAJ.

[33] Daley, B., White, M., and Machado, M. C., "Averaging *n*-step Returns Reduces Variance in Reinforcement Learning," , 2024. URL https://arxiv.org/abs/2402.03903.

[34] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T., "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 2, 2002, pp. 182–197.

[35] Ng, A. Y., *Shaping and policy search in reinforcement learning*, University of California, Berkeley, 2003.

[36] Ng, A. Y., and Russell, S. J., "Algorithms for Inverse Reinforcement Learning," *Proceedings of the International Conference on Machine Learning (ICML)*, 2000, pp. 663–670.

[37] Moerland, T. M., Broekens, J., and Jonker, C. M., "Model-based Reinforcement Learning: A Survey," *CoRR*, Vol. abs/2006.16712, 2020. URL https://arxiv.org/abs/2006.16712.

[38] Jo, B.-U., and Ahn, J., "Optimal staging of reusable launch vehicles considering velocity losses," *Aerospace Science and Technology*, Vol. 109, 2021, p. 106431. https://doi.org/https://doi.org/10.1016/j.ast.2020.106431, URL https://www.sciencedirect.com/science/article/pii/S1270963820311135.

[39] Sutton, G. P., and Biblarz, O., *Rocket Propulsion Elements*, 9th ed., John Wiley & Sons, Hoboken, NJ, 2016.

[40] Washington, W. D., and Miller, M. S., "Grid Fins – A New Concept for Missile Stability and Control," *Proceedings of the 31st Aerospace Sciences Meeting & Exhibit*, Reno, Nevada, 1993. https://doi.org/10.2514/6.1993-35.

VI. Appendix

A. First stage design values

This appendix section documents the values used in the environment for purposes of reusability. A two-stage launch vehicle with a reusable stage was optimally staged, considering estimated velocity losses taken from the literature [38], to carry a 100-tonne payload to LEO. Following this, the quantity of engines (Raptor 3) for the first stage matches the thrust-to-weight ratios of the Super Heavy Booster, setting the rocket's radius and allowing for tank sizing, as well as a low-fidelity estimation of the centre of gravity and inertia changes in the y"-axis with fuel consumption. The values used within the environment are reported in Table 2, excluding aerodynamic coefficients, which are in

the preceding section, and the inertia and centre of gravity functions, which are available in the GitHub repository: `https://github.com/JvanZyl1/PSSO-SAC-for-powered-descent`.

Table 2 Environment parameters set to three significant figures.

Parameter	Value	Source
A_e	$1.33m^2$	Corresponds to the reported thrust for the Raptor 3 sea-level optimised engines of SpaceX.
d_{cp}	$0.750 \cdot L$	Forward of the centre of gravity to ensure aerodynamic stability.
L	$50.4m$	Estimated length to fit the propellant and structural mass within the rocket's radius.
n_e	16	Number of gimballed engines aboard the first stage. 26 fixed engines are also present, but not used during descent, as seen on *Super Heavy*.
p_e	$100kPa$	Sea-level atmospheric pressure, as Raptor 3 sea-level variant is assumed optimised for sea-level atmospheric pressure.
S	$108\,\mathrm{m}^2$	A cylindrical rocket with a radius of 5.85 m to fit the 26 outer engines gives the frontal area.
T_e	$2.75MN$	Corresponds to the reported thrust for the Raptor 3 sea-level optimised engines of SpaceX.
v_{ex}	$3430m/s$	Corresponds to the Raptor 3 sea-level optimised engines of SpaceX.

Aerodynamics coefficients for lift and drag are prescribed as functions of effective angle of attack and Mach number, taken from the V2 curves as a benchmark [39]. The grid fin's normal and axial force coefficients are provided as functions of the local angle of attack at the grid fin and Mach number. Coefficient curves are taken from literature [40], with the drag coefficient approximated as the axial force coefficient.

B. Powered descent initial conditions

The powered descent phase starts at the end of a ballistic arc. Following rocket sizing using estimated velocity losses, the launch vehicle ascends following a trajectory constrained by the RETALT flight envelope for a RTLS scenairo [29], causing unused propellant from its allocation; which is acceptable as the velocity losses used in the optimal staging are not accurate, as it is a first iteration. Following stage separation, assumed instantaneous, the first stage flipped over and performed a boostback burn to an unoptimized boostback burn velocity of $100m/s$; this incorrect velocity will be updated in future work. This incorrect velocity will cause the rocket not to reach the landing pad, but it is not the purpose of this study. After the ballistic arc concluded, the powered descent phase began with a low effective angle of attack at a threshold dynamic pressure, carefully tuned to ensure maximum thrust is applied, thereby avoiding a lower dynamic pressure threshold of 60 kPa.

Table 3 Powered descent initial conditions to three significant figures.

Variable	$t[s]$	$x[km]$	$y[km]$	$v_x[m/s]$	$v_y[m/s]$	θ[rad]	$\dot{\theta}$[rad/s]	γ[rad]	$m[t]$	$m_p[t]$
Value	296	31.5	29.9	−52.2	−1030	1.52	0.000473	4.66	1630	1280

C. Policy optimisation algorithm parameters

1. SAC hyperparameters

Table 4 SAC hyperparameters used for the temperature ablation study.

Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Actor's hidden dimension	256	Actor's hidden layers	2	Critic's hidden dimension	256
Critic's hidden layers	2	Initial temperature	0.2	γ	0.99
τ	0.005	Grad-max-norm (clipping)	10.0	Maximum replay buffer size	100,000
Actor's learning rate	1×10^{-4}	Batch size	256	Critic's learning rate	1×10^{-4}

2. Reward function changes between experiments

Throughout the ablation study, the intuition gained allowed for a slightly tweaked reward function, as shown in Table 5, which displays the reward coefficients for the tests. The values in bold indicate the coefficients influencing the reward in these tests. A significant change in performance was observed when increasing the crash reward coefficient from 5 to 50 during testing of the Polyak averaging coefficient; here, the reward was clipped to a value of -10 until the ground was reached.

Table 5 Reward weights used for ablation study tests.

Ablation	W_Q	W_G	W_P	W_S	W_D	W_C	Clip
$\alpha_{\log(\alpha)}$	**1**	**1**	**0.5**	1.5	5	**5**	10
α_Q	**1**	**1**	**0.5**	1.5	105	**5**	10
τ	**1**	**1**	**0.5**	**5.5**	405	**50**	**10**
α_ζ	**1**	**1**	**0.5**	**5.5**	400	**50**	**10**

3. Particle SubSwarm Optimisation hyperparameters

The neural network consisted of three hidden layers, each with eight neurons, resulting in 169 parameters. The network was kept small due to computational constraints, such that a swarm of 150 particles resulted in nearly one particle per parameter.

Table 6 Particle SubSwarm optimisation hyperparameters

Variable Name	N	S	c_1	c_2	$w_{\max}$	$w_{\min}$	τ_{mig}	n_{mig}	ρ_{comm}	τ_{comm}	α_{comm}
Value	150	2	1.0	1.0	0.9	0.4	5	1	0.5	10	0.3

D. Extra long run results for SAC in the non-sparse rewards setting.

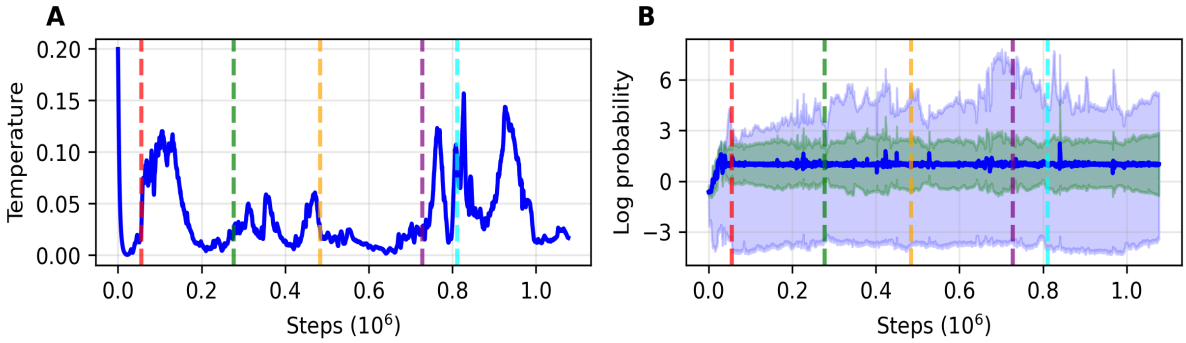


Fig. 12 SAC temperature parameter and log probabilities during training 2000 episodes. The left plot (A) shows the smoothed temperature coefficient, which balances exploration and exploitation by weighting the entropy term in the objective function. The right plot (B) shows log probabilities of actions with mean (blue line), $\pm 1\sigma$ bounds (green area), and full range (blue area). Vertical lines mark key episodes (390, 730, 1070, 1420, 1610) across both plots.

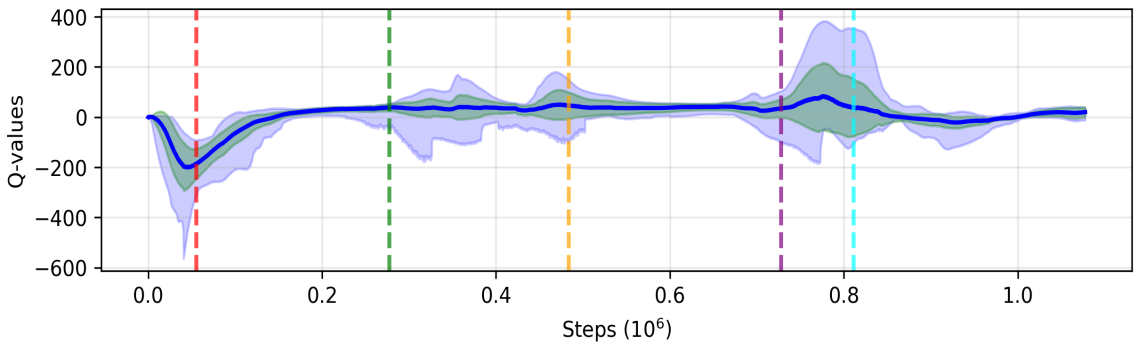


Fig. 13 SAC predicted Q-values over 2000 episodes. The plot shows mean Q-values (blue line) with $\pm 1\sigma$ bounds (green area) and full range from minimum to maximum (light blue area). Vertical lines mark key episodes (390, 730, 1070, 1420, 1610) across both plots.

E. Prioritised Experience Replay ablation study results

The PER buffer was tested against the uniform buffer. The PER buffers used a β of 0.4 and N_β of 100000 steps.

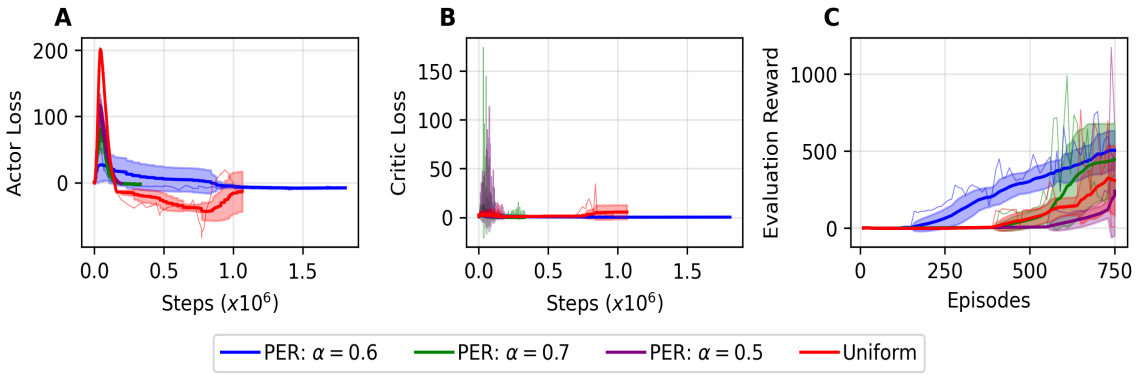


Fig. 14 Comparison of SAC RL training using Uniform and PER buffers with varying prioritisation exponents (α). Subplots show: (A) actor loss, (B) critic loss, and (C) evaluation rewards across training steps or episodes. Each curve corresponds to a different buffer configuration: Uniform (red), PER $\alpha = 0.5$ (purple), PER $\alpha = 0.6$ (blue), and PER $\alpha = 0.7$ (green). Solid lines indicate moving average trends (window size: 50 for losses, 20 for rewards), while shaded regions denote $\pm 1\sigma$, reflecting statistical uncertainty. Results are averaged over up to 2000 episodes.

F. Discount factor ablation study results

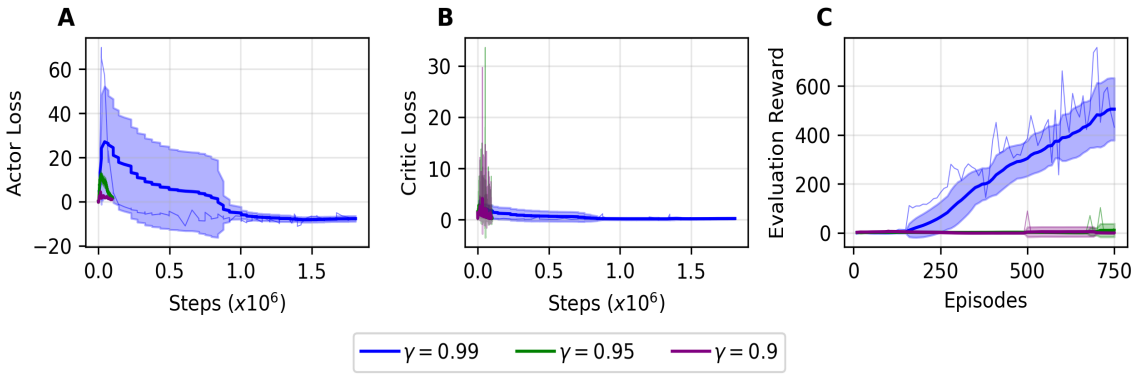


Fig. 15 Comparison of SAC RL with different discount rates γ over 750 episodes. The figure presents (A) actor loss, (B) critic loss, and (C) evaluation rewards. Shaded regions represent $\pm 1\sigma$, calculated using a moving window of 50 timesteps for loss plots and 20 timesteps for evaluation rewards.

G. L1 and L2 loss

L1 loss calculates the mean absolute error through Equation 17, and L2 loss the mean squared error through Equation 18. L2 loss squares the error and, as such, penalises larger errors heavily.

$$\mathcal{L}_1 = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{17}$$

$$\mathcal{L}_2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{18}$$

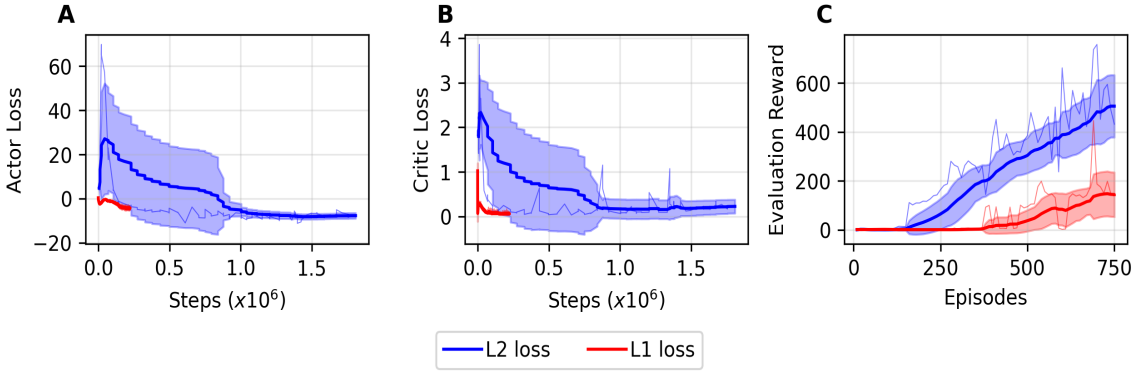


Fig. 16 Comparison of SAC RL using L1 and L2 loss functions for the critic update. Subplots show: (A) actor loss, (B) critic loss, and (C) evaluation rewards across training steps or episodes. Each curve corresponds to a different loss function: L2 loss (blue) and L1 loss (red). Solid lines indicate moving average trends (window size: 50 for losses, 20 for rewards), while shaded regions denote $\pm 1\sigma$.

3

Literature Study

The chapter reports the findings of a literature study aimed at establishing a foundation for applying policy optimisation methods to the powered descent problem. First, Section 3.1 reviews the current Guidance, Navigation and Control (GNC) system used in industry and academic studies, along with the flight phases encountered for a reusable launch vehicle. Following this, Section 3.2 reviews the main choices in selecting a policy optimisation method for the powered descent problem to provide a more targeted approach for the RL literature study of Section 3.3. Section 3.4 reviews *Starship* as a case study, before documenting a procedure to stage a reusable launch vehicle optimally, and finally presents launch vehicle modelling features from literature. Finally, neuroevolution as an alternative to backpropagation is discussed in Section 3.5.

3.1. Launch vehicle landing control systems

Space exploration is a rapidly evolving field, with engineering advancements driving a shift towards more precise and efficient landing systems. Landing systems are critical to enabling cost-effective launches of satellites and human space exploration. Blue Origin and SpaceX have developed reusable launch vehicles, such as the New Shepard 4[1] and the Falcon 9, to reduce the costs associated with space access. The most advanced launch vehicle is SpaceX's *Starship*, which has to hover accurately a few 10s of meters above ground before being caught by metal arms, as shown in Figure 3.1. NASA's Artemis program[2] will extend reusable launch vehicles to the Moon and Mars, requiring landing systems in environments less understood than Earth.

To land the launch vehicle, precision and robust control systems are required, containing sensors, actuators and control algorithms. To understand the traditional mechanisms and algorithms used, first, RETALT1's[3] GNC system was reviewed and applied to the problem definition in Subsection 3.1.1. Following this, the different flight phases a two-stage launch vehicle covers during ascent and landing are reviewed in Subsection 3.1.2, focusing on the RETALT1 description and real-world mission profiles from *Starship*.

[1] `https://www.blueorigin.com/new-shepard`. Accessed: 06/23/2025.

[2] `https://www.nasa.gov/humans-in-space/artemis/`. Accessed 23/06/2025.

[3] The RETALT1 project was a European Union Horizon program initiative to advance the research and development into technologies needed for vertical landing launch vehicles. `https://www.retalt.eu/project/`. Accessed 23/06/2025.

Figure 3.1: Starship catch. Image credit: SpaceX.

Finally, the popular high-level trajectory optimisation methods of convex optimisation are reviewed in terms of lossless convex optimisation and MPC. Trajectory optimisation is a crucial aspect of landing, as it ensures a safe and accurate landing with minimal fuel consumption. However, traditional methods are shown to be computationally complex, which can limit their effectiveness in real-time applications, resulting in a trade-off between solution optimality and real-time feasibility. Although CVXGEN, a generator for fast code for quadratic program representable convex optimisation problems, used the Falcon 9 diminishes the validity of this argument. Advances in machine learning in the field of deep reinforcement learning have utilised neural networks as function approximators, providing a potential solution that can offer a real-time feasible and integrated guidance and control solution. These three methods are reviewed in Subsection 3.1.3.

3.1.1. Guidance, Navigation and Control architecture

GNC systems play a crucial role during powered descent by autonomously manoeuvring the spacecraft to a safe landing. Here, RETALT1's GNC system is reviewed as a case study [7]. Sensor readings and a reference trajectory are the inputs to RETALT1's GNC system. First, the **flight manager** sets the flight phase, for example, vertical rising or boostback burn. Then, the **navigation module** takes in sensor readings and processes them when required through techniques like filtering or state estimation to estimate the state of the rocket. With the flight phase given from the flight manager, the estimated state from the navigation model and the reference profile already generated, the **guidance module** provides a reference attitude and actuation commands to the control module, where feedback is used from the control system to augment its outputs (control reference). Finally, the **control module** controls the actuators. This process is shown clearly in Figure 3.3.

To better understand the observable state of the vehicle without complex state-estimation techniques, the sensors were examined. These are listed below, and the total observable state space is defined in Equation 3.1. Aside from sensor readings, mass states are assumed to be observable, as the mass used can be estimated from the throttle commands.

- Inertial Measure Unit (IMS): measures acceleration and angular velocity through gyroscopes and accelerometers.
- Global Navigation Satellite System (GNSS): provides the rocket's position.
- Differential Global Positioning System (DGPS): is similar to GNSS, but provides more accurate position information through a ground-based correction.

- Flush Air Data System (FADS): measures aerodynamic parameters like dynamic pressure, static pressure, angle of attack, side-slip angle and Mach number.
- Altimeter: measures altitude.

$$\mathbf{o} = \Bigg[\underbrace{\{x, y, z\}}_{\text{Position}}, \underbrace{\{\vec{v}, \vec{a}\}}_{\text{Linear states}}, \underbrace{\{\omega_x, \omega_y, \omega_z\}}_{\text{Angular velocities}}, \underbrace{\alpha, \theta, \gamma, \beta}_{\text{Angles}}, \underbrace{\rho, M, p_a}_{\text{Aerodynamics}}, \underbrace{m_{\mathrm{p}}, m}_{\text{Masses}}\Bigg] \tag{3.1}$$

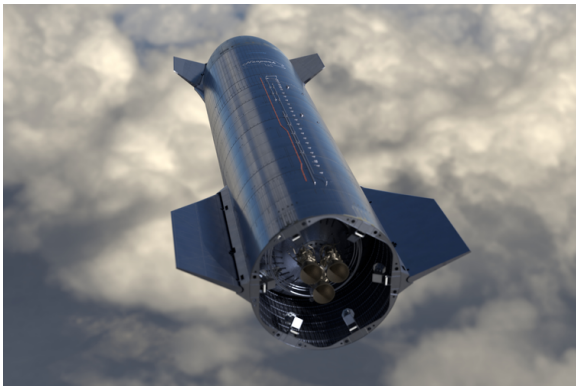
In terms of actuators, each thruster has a main valve that initiates or shuts down the thruster. Second, for a full-flow staged combustion cycle, where the propellants are gasified in pre-burners and drive turbopumps in the main combustion chamber, the turbopump's rpm is altered to throttle the engines. Furthermore, throttling is required during ascent to prevent the launch vehicle from travelling above its dynamic pressure threshold. During ascent, the launch vehicle needs to build up vertical speed to reach the desired semi-major axis and minimise gravity losses. However, it also needs to generate a horizontal speed to ensure circularisation of the orbit, resulting in a curved mission profile, which requires actuators to torque the launch vehicle into performing a pitch-over manoeuvre. In terms of thrusters, the Thrust Vector Control (TVC) system is responsible for this. SpaceX's launch vehicles have a gimbaled pivot TVC system, a "simple, proven technology" [52].

Propulsive free flight, *ballistic arcs*, require actuators capable of attitude control without ignited thrusters. Two systems are employed: the Aerodynamic Control Surfaces (ACS) are used in the lower (denser) atmosphere, as their effectiveness is linearly dependent on dynamic pressure, and the Reaction Control System (RCS) is used in the upper atmosphere.

The RCS provides small Δv and orientation corrections in the upper atmosphere, but it is mainly used for attitude control [52]. RCS's control authority is dominated by cold-gas thrusters, with pairs in each axis at the top and bottom of the launch vehicle, where the moment arm is greatest. Due to the "pair" nature of the actuators, they can provide a pure moment without forces. Reaction wheels can also be used for fine control without propellant consumption; however, only thrusters are considered here, as they give a larger moment. Notably, SpaceX doesn't carry dedicated cold-gas propellant, but instead uses excess ullage gas. Ullage gas is an inert non-combustible gas, like gaseous nitrogen (N_2), used to pressurise propellant tanks.

In terms of ACSs, Super Heavy, the first stage of SpaceX's launch vehicle *Starship*, features four electrically actuated grid fins (Figure 3.2b), each containing a lattice of small aerodynamic control surfaces. Mounted at the top of the first stage, a large moment arm allows for heavy attitude control, and it also provides additional aerodynamic drag to the vehicle. Starship, the second stage of its namesake, uses a pair of flaps at the top and bottom to control orientation, allowing Starship to perform a near-horizontal "*bellyflop*" landing to utilise aerodynamic drag.

The *bellyflop* manoeuvre optimises landing fuel efficiency and manages re-entry forces. After re-entering Earth's denser atmosphere, Starship will reorient horizontally, as shown in Figure 3.2a, significantly reducing drag and terminal velocity while distributing heating loads across the vehicle. At around 500 metres above ground, Starship's sea-level-optimised Raptor engines reignite, and the rear flaps fold inward, causing a rapid flip to vertical orientation. Aside from SpaceX, New Glenn, Blue Origin's reusable launcher, uses four adjustable fins at the top of its upper stage for aerodynamic orientation control.



(a) Starship's orientation during bellyflop.

(b) Super Heavy's grid fins.

Figure 3.2: Aerodynamic control surfaces used by SpaceX vehicles. (Credit: SpaceX).

Using SpaceX's Starship launch vehicle as a reference, the total available action space can be constructed. This is illustrated below, with the RCS simplified to a single moment command.

- Super Heavy contains 20 fixed perimeter engines, 13 gimballed engines, cold-gas thrusters for reaction control and four grid fins.
- Starship has three fixed vacuum-optimised engines and three gimballed sea-level-optimised engines used for the landing burn. Also, there are pairs of fins at the top and bottom of the stage, along with a RCS.

$$\mathbf{a}_{\text{superheavy}} = \left[\underbrace{\{\tau_i, MVM_i\}_{i=1}^{33}}_{\text{Throttle}}, \underbrace{\{\theta_i^g, \psi_i^g\}_{i=1}^{13}}_{\text{Gimbal}}, \underbrace{M_{RCS}}_{\text{RCS}}, \underbrace{\{\theta_i^{gf}, \psi_i^{gf}\}_{i=1}^{4}}_{\text{Grid Fins}} \right] \tag{3.2}$$

$$\mathbf{a}_{\text{starship}} = \left[\underbrace{\{\tau_i, MVM_i\}_{i=1}^{6}}_{\text{Throttle}}, \underbrace{\{\theta_i^g, \psi_i^g\}_{i=1}^{3}}_{\text{Gimbal}}, \underbrace{M_{RCS}}_{\text{RCS}}, \underbrace{\{\theta_i^{gf}\}_{i=1}^{2}}_{\text{Grid Fins}} \right] \tag{3.3}$$

The action space can be simplified for our problem by excluding roll control, which involves removing the gimbal and grid fin roll angles, thereby reducing the number of controllable grid fins to two. Additionally, for different flight phases, different actuators will be active, as set by the flight manager. Finally, the gimbal and throttle commands can be uniform across each gimballed or non-gimballed thruster, thereby reducing the complexity of the problem and, consequently, the time required to learn a policy. Applying uniform thrust and gimbal angles if often seen in work on lossless convex optimisation for powered descent [3].

The available action space in the reduced form for the problem is Equation 3.4 for the first stage and Equation 3.5 for the second.

$$\mathbf{a}_{\text{superheavy}} = \left[\tau, \theta^g, \tau_{RCS}, \theta_{left}^{gf}, \theta_{right}^{gf} \right] \tag{3.4}$$

$$\mathbf{a}_{\text{starship}} = \left[\tau, \theta^g, \tau_{RCS}, \theta_{upper}^{flap}, \theta_{lower}^{flap} \right] \tag{3.5}$$

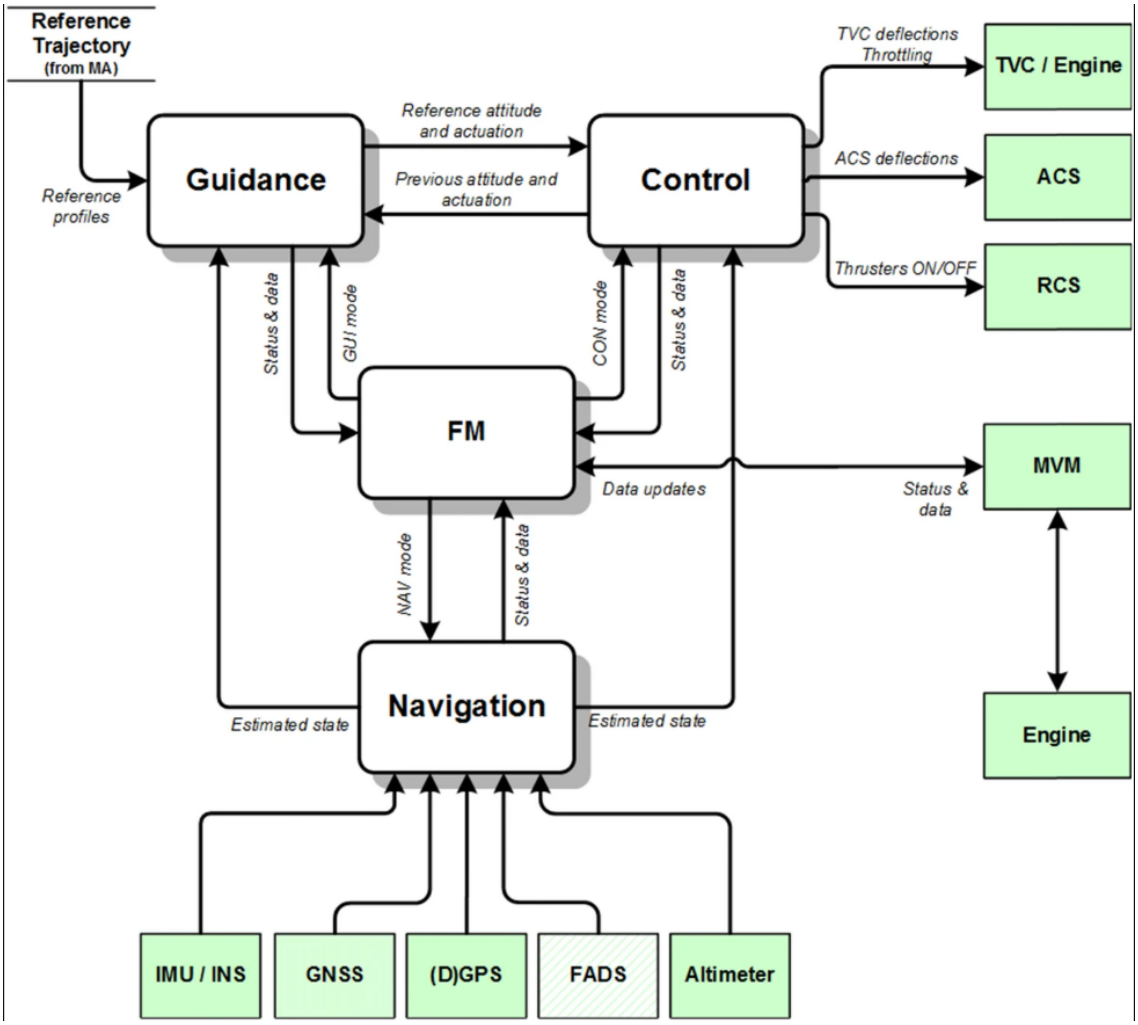


Figure 3.3: RETALT1 rocket's GNC architecture for recovery [7]

3.1.2. Flight phases

As discussed in the previous section, the launcher has various flight phases, which can be viewed as periods of flight characterised by specific aims and atmospheric conditions. For a two stage launch vehicle the launch starts with a **vertical rising** phase where the launcher travels vertically upwards to clear the launch tower, before the launch vehicle performs a pitch over maneuver called a **gravity turn** to build the horizontal velocity required for orbital insertion [53]. The gravity turn may require throttling to avoid breaching the launcher's maximum dynamic pressure in the denser atmospheric layers. Furthermore, aerodynamic forces are directly proportional to dynamic pressure. Thus, a low angle of attack is desired at high dynamic pressures to minimise aerodynamic disturbances and stress aboard the vehicle.

After the first stage ascent burn, stage separation will occur. In *cold-staging*, the fairing between the stages separates, and both stages coast for a few seconds before the second stage's engines ignite. However, Starship performs *hot-staging* to ignite the second stage's engines before fairing separation. This is beneficial as it reduces gravity losses and can provide the first stage with a pitch moment to assist the flip-over manoeuvre.

After separation, the second stage performs a burn to direct it into orbit, using TVC to curve its thrust vector and the RCS to counteract minor angular disturbances. Following the burn, it will coast to the desired semi-major axis before a circularisation burn occurs. Trajectory optimisation for this second-stage burn can utilise Pontryagin's Maximum Principle to compute the thruster vector and burn time [54].

Meanwhile, after separation, the first stage begins its landing procedure by performing a **flip-over manoeuvre** to rotate the stage to a near-horizontal position, setting up for the **boostback burn** to cancel the horizontal velocity and direct it back to the landing pad. The TVC performs the flip-over manoeuvre on a limited number of the gimballed engines, which are then fired at maximum throttle to perform the boostback burn, which cuts off the engines once a terminal horizontal velocity has been reached.

The direction of the flip-over manoeuvre depends on the landing scenario. Figure 3.4 depicts these two scenarios, with scenario A showing Return To Launch Site (RTLS) and B showing Away From Launch Site (AFLS) landings. For SpaceX, AFLS landings involve landing the launch vehicle's first stage on an autonomous drone ship downrange, and RTLS back at a landing pad near the launch site of origin.

Trading off, RTLS has lower operational costs and is logistically simpler, as no drone ship needs to be deployed for recovery; this also results in faster refurbishment time, as the booster is closer to the processing facilities. Secondly, there is less weather dependence as ocean conditions affect the feasibility of an AFLS landing occurring successfully. However, AFLS requires less fuel as the boostback burn needs to correct less of the horizontal velocity and thus allows for increased payload capacity. For this scenario, RTLS will be considered due to its ability to allow for rapid reusability, one of the primary motivations for reusable launch vehicles.

A **high-altitude ballistic arc** takes place after the flip over. The primary control authority for orientation comes from the RCS as the stage is flying through the upper atmosphere, where the ACS is less effective.

As the stage descends into the denser atmosphere, the dynamic pressure increases due to the higher density and vertical speed. A burn then occurs to slow the rocket down to avoid dynamic pressure limits, before decelerating for landing. Also, with higher dynamic pressure, the ACS becomes more effective than the RCS and is used for orientation control.

For the *SuperHeavy* booster, this burn is referred to as the **landing burn**, and the engines continue to burn until landing[4][5]. However, for some mission scenairos, like RETALT1's mission profile presented in Figure 3.4, the burn is split into two burns with a **re-entry burn** to avoid the dynamic pressure limits followed by an aerodynamic phase where the ACS maneuvers the stage, before a final **landing burn**.

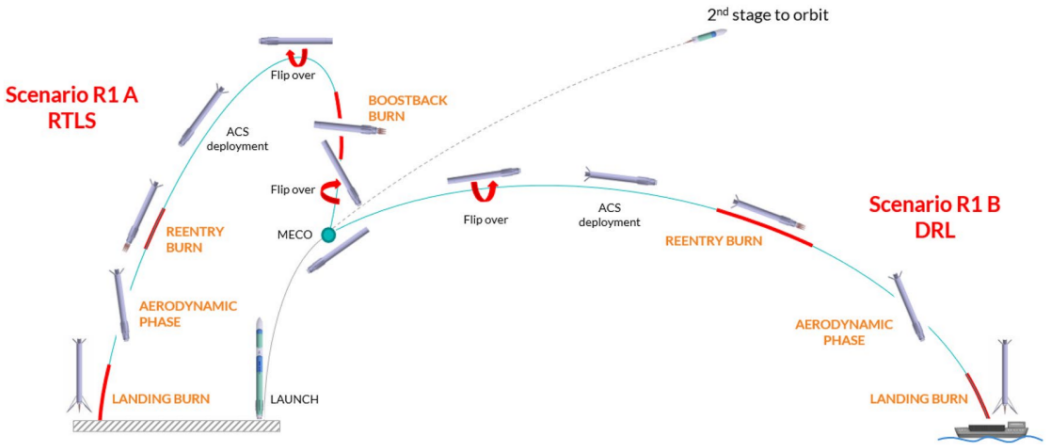


Figure 3.4: RETALT1 return mission concept [55]

3.1.3. Descent trajectory optimisation methods

Optimal descent trajectory generation for a launch vehicle involves balancing competing objectives, resulting in a multiple-objective optimisation problem (MOOP). First, this MOOP aims to minimise fuel

[4] https://www.spacex.com/launches/mission/?missionId=starship-flight-2. Accessed: 23/06/2025.
[5] https://www.spacex.com/launches/mission/?missionId=starship-flight-3. Accessed: 23/06/2025.

consumption while providing feasibility to the solution. Secondly, the solution must manage strict constraints throughout the flight on variables such as dynamic pressure and terminal conditions at the landing site. Several optimisation frameworks have been used for the powered descent problem, each with differing solution optimality, robustness and computational requirements.

This section reviews three key approaches found in the literature: first, Model Predictive Control (MPC), which provides an optimal solution and updates with feedback to balance trajectory deviation and actuator usage, allowing for fuel consumption to be minimised. Secondly, the widely applied lossless convexification and its extensions for powered descent provide reliable and efficient solutions, and are proven upon the Falcon 9[6]. Finally, a DRL approach is presented, along with the author's (Gaudet) argumentation on why this method was a valid choice.

Model Predictive Control

MPC is part of the optimal control family and utilises constrained optimisation at each time step. Given the current state, the future states can be predicted over a finite horizon through the predicted control inputs [56]. The cost function of Equation 3.6 is minimised to determine optimal inputs, while the states and inputs can be constrained for feasibility. Here, Q and R serve as cost weights for state deviation and control effort, respectively, setting the costs associated with state errors and actuation. The process of a MPC is shown in Figure 3.5, containing a predictor, plant (simulation model), cost function, constraints and an optimiser to track a reference.

$$J = \sum_{k=0}^{N-1} [(\mathbf{x}_{t+k} - \mathbf{x}_{\text{ref}})^T \cdot Q \cdot (\mathbf{x}_{t+k} - \mathbf{x}_{\text{ref}}) + \mathbf{u}_{t+k}^T \cdot R \cdot \mathbf{u}_{t+k}] \tag{3.6}$$

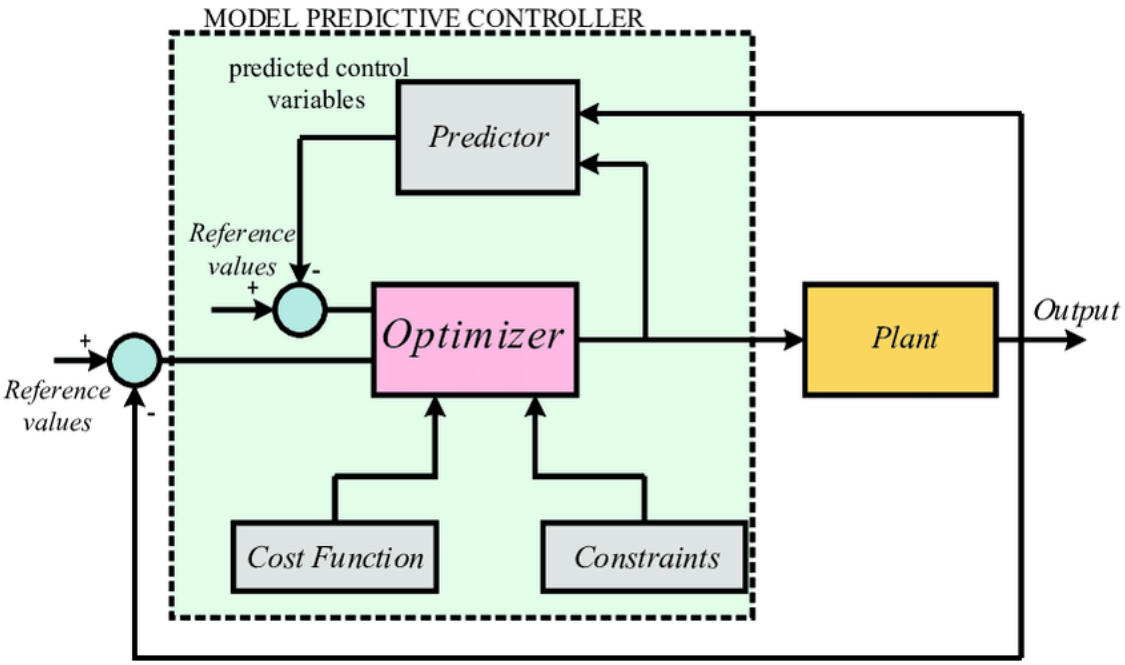


Figure 3.5: A schematic of how an MPC works [57].

Convex MPC has been introduced for rocket landing, to leverage its ability to solve optimisation problems while satisfying constraints [58]. In terms of rocket landing, it is constrained by dynamic pressure, landing conditions, and thermal heating, and must optimise for minimal fuel consumption. In the convex MPC method, the system is repeatedly linearised and non-convex constraints are convexified, allowing a convex optimiser to be used. Furthermore, MPC has been used to minimise the usage of tracking error controls [58].

[6] `https://ee.stanford.edu/news/2021/jan/stephen-boyd-cvxgen-guides-spacex-falcon?`. Accessed 03-06-2025.

MPC provides a well-formulated solution for the rocket landing problem due to its ability to adapt to uncertainties in real-time as a finite-horizon optimisation problem. Secondly, it ensures optimal performance under constraints by minimising the tracking error and control effort. However, the finite-time horizon can impact its long-term trajectory planning, resulting in a trade-off between computation time and solution optimality.

Algorithm 1 Model Predictive Control (MPC) for Linear State-Space System

1: **Input:** Initial state $\mathbf{x}_0$, prediction horizon N, reference state $\mathbf{x}_{\text{ref}}$, feasible set for states $\mathcal{X}$, feasible set for controls $\mathcal{U}$, state transition matrix A, control input matrix B, state weighting matrix Q, control weighting matrix R.
2: **for** each time step $t = 0, 1, 2, \ldots$ **do**
3: Measure or estimate current state $\mathbf{x}_t$
4: **Solve the following optimization problem:**

$$\begin{aligned} \min_{\{\mathbf{u}_t,\ldots,\mathbf{u}_{t+N-1}\}} \quad & \sum_{k=0}^{N-1} \left[(\mathbf{x}_{t+k} - \mathbf{x}_{\text{ref}})^T Q(\mathbf{x}_{t+k} - \mathbf{x}_{\text{ref}}) + \mathbf{u}_{t+k}^T R \mathbf{u}_{t+k}\right] \\ \text{subject to:} \quad & \mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k \\ & \mathbf{x}_k \in \mathcal{X}, \quad \mathbf{u}_k \in \mathcal{U}, \quad \forall k = t, \ldots, t+N-1 \end{aligned}$$

5: Obtain optimal control sequence $\{\mathbf{u}_t^*, \ldots, \mathbf{u}_{t+N-1}^*\}$
6: Apply the first control input $\mathbf{u}_t^*$ to the system
7: Advance to the next time step

Lossless Convex Optimisation

Lossless convexification reformulates non-convex optimal control problems to convex problems where the relaxed convex problem's solution is also a solution of the non-convex problem, guaranteeing global optimality and feasibility. Lossless convex optimisation has been applied to the concave powered descent problem in the Martian environment [3], [5]. Minimum thrust and thrust vector direction were convexified so that the optimal solution of the convex problem was the same as the concave problem. A lossless solution occurs when the solution to the relaxed convex problem is equivalent in optimality to that of the nonconvex (concave) problem. A benefit of lossless convex optimisation is that interior point solution methods can guarantee solutions within a bounded time [14].

Algorithm 2 Lossless Convexification

1: **Input:** Non-convex optimal control problem
2: Reformulate non-convex constraints as convex relaxations.
3: Formulate the convex relaxation of the original problem.
4: Solve the convex problem with a convex optimisation solver.
5: **if** solution satisfies original non-convex constraints **then**
6: **return** Solution globally optimal.
7: **else**
8: **return** Solution cannot be found.

A lumped mass rocket model with three DoF translation motion and rotational motion considered uncoupled had lossless convexification successfully applied to solve the powered descent problem on Mars [3]. The thrust and cant angles were assumed to be uniform over the thrusters. Position and velocity were used as the decision variables. The objective was to minimise fuel consumption, which is equivalent to thrust minimisation. A simple introduction of a slack variable on a nonconvex thrust constraint $T_{\text{min}} \leq ||T(t)|| \leq T_2, \forall t \in [0, t_f]$ allowed the problem to be formulated as a Second Order Cone Programming (SOCP) problem. The constraints were convexified and formulated as an SOCP problem, then discretised into a finite-dimensional problem so optimisation solvers can handle it. This approximated the control inputs as a linear combination of basis functions.

SOCP is a class of convex optimisation problems with constraints including second-order (Lorentz) cones [59]. This is shown in Equation 3.7, with f as the cost weighting, x the decision variables, m the number of constraints, and A, b, c, d, F, g are constraint parameters[7].

$$\begin{aligned} &\text{minimise:} f^T \cdot x \\ &\text{subject to:} ||A_i \cdot x + b_i||_2 \leq c_i^T + d_i, i = 1, ..., m \\ &\qquad\qquad F \cdot x = g \end{aligned} \tag{3.7}$$

Blackmore extended the study to include minimum-landing error approach. The aim was to minimise the final distance to the landing site when there was no feasible trajectory due to constraints. First, the algorithm assumes it is feasible and solves the problem as before; if it is found to be infeasible, a new trajectory is found to minimise landing error [11]. Later, Açıkmeşe and Blackmore extended this to include constraints on thrust pointing [5].

Successive Convexification (SCvx) was introduced as an extension to this algorithm through applying the trust region method differently to nonlinear dynamics and non-convex constraints. Successive convexification is an iterative approach to handling non-convex problems with non-convex state constraints and nonlinear dynamics, achieved by repeatedly linearising and convexifying around the current solution. This allows the problem to be solved as a sequence of convex subproblems, with the optimality guarantees of lossless convexification. This approach has been successfully applied to the planetary landing problem with collision avoidance constraints [60]. This algorithm iteratively convexifies the problem before applying a "*project-and-linearise*" procedure to transform the non-convex constraints (state and dynamic) into convex approximations, solving the problem as a sequence of smaller problems. In more detail, the current solution was iteratively projected onto the feasible set from the non-convex constraints before being linearised around the constraints. Finally, it was shown to converge quickly than lossless convexification on this problem. SCvx has been used to extend the work of Blackmore and Açıkmeşe to incorporate aerodynamic drag [12].

Algorithm 3 Successive Convexification

1: **Input:** Non-convex optimal control problem
2: Initialise trajectory: the sequence of states over time from the initial to final state (e.g.) position, velocity.
3: Initialise controls: the sequence of control inputs over time (e.g.) throttle command.
4: **repeat**
5: Linearise or convexify dynamics and constraints around the current solution.
6: Formulate and solve the resulting convex subproblem.
7: Update trajectory and controls.

Szmuk, Reynolds and Açíkmeşe presented a real-time feasible SCvx formulation for a generalised free-final-time 6 DoF powered descent guidance problem [13]. Key innovations include a free-ignition-time modification, a "tractable" aerodynamics model, and a continuous state-triggered constraint framework. The free-ignition-time modification found the optimal engine ignition time following a coasting phase. While the continuous state-triggered constraint framework automatically enabled constraints based on the system's state, these were:

- An "angle of attack state-triggered constraint": the angle of attack is limited at large dynamic pressures.
- A "field of view state-triggered constraint": limits the line of sight angle to the landing sight at a certain distance from the landing site.

The resulting non-convex optimal control problem was transformed into a sequence of convex SOCP subproblems, which were solved using SCvx, utilising modified trust regions to address artificial unboundedness and virtual control to address artificial infeasibility. Virtual control is an artificial control input added to the dynamics for each subproblem, allowing the optimiser to "violate" the linearised

[7] Note that the variables used here are not included in the nomenclature.

dynamics when necessary to prevent infeasibility due to linearization errors or underperforming initial guesses. A heavily weighted penalty term was added to the cost function to push virtual control terms to zero upon convergence. Artificial unboundeness refers to ”linearised constraints” permitting ”the cost of a subproblem to be minimised indefinitely”; as a result, a soft quadratic trust region was added to the cost function. Additionally, they note this ensured that the solve step did not deviate significantly from the reference trajectory of the propagation step.

Shen, Zhou and Yu combined convex optimisation with deep neural networks to generate their initial guesses; this reduced computational demands to a 40.8% computational time with 99.1% of test cases reaching real-time requirements [61]. This provided a proof-of-concept for data-driven control techniques to emulate the optimal behaviours of a convex optimiser, albeit only for the initial condition and not over the entire trajectory.

Deep Reinforcement Learning

Gaudet, Linares and Furfaro applied DRL to a 6 DoF planetary landing problem on Mars. PPO learns a policy to map the vehicle's state to thrust commands of separate engines [14]. They argued that DRL bypasses the need for individual guidance and control modules, which convex optimisation requires due to simplifications in linear models and the convexification of constraints. DRL thus can provide an end-to-end optimisation solution. Rewards were shaped through a reference velocity profile to guide the lander towards a soft landing within a five-meter ellipse and at a speed of less than $2m/s$. The authors argued that the velocity target was required, as only a reward for pinpointing landing would cause the policy to never meet the landing target in a feasible number of iterations. Alternatively, inverse reinforcement learning could be used to learn a reward function; however, it was deemed too time-intensive [62], [63]. Terminal rewards incentivised upright attitude and minimal rotational velocity at touchdown. Additionally, a novel improvement was separate discount rates for terminal and shaping rewards to improve learning stability.

Pinpoint accuracy was achieved under Monte Carlo tests, despite disturbances from sensor noise, wind disturbances, and mass variation. Fuel consumption was 4% higher than 3 DoF optimal control (the GPOPS[8] algorithm was used), acknowledged due to a suboptimal target velocity. Additionally, an 800m divert manoeuvre was tested, resulting in a 30kg average increase in fuel, with no impact on overall performance.

3.2. Policy optimisation families

The powered descent problem presents a challenging guidance and control problem. Current control methods for powered descent employ convex optimisation with SOCP solvers or an MPC to control the trajectory. Previously, the real-time feasibility of these algorithms was a key constraint; however, Falcon 9 utilises these algorithms with fast code generation to provide a validated real-world use case. Alternatively, policy optimisation methods can provide a real-time, feasible, and integrated guidance and control solution by learning an optimal policy directly on non-convex constraints and non-linear dynamics. For the control part of GNC systems, traditional control methods, such as Linear Quadratic Regulators (LQR), PID controllers, and feed-forward models, are employed to provide reference tracking and reject disturbances. This thesis aims to investigate the application of policy optimisation methods over current methods for the powered descent problem. Optimal control policies can be learnt through interaction with a simulation environment, without the need for carefully formulated optimisation problems or convexification of constraints.

As there are many policy optimisation methods available, decisions can be made regarding which *family* they are from to reduce the scope of the literature study. These choices are shown in Figure 3.6.

- The algorithm family (Subsection 3.2.1) decides the best policy optimisation group for the study.
- The learning strategy (Subsection 3.2.2) determines whether learning will occur online or offline, thereby deciding whether updates will be made within flight or on a fixed dataset.
- Policy learning (Subsection 3.2.3) decides whether to use off or on-policy learning, referring to whether to update the policy experiences generated by the current policy.

[8] `https://gpops2.com/`. Accessed: 23/06/2025.

- The control has the option to use a model to guide decision-making through model-based methods. Alternatively, model-free methods learn a policy through environment interaction without using a learnt model to change its policy. Model-based methods utilise a learnt or known environment model to plan future outcomes, guiding the policy update. Alternatively, model-free methods learn a policy directly through interaction with the environment. This was covered in Subsection 3.2.4.

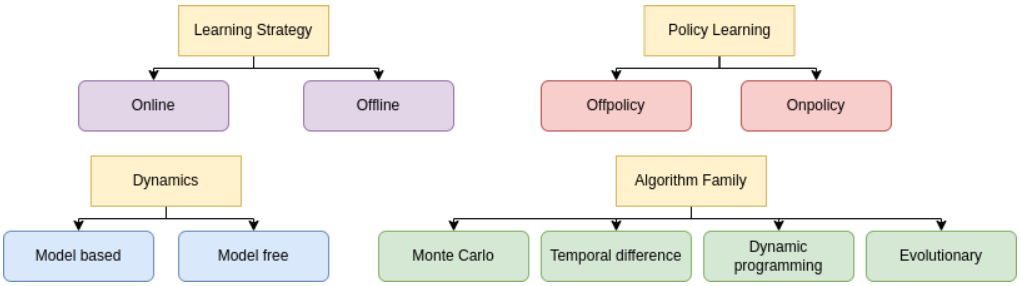


Figure 3.6: Key decisions for selecting a policy optimisation method.

3.2.1. Algorithm families

Policy optimisation methods have been divided into four groups:

- Monte Carlo methods: average returns over complete episodes to update the policy [19].
- TD methods: use incremental learning to update value estimates from differences in actual and predicted rewards.
- DP methods: require complete knowledge of the environment, but are often impractical for large systems due to high computational requirements [64].
- Evolutionary methods: are gradient-free methods which optimise policies through algorithms inspired by nature. However, they can be sample-inefficient due to a large number of independent policy searches required to cover the chosen search space.

These methods can be combined with neural networks to serve as function approximators, effectively handling complex and continuous problems [18]. TD methods, encompassing RL, were chosen over Monte Carlo and DP methods due to increased sample efficiency and scalability to significant problems. Secondly, evolutionary methods are chosen as a non-gradient method to be compared to RL[9].

3.2.2. Online vs Offline learning

Policy learning can be divided into online and offline learning [19]. Offline learning uses pre-collected data or simulations to train policies, whereas online learning can dynamically adapt to the environment in real-time. Offline learning enables training before deployment, which is particularly practical in safety-critical applications, such as launch vehicle landings. Alternatively, online learning can occur within a simulation, allowing policy to be updated in real-time before deployment.

Online learning within simulations has to be used as the policy will be initially insufficient to perform powered descent without the use of an expert solution generating a dataset.

3.2.3. Off-policy vs On-policy methods

The on/off policy nature determines how the agent uses environment transitions to learn optimal policies. On-policy methods only update on data generated by the current policy [19]. For example, Q-learning updates the action-value function with the maximum expected future rewards from the next state, independent of the current policy, allowing the agent to learn the optimal policy independent from its own actions [23]. A key benefit of this is that the policy continues to learn from exploratory or suboptimal transitions, increasing its ability to generalise.

[9] Evolutionary methods are independent of the on/offline and off/on policy decisions.

Alternatively, off-policy methods utilise experiences generated from different policies or expert solutions, stored in a replay buffer. For example, State-Action-Reward-State-Action (SARSA), where the action-value function is updated by the agent for each action taken using the received reward and value of the next state-action pair chosen by the current policy [19]. This allows the agent to learn from only its own transitions with the environment directly.

Off-policy methods are more sample efficient as they can reuse previous transitions through a buffer and, as such, are chosen [65].

3.2.4. Model-free vs Model-based methods

Model-free control methods have emerged as promising candidates to learn an optimal policy for launch vehicle landing. For example, PPO, a model-free RL algorithm, has successfully learnt policies for Martian powered descent [14]. Model-free methods directly learn a policy through interaction with the environment without requiring a learnt or given high-fidelity mathematical model of the underlying dynamics. Furthermore, PPO has been utilised with a ”random annealing jump start” to improve the tractability of applying RL to the powered descent problem [66]. Alternatively, model-based methods can roll out a trajectory to support the planning capabilities of policy, which can benefit tasks with long time horizons.

Due to proven examples of model-free RL on the powered descent problem, and the lack of needing to learn a dynamics model, **model-free** methods are chosen over model-based methods. However, future work could compare model-based and model-free methods on this problem.

3.3. Reinforcement Learning

RL is a topic in machine learning where *agents* are trained to learn decisions through environment interaction, as visualised in Figure 3.7. A *reward function* indicates the solution’s optimality, leading the agent to converge to an optimal policy, where the actions that maximise reward are known for each state. The extension of deep reinforcement learning utilises a neural network as a function approximator to extend RL’s feasibility to larger problems.

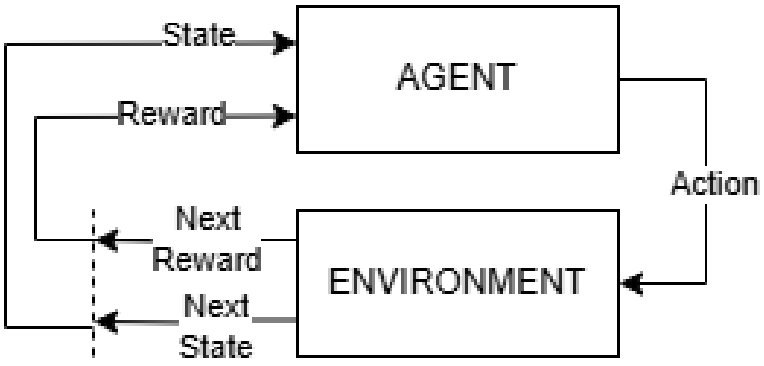


Figure 3.7: Reinforcement Learning Model

This section aims to explain the theory behind model-free, off-policy, and online learning RL methods, and to survey techniques that can be used to help stabilise and speed up learning. First, the reader is provided with foundations in DRL through the presentation of Q-learning and Deep Q-learning in Subsection 3.3.1. To facilitate off-policy learning, various types of replay buffers are documented in Subsection 3.3.2. As exploration is key to unlocking new control strategies for the agent, a literature review of exploration strategies relevant for this problem is conducted in Subsection 3.3.3. Following this, the algorithms used for learning a problem of our specification are provided in Subsection 3.3.4. Finally, techniques to aid learning stability and efficiency are presented in Subsection 3.3.5.

3.3.1. Q-learning

A value-based algorithm with an agent learning values of action-state pairs to choose the resulting actions to maximise rewards [23]. This is an off-policy algorithm, so it learns the optimal policy inde-

pendent of the agent's actions, allowing for model-free RL, as the agent learns the optimal action-value function $Q^*(s,a)$ through interacting with the environment, providing the maximum expected rewards $R(s_t,a_t)$ from acting a_t in the state s_t. The function is updated through Equation 3.8, where the parameters of the Q-network are periodically updated to match those of the target network. Here, the discount γ determines the importance of future rewards, and τ determines how much new information updates the current Q value.

$$Q(s_t,a_t;\theta) \leftarrow Q(s_t,a_t;\theta) + \alpha[R(s_t,a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\theta^-) - Q(s_t,a_t;\theta)] \tag{3.8}$$

This section then discusses the extension of the Q-learning algorithm. First, Deep Q-learning (DQN), which leverages neural networks acting as function approximators to extend Q-learning to high dimensional problems. Then, duelling Q-networks, which help in scenarios with sparse rewards, and finally double Q-learning is shown to help reduce value function overestimation bias.

Deep Q-learning

DQN was developed to learn how to play Atari 2600 games [18], [67]. DQN allows learning of more complex environments, overcoming Q-learning's "curse of dimensionality" through generalisation over similar states. Standard Q-learning (as in Equation 3.8) stores Q-functions in a Q-table, which becomes infeasible with high dimensional problems, as a result DQN proposed utilising a neural network as a function approximator. The neural network's weights are updated through a Mean-Squared Error (MSE) loss function of Equation 3.9. Furthermore, target networks are used to periodically update the parameters of the online network at a set number of steps, thereby reducing oscillations and producing a more stable loss. Finally, a replay buffer is utilised to improve sample efficiency. The pseudo-code is shown in Algorithm 4.

$$\mathcal{L}(\theta) = \mathbb{E}[(R(s_t,a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1},a_{t+1};\theta^-) - Q(s_t,a_t;\theta))^2] \tag{3.9}$$

Dueling Q-networks

When learning, certain actions may yield no rewards, even though they are beneficial. For instance, when playing football, scoring a goal earns you a point; however, no points are gained for tackling the opposition or providing an excellent cross into the box, although these actions are encouraged. To encourage this type of behaviour in places where actions have less effect on the outcome, a duelling network architecture was proposed [68]. In this network, the state value and action advantage are separated into a value stream $V(s)$ showing the state value and an advantage stream $A(s,a)$ giving the advantage of the action. As a result, the Q-Network update rule is changed to Equation 3.10.

$$Q(s,a;\theta,\theta^A,\theta^{V_d}) = V_d(s;\theta,\theta^{V_d}) + \left(A(s_t,a_t;\theta,\theta^A) - \frac{1}{|\mathcal{A}|} \sum_{a_{t+1}} A(s_t,a_{t+1};\theta,\theta^A) \right) \tag{3.10}$$

Double Q-learning

Standard DQNs tend to be biased towards over-optimistic value estimations because the same Q-network both selects and evaluates the action. For instance, any random overestimation in Q-values is likely to be chosen and reinforced, leading to a consistent positive bias in value estimations. As a result, it was proposed to decouple the selection and evaluation stages through a *Double Q-network*, which has been shown to give better estimates and performances in complex environments [69].

These two Q-networks use the update rule of Equation 3.11 where the leading network, parameters θ, is updated each iteration for action a_{t+1} selection, while the target network, parameters θ^- evaluates these actions. After a set interval, the target network is updated to lower the over-estimation bias.

$$Q(s_t,a_t;\theta) \leftarrow Q(s_t,a_t;\theta) + \alpha \left[R(s_t,a_t) + \gamma Q(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1},a_{t+1};\theta);\theta^-) - Q(s_t,a_t;\theta) \right] \tag{3.11}$$

Algorithm 4 Deep Q-Network (DQN) under an epsilon-greedy exploration strategy.

Input: Replay buffer capacity N, target network update frequency F, initial exploration rate ϵ, number of episodes M, batch size N_b, action space $\mathcal{A}$, learning rate α, epsilon decay coefficient κ.
Initialize online network $Q(\cdot;\theta)$ with random weights θ
Initialize target network $Q(\cdot;\theta^-)$ with weights $\theta^- \leftarrow \theta$
Initialize replay buffer $\mathcal{R}$ with capacity N
for episode $= 1, 2, \ldots, M$ **do**
  Reset environment and observe initial state s_0
  for $t = 0, 1, \ldots, T$ until terminal state **do**
    Sample random probability $p \sim \mathcal{U}(0,1)$
    if $p < \epsilon$ **then** ▷ Exploration
      $a_t \leftarrow$ random action from $\mathcal{A}$ ▷ Sample from action space
    else ▷ Exploitation
      $a_t \leftarrow \arg\max_a Q(s_t, a; \theta)$ ▷ Greedy action
    end if
    Execute a_t, observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{R}$
    Sample random batch (s_j, a_j, r_j, s_{j+1}) of size N_b from $\mathcal{R}$
    Compute target for each sample:

$$y_j = \begin{cases} r_j & \text{if } s_{j+1} \text{ is terminal} \\ r_j + \gamma \max_a Q(s_{j+1}, a; \theta^-) & \text{otherwise} \end{cases}$$

    Compute loss: $\mathcal{L}(\theta) = \frac{1}{N_b} \sum_{j=1}^{N_b} \left(y_j - Q(s_j, a_j; \theta)\right)^2$
    Update θ via gradient descent: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta)$
    if $t \% F = 0$ **then**
      Update target network: $\theta^- \leftarrow \theta$
    end if
    $s_t \leftarrow s_{t+1}$ ▷ Advance state
  end for
  Decay exploration rate: $\epsilon \leftarrow \epsilon \cdot \kappa$

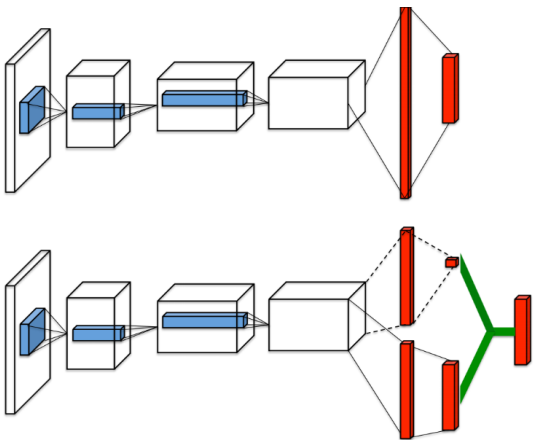


Figure 3.8: The top network is a standard Q-network, and the bottom network features a duelling architecture with two separate streams: the value stream and the advantage stream. [68]

3.3.2. Replay buffers

Off-policy methods have increased sample efficiency by allowing the reuse of experiences from previous transitions. For an environment like rocket landing, where the maneuverer can take minutes, this leads to an episode with thousands of steps when discretised by a modest 0.1s time step. Allowing the data to be reused provides the agent with a broader range of experiences to update from, as a result of its exploration. The *replay buffer* stores the transition data so that it can be reused later.

The first instance of the replay buffer was the **uniform replay buffer** used to play Atari games [18], here transitions are stored in a First-In-First-Out manner with all samples having equal probability of being sampled to provide diverse training samples. The buffer stores the state, action, reward, and next state for random batches of experiences to be sampled during training, breaking the correlations of consecutive episodes.

Prioritised Experience Replay (PER) was then introduced to take actions with a better learning potential through sampling via their TD error [70]. TD measures the prediction and target networks' Q-value differences, as shown in Equation 3.12. The immediate reward is summed with the future rewards (when active) and subtracted from the current Q-value estimate. A small constant is added to get the priority, with 'i' being the transition; this computes the probability $P(i)$. The hyperparameter α_{per} sets the importance of sampling previous transitions with a higher reward probability. The weights w_i are updated through *importance sampling*, where the hyperparameter β determines the correction, balancing faster learning and bias correction. The pseudo code for this buffer is shown in Chapter 2.

$$\begin{aligned} \delta =& |R(s_t, a_t) + (1-d) \cdot \gamma \cdot \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^-) - Q(s_t, a_t; \theta)| \\ p_i =& (\delta_i + 10^{-5})^{\alpha_{per}} \\ P(i) =& \frac{p_i}{\sum_k p_k} \\ w_i =& \left(\frac{1}{N \cdot P(i)}\right)^{\beta} \\ \bar{w}_i =& \frac{w_i}{\max_j w_j} \end{aligned} \tag{3.12}$$

Hindsight Experience Replay (HER) is a goal-oriented replay buffer where an episode is replayed with a different goal the agent was trying to achieve. This replay buffer enhances learning efficiency for environments with sparse or delayed rewards. HER takes the states from failed episodes as alternative goals to generate valuable strategies for successful experiences. For example, suppose a robotic arm places an object in a different position. In that case, HER can leverage this as a new goal, leveraging failed trajectories to create additional data and improve sample efficiency [71]. The pseudocode is shown in Algorithm 5.

Algorithm 5 Hindsight Experience Replay (HER)

Input: replay buffer $\mathcal{R}$, number of episodes M, hindsight count k, goal-sampling function $\mathrm{SampleHindsightGoal}$
for episode $= 1 \dots M$ **do**
    Initialise empty episode memory $\mathcal{E}$ ▷ re-initialise per episode
    Sample initial state s_0 and goal g
    for $t = 0 \dots T$ until termination **do**
        Select action $a_t \sim \pi(s_t, g)$
        Execute a_t, observe s_{t+1} and $r_t = R(s_t, a_t, s_{t+1}; g)$
        Append $(s_t, a_t, r_t, s_{t+1}, g)$ to $\mathcal{E}$
        $s_t \leftarrow s_{t+1}$
    end for
    for all $(s, a, r, s', g) \in \mathcal{E}$ **do**
        Store (s, a, r, s', g) in $\mathcal{R}$ ▷ actual transition
        for $i = 1 \dots k$ **do**
            $g' \leftarrow \mathrm{SampleHindsightGoal}(\mathcal{E})$
            $r' \leftarrow R(s, a, s'; g')$ ▷ hindsight reward
            Store (s, a, r', s', g') in $\mathcal{R}$ ▷ hindsight transition
        end for
    end for
end for=0

3.3.3. Exploration strategies

Exploration strategies balance the sensitive exploration and exploitation trade-off. Exploration is necessary, especially in the early stages of learning a new part of the environment, to allow the agent to uncover new experiences that maximise the Q-value. Meanwhile, exploitation is also needed to force the agent to converge by taking the path with the maximum Q-value at that point.

Exploration strategies can be divided into two use cases: discrete spaces and continuous spaces. For rocket landing, the state space (e.g., position, velocity, orientation) and the action space (e.g., thrust levels, gimbal angles) are continuous; therefore, only these continuous strategies are considered.

Action noise-based exploration is an efficient way to encourage exploration in continuous space by adding noise.

- *Gaussian Noise*: The exploration policy is created by adding random noise to the actor's policy. This is used in several actor-critic RL configurations, such as the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm, which is described in Equation 3.3.4.
- *Ornstein-Uhlenbeck (OU) noise*: is used to temporarily correlate noise. Lillicrap et al. utilised this approach to enhance their exploration efficiency, demonstrating its effectiveness for "physical control problems with inertia"[72].
- *Adaptive Noise Scaling*: can be combined with one of the methods above to balance exploration and exploitation during training. Here, the noise is scaled automatically, like in SAC.

Lemma. Gaussian and OU noise

Exploration in deterministic reinforcement learning methods, such as DDPG or TD3 (explained later in Subsection 3.3.4), requires the injection of external noise into the policy's action space to discover new and previously unseen solutions. Two common types of noise processes are Gaussian and OU noise.

Gaussian noise is temporally uncorrelated and drawn independently at each timestep from a normal distribution. This noise is simple and suitable for environments where smoothness in action evolution over time is not critical.

$$\epsilon_t \sim \mathcal{N}(0, \sigma^2)$$

Ornstein–Uhlenbeck noise introduces temporal correlation by modelling the stochasticity as a mean-reverting process, meaning that the stochastic process tends to revert towards a long-term mean. This mean-reverting process temporally smooths noise and is an idea applicable to physical control tasks with momentum, such as rocket landing, where a sudden change in action can lead to instability.. The correlation across timesteps simulates inertia, yielding more natural and consistent exploration trajectories.

$$\epsilon_{t+1} = \epsilon_t + \theta(\mu - \epsilon_t)\Delta t + \sigma\sqrt{\Delta t}\mathcal{N}(0, 1)$$

Gaussian noise is simpler to implement than OU noise, however OU noise encourages smoother transitions in systems with momentum, like physical control tasks.

Parameter Space Noise (PSN) was introduced to overcome the downfall of the action space noise of inconsistent exploratory behaviours due to a fixed state always giving different actions [73], [74]. This can lead to problems such as early truncation, where at certain points in the episode, the action is sensitive to noise and requires precision. PSN adds noise through Equation 3.13 to perturb the policy network parameters at the start of each episode and maintain them fixed afterwards; this results in state-dependent actions, unlike Gaussian noise methods.

$$\tilde{\theta} = \theta + \mathcal{N}(0, \sigma^2 \cdot I) \tag{3.13}$$

PSN has been shown to offer good exploration and avoidance of premature convergence on continuous environments, providing significant benefits in environments with sparse rewards. Additionally, it outperformed Evolutionary Strategies (ES) [75].

In environments where extrinsic rewards are scarce, intrinsic rewards can enable the agent to explore more effectively; this can be achieved through an **Intrinsic Curiosity Module** (ICM) [76]. The ICM encourages the agent to explore through intrinsic rewards provided by the agent's own experience, motivating them to explore when extrinsic rewards aren't present.

An inverse model, which predicts the action $\hat{a}(s_t, s_{t+1})$, is trained via Equation 3.14 to minimise the loss between the actual and predicted actions. The forward model predicts the next state's features $\hat{\phi}_{s_{t+1}}(\phi_{s_t}, a_t)$, through minimisation in state feature prediction via Equation 3.15. The intrinsic reward, Equation 3.16, depends on the hyperparameter η_{ICM} acting as the scaling factor. The total resulting reward is the sum of the extrinsic and intrinsic rewards.

$$\mathcal{L}_I(\hat{a}_t, a_t; \theta^I) = -\log P(a_t | s_t, s_{t+1}; \theta^I) \tag{3.14}$$

$$\mathcal{L}_F(\phi(s_t), \hat{\phi}(s_{t+1}; \theta^F)) = \frac{1}{2}\|\phi(s_{t+1}) - \hat{\phi}(s_{t+1}; \theta^F)\|^2 \tag{3.15}$$

$$R_t^{\text{intrinsic}} = \eta_{ICM} \cdot \frac{1}{2}\|\phi(s_{t+1}) - \hat{\phi}(s_{t+1}; \theta^F)\|^2 \tag{3.16}$$

An alternative to ICM is **Random Network Distillation** (RND), where they argued it is simpler and more stable than ICM, through testing on Montezuma's Revenge [77]. This method works well for sparse

reward environments, allowing extrinsic and intrinsic rewards to be flexibly combined. Here, intrinsic rewards are based on a prediction error from a trainable ”prediction” network that approximates a fixed and randomly initialised ”target” network. The prediction network is trained to minimise the prediction error through Equation 3.17. The intrinsic reward can be set equal to the prediction network loss. As such, states explored less will incur a higher prediction error, and thus, the RND encourages them to be explored. This process is visualised in Figure 3.9. In essence, RND measures the novelty of a state by comparing the prediction error on a fixed random target network.

$$\mathcal{L}_{\text{predict}} = \frac{1}{2} \cdot ||y^{-} - \hat{y}||^2 = r_{\text{intrinsic}} \tag{3.17}$$

Noisy networks incorporate parametric noise into the network weights to enhance exploration efficiency via stochastic diversity of the agent's policy [78]. Here, Stochastic Gradient Descent (SGD) learns the noise scale, adapting it over time; for instance, less noise is present as the agent becomes more confident, decreasing exploration. The authors also demonstrated its compatibility with any RL architecture that employs SGD. Ultimately, they argued that noisy networks provide structured and adaptive exploration.

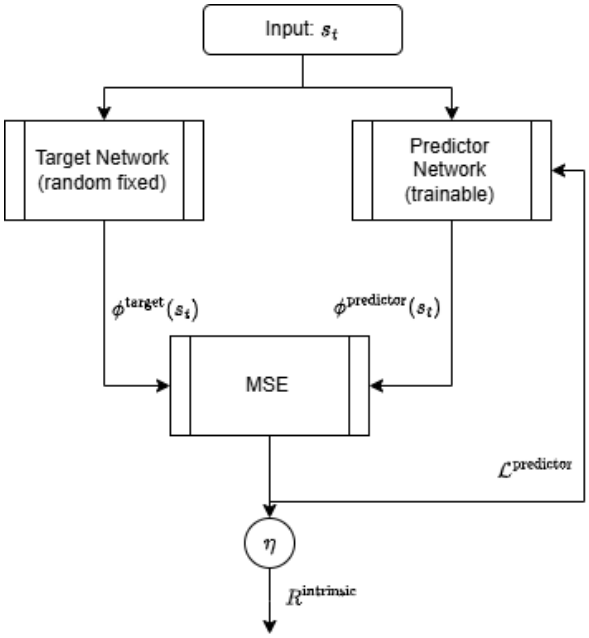


Figure 3.9: Flow diagram showing how RND works.

This section reviewed three types of action noise-based exploration, along with three additions to enhance the exploration of a neural network. Table 3.1 presents the strengths and weaknesses of each method discussed.

Table 3.1: Trade-offs between common exploration strategies in reinforcement learning.

Method	Strengths	Weaknesses	Best for
Gaussian Noise	Simple.	Constant state-independent noise can destabilise learning in environments with momentum or for states requiring fine-tuned actions.	Simple benchmark RL problems.
OU Noise	Best for systems with inertia and momentum.	More complex than Gaussian noise.	Continuous control problem with inertia.
Adaptive Noise	Automatically tunes noise to balance the exploration-exploitation trade-off.	Extra complexity.	Environments where optimal noise scale changes over time.
PSN	Consistent exploration.	Can introduce high variance with improperly scaled noise.	Escaping local optimum.
ICM	Drives exploration in sparse reward environments.	Extra parameters require learning.	Sparse or delayed reward tasks.
RND	Provides intrinsic rewards and simpler than ICM.	Extra trainable network.	Sparse or delayed reward tasks.
Noisy Networks	Adaptive PSN to provide scaled exploration.	Increased learning complexity and can introduce high variance.	State-dependent exploration and for escaping local optimums.

3.3.4. Offpolicy Continuous Online Methods

As reasoned in Section 3.2, an online off-policy method that works with a continuous action and observation space was chosen. The actor-critic framework is commonly used to learn an optimal policy for online and continuous domain problems. The actor learns the optimal policy by maximising the Q-value, which maximises the expected cumulative return from taking a given action in a given state. The actor, $\zeta(s_t;\theta^Z)$, learns the action for each state to maximise expected return through Equation 3.19, the mapping of state to action can be learnt through a neural network acting as a function approximator. The critic estimates the state-action value $Q(s_t,a_t;\theta^Q)$; the expected reward for taking an action in a given state, which the actor uses to update its policy. The critic is used instead of raw rewards to train the actor because it reduces the variance of policy gradient updates by providing smoothed action-value estimates.

The critic learns to minimise TD error. For example, Equation 3.18 illustrates the scenario for a critic, where the TD error is the difference between the predicted state-action pair and the critic's target network.

$$\mathcal{L}_Q(\theta^Q) = \frac{1}{N_b}\cdot\sum_{i=1}^{N_b}(R^i(s_t^i,a_t^i)+\gamma\cdot Q'(s_{t+1}^i,\zeta'(s_{t+1}^i;\theta^{\zeta'});\theta^{Q'})-Q(s_t^i,a_t^i;\theta^Q))^2 \tag{3.18}$$

$$\mathcal{L}_\zeta(\theta^\zeta) = -\frac{1}{N_b}\cdot\sum_{i=1}^{N_b}Q(s_t^i,\zeta(s_t^i;\theta^\zeta);\theta^Q) \tag{3.19}$$

The following subsections cover 5 of the main continuous online off-policy reinforcement learning algorithms found in literature. SAC and Maximum a posteriori Policy Optimisation (MPO) learn a stochastic

policy, while DDPG, Distributed Distributional Deterministic Policy Gradient (D4PG), and TD3 learn a deterministic policy.

Deep Deterministic Policy Gradient method

DQNs work with a discrete action space; as such, with a continuous control problem, the continuous outputs have to be discretised, leading to a massive action dimension, inhibiting learning. DDPG was presented as a solution to this problem through an extension to the Deterministic Policy Gradient (DPG) method ([72], [79]), allowing for a model-free off-policy online actor-critic algorithm with neural networks utilised as deep function approximators to learn a policy and state-action value function in continuous domains.

The actor provides deterministic actions with the critic approximating the resulting Q-value. Target networks and experience replay are utilised, like in DQNs, to stabilise training and improve sample efficiency. OU noise is used for action noise-based exploration, as explained in Subsection 3.3.3 [67].

To summarise, DDPG is an actor-critic framework with extensions being:

1. Action-space-based OU noise used for exploration.
2. A replay buffer $\mathcal{R}$ for off-policy learning, improving sample efficiency.
3. Target networks used to improve learning stability, through Polyak averaging Equation 3.20.

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \cdot \theta^{Q} + (1-\tau) \cdot \theta^{Q'} \\ \theta^{\zeta'} &\leftarrow \tau \cdot \theta^{\zeta} + (1-\tau) \cdot \theta^{\zeta'} \end{aligned} \tag{3.20}$$

Lemma. Target network updates via Polyak averaging

In deep reinforcement learning tasks, target networks are often used to stabilise learning by slowly changing the targets used for value estimation. Instead of directly copying the online network's weights, they are soft updated through *Polyak averaging*.
Let θ denote the parameters of the online network and $\theta^{'}$ the parameters of the corresponding target network. The Polyak update rule is defined as:

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

Where $\tau \in (0,1)$ is the averaging coefficient, a larger coefficient tracks the target network faster and can induce instability. Alternatively, a small coefficient gives stable but slower updates, which can enhance robustness.
Polyak averaging ensures that the parameters of the target network move slowly towards those of the online network, thereby reducing oscillations and divergence in value estimation and promoting learning stability.

Algorithm 6 Deep Deterministic Policy Gradient (DDPG)

Inputs: Critic learning rate α_Q, batch size N_b, actor learning rate α_ζ, Polyak averaging coefficient τ, discount rate γ.
Initialize:
  Actor network $\zeta(s;\theta^\zeta)$ with random weights θ^ζ
  Critic network $Q(s,a;\theta^Q)$ with random parameters θ^Q
  Target actor $\zeta'(s;\theta^{\zeta'}) \leftarrow \zeta(s;\theta^\zeta)$
  Target critic $Q'(s,a;\theta^{Q'}) \leftarrow Q(s,a;\theta^Q)$
  Replay buffer $\mathcal{R}$
  Ornstein-Uhlenbeck noise process $\mathcal{OU}$

for episode $= 1, 2, \ldots, M$ **do**
    Initialize environment, observe initial state s_0
    for $t = 0, 1, \ldots, T$ until terminal state **do**
      Select action $a_t = \zeta(s_t;\theta^\zeta) + \mathcal{OU}$
      Execute a_t, observe reward r_t and next state s_{t+1}
      Store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{R}$
      Sample random minibatch $(s_i, a_i, r_i, s_{i+1}, d_i)$ from $\mathcal{R}$
      Compute target action $a_{i+1} = \zeta'(s_{i+1};\theta^{\zeta'})$
      Compute target value $y_i = r_i + \gamma \cdot (1 - d_i) \cdot Q'(s_{i+1}, a_{i+1};\theta^{Q'})$
      Compute critic loss: $\mathcal{L}(\theta^Q) = \frac{1}{N_b}\sum_i (y_i - Q(s_i, a_i;\theta^Q))^2$
      Update critic: $\theta^Q \leftarrow \theta^Q - \alpha_Q \nabla_{\theta^Q}\mathcal{L}(\theta^Q)$
      Compute actor loss: $\mathcal{L}(\theta^\zeta) \leftarrow -\frac{1}{N_b}\sum_i Q(s_i, \zeta(s_i;\theta^\zeta);\theta^Q)$
      Update actor parameters: $\theta^\zeta \leftarrow \theta^\zeta - \alpha_\zeta \nabla_{\theta^\zeta}\mathcal{L}(\theta^\zeta)$
      Update target networks (Polyak averaging):
        $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
        $\theta^{\zeta'} \leftarrow \tau\theta^\zeta + (1-\tau)\theta^{\zeta'}$
      $s_t \leftarrow s_{t+1}$
    end for
end for

Distributed Distributional Deterministic Policy Gradients

DDPG was extended to D4PG, which includes a distributional critic, a PER buffer and N-step returns [80].

- A distributed framework is used, where multiple actors asynchronously generate experiences and populate a shared replay buffer, which is updated by a central learner to refine the policy.
- A *distributional critic* is used, essentially instead of modelling a scalar $Q(s,a;\theta^Q)$ it predicts a probability distribution $Q_Z(s,a;\theta^{Q_Z})$ over all possible returns, this distribution modelling works well for stochastic environments where differing Q values can come from different actions in the same state. The critic now outputs a probability distribution, leading to the use of Kullback-Leibler (KL) divergence instead of MSE as the loss function.
- N-steps are used to give the reward over a longer time horizon, which is beneficial for tasks where there are long-term dependencies, such as in rocket landing. The N-step reward equation is shown in Equation 3.21.

$$y_t^i = \left(\sum_{m=0}^{N_{\text{episode}}} \gamma^m \cdot R(a_m^i, s_m^i) \right) + \gamma^{N_{\text{episode}}} \cdot Q'_Z(s^i_{N_{\text{episode}}}, \zeta'(s^i_{N_{\text{episode}}};\theta^{\zeta'});\theta^{Q'}) \tag{3.21}$$

Lemma. KL divergence

The KL divergence is a measure of how one probability distribution P diverges from a second, reference distribution Q. For continuous distributions with probability densities $f_p(x)$ and $f_q(x)$, the KL divergence from Q to P is defined as:

$$\mathcal{D}_{\mathrm{KL}}(P \parallel Q) = \int f_p(x) \log \frac{f_p(x)}{f_q(x)}\, dx$$

The smaller the KL divergence, the more identical the distributions are.

Lemma. N-step returns

In traditional TD learning, the value estimate of a state is updated based on the immediate reward and the estimated discounted value of the next state. Equation 3.22 is the equation for calculated the 1-step return at time t ($G_t^{(1)}$) using the immediate reward (R_{t+1}) and the estimated value of the next state ($Q(s_{t+1}, a)$).

$$G_t^{(1)} = R_{t+1} + \gamma \cdot Q(s_{t+1}, a) \tag{3.22}$$

N-step returns extend this concept to look ahead n steps through *booststrapping* the action-value estimation. Equation 3.23 shows the generic formula for n-step rewards. N-step returns allows the agent to learn from a longer sequence of rewards before bootstrapping, giving it more information about long-term dependencies

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n Q(s_{t+n}, a)) \tag{3.23}$$

This formula changes when terminal states are within the trajectory length set for n-step returns (n), also known as the n-step horizon. The returns should only include rewards up to the terminal state, either a truncated or completed episode, with no bootstrapping or rewards after termination if step $t + k$ is a terminal state, where $k < n$, the n-step return becomes Equation 3.24.

$$G_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{k-1} R_{t+k} \tag{3.24}$$

Twin Delayed DDPG

Double Q-learning was incorporated into DDPG to reduce overestimation bias of the value function through a pair of independent critics and double Q-learning [81]. Through this approach, DDPG is run with two extensions:

1. Clipped Gaussian noise is used instead of OU for action space-based noise; clipped noise is chosen to prevent sharp action value changes.
2. Through the use of double Q-learning (dual critics), the loss functions for the critic have the ”target critic changed” to the minimum between the networks, ensuring the most conservative estimate. For the actor loss, only the primary Q-network is considered, ensuring it is based on a single Q-value estimate, which reduces noise and learning instability.

Soft Actor-Critic

SAC as an actor-critic algorithm which maximises both expected rewards and the policy’s entropy, encouraging exploration and thus leading to more robust policies [28]. TD3 uses Gaussian action space noise, whereas SAC’s actor outputs a Gaussian distribution $\zeta_S(s; \theta^\zeta, \theta^\mu, \theta^\sigma)$, with the subscript ”S” denoting it as a stochastic actor. This actor outputs $\mu(s_t; \theta^\mu)$ and $\sigma(s_t; \theta^\mu)$, which are converted to a noisy action through Equation 3.25. Compared to TD3, the main difference is that the noise distribution is a learned parameter providing *state-dependent exploration*.

$$\sigma(s; \theta^\mu) = e^{\log \sigma(s; \theta^\mu)}, \quad \tilde{a} = \mu(s; \theta^\mu) + \sigma(s; \theta^\mu) \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad a = tanh(\tilde{a}) \tag{3.25}$$

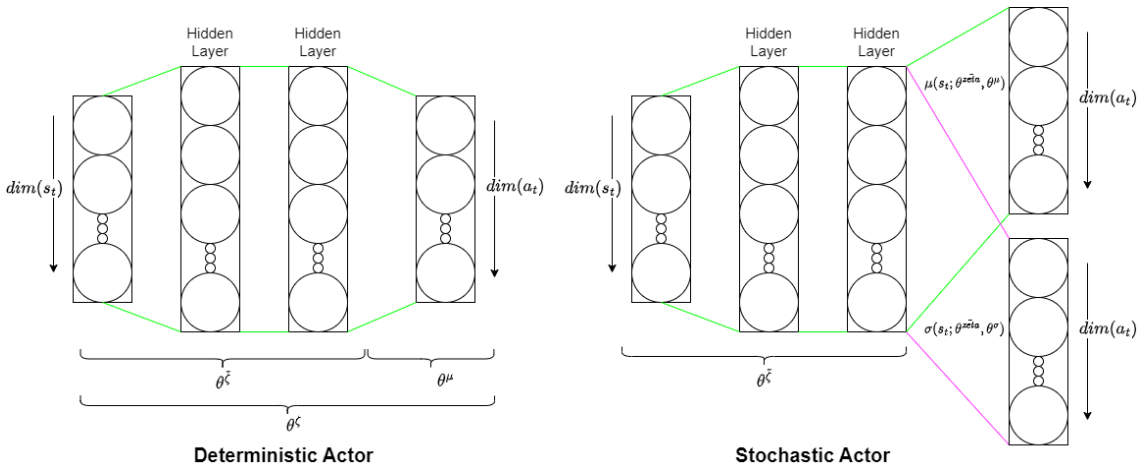


Figure 3.10: Diagram showing the differences in actor networks, for deterministic and stochastic.

SAC has an additional entropy term to control the state-dependent exploration. *Temperature* v updates to encourage an entropy to meet the target entropy, controlling the degree of randomness and traditionally set as $\mathcal{H}_{\text{target}} := -\dim(a_t)$. Equation 3.26 temperature is updated to encourage the negative Gaussian likelihood equal to the target entropy.

$$\mathcal{L}_{\log(\nu)} = \frac{1}{N_b} \cdot \sum_{i=1}^{N_b} -\log(\nu) \cdot \left(\log P(a_t^i|\mu, \sigma^2) + \mathcal{H}_{\text{target}} \right) \tag{3.26}$$

Lemma. Gaussian likelihood

The Gaussian likelihood is the probability density function of a normal distribution evaluated at a given point. For a random variable $x \in \mathbb{R}^d$, with mean $\mu \in \mathbb{R}^d$ and standard deviation $\sigma \in \mathbb{R}^d$, the log-likelihood is given by:

$$\log p(x \mid \mu, \sigma^2) = -\frac{1}{2}\left[\frac{(x-\mu)^2}{\sigma^2} + 2\log\sigma + \log(2\pi)\right]$$

The Gaussian likelihood shows how likely a point x is under the μ and σ^2. The first term is called the *quadratic term* and penalises deviations from the mean scaled by the variance. The second term called the *normalisation term* accounts for the distribution spread, and the final term is a constant ensuring proper normalisation.
A lower negative log probability, *Gaussian likelihood*, indicates the sample is more likely under the distribution. In contrast, a high negative log probability indicates a less likely sample under the distribution.

The actor has two terms in its loss function: first, an entropy regularisation term, where the log probability provides a metric for the likelihood of the actor, weighted by the temperature. The second term extracts the minimum of two Q-network (critic) outputs, similar to double Q-learning, to reduce overestimation bias; this term encourages the policy to select the actions for that state with the highest expected returns.

$$\mathcal{L}_{\zeta^S} = \frac{1}{N_b} \cdot \sum_{i=1}^{N_b} \left(\nu \cdot \log p(a_t^i|\mu, \sigma^2) - \min_{i=1,2} Q(s_t^i, a_t^i; \theta^{Q_i}) \right) \tag{3.27}$$

The TD error is computed to represent the mean-squared Bellman error for each critic, which is trained to minimise the square difference between its predicted Q-value and the shared TD target, through the loss function of Equation 3.29.

$$\begin{aligned}\hat{Q}_{\text{target}} =& R_t + \gamma \cdot (1-d) \cdot \left(\min_{i=1,2} \{Q'(s', a'; \theta^{Q'})\} - \nu \cdot \log p(a'|\mu', (\sigma')^2) \right) \\ \delta =& \frac{1}{2} \cdot \left((\hat{Q}_{\text{target}} - Q_1)^2 + (\hat{Q}_{\text{target}} - Q_2)^2 \right)\end{aligned} \tag{3.28}$$

$$\mathcal{L}_Q = \frac{1}{N_b} \cdot \sum_i^{N_b} \delta_i \tag{3.29}$$

SAC has been used to perform many tasks, including in a distributed learning form to perform control and navigation tasks [82]. Additionally, SAC with a distributed framework utilising a shared replay buffer with a multivariate reward representation containing sub-rewards for transfer learning, has learnt a policy for a humanoid walking through curriculum learning [83].

Maximum a posteriori Policy Optimisation

Trust Region Policy Optimisation (TRPO) is an iterative procedure for policy optimisation suitable for the large and non-linear policies of neural networks [26]. Here, a trust region controls the update between policies through KL divergence. PPO was presented as a first-order approximation of the second-order optimisation TRPO [27]. The introductory paper stated that PPO was simpler, more generic, and had better sample efficiency than TRPO, while consistently outperformed other on-policy online methods, such as Advantage Actor-Critic (A2C).

PPO is a stochastic actor-critic framework which prevents large updates causing instability through constraints on the deviation of the policy through its KL divergence. PPO was first employed without a value function, where the advantage estimate $\hat{A}_t$ is equal to the Monte Carlo return $\hat{A}_t = G_t(s_t, a_t) = \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k}(s_t, a_t)$. From here, a clipped surrogate objective is used to update the policy, as shown through the second equation in Equation 3.30. Note that the other objectives, including a KL divergence-based penalty, were used and showed decreased performance.

$$\begin{aligned}\text{No clipping or penalty:} \quad & \mathcal{L}(\theta) = r_t(\theta) \cdot \hat{A}_t(s_t, a_t) \\ \text{Clipped:} \quad & \mathcal{L}(\theta) = \min \left\{ r_t(\theta) \cdot \hat{A}_t(s_t, a_t),\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t(s_t, a_t) \right\} \\ \text{KL penalty:} \quad & \mathcal{L}(\theta) = r_t(\theta) \cdot \hat{A}_t(s_t, a_t) - \beta^{\text{PPO}} \cdot \mathcal{D}(\zeta^-(s_t; \theta^{\zeta^-}) \parallel \zeta(s_t; \theta^{\zeta})) \\ \text{Importance sampling ratio:} \quad & r_t(\theta) = \frac{\zeta(a_t|s_t; \theta^{\zeta})}{\zeta^-(a_t|s_t; \theta^{\zeta^-})}\end{aligned} \tag{3.30}$$

The actor-critic PPO was introduced to improve sample efficiency through value function estimation, stabilising the advantage function estimate as shown in Equation 3.31. Here, the critic is trained to minimise the value loss through of $MSE[G_t, V_t]$.

$$\begin{aligned}G_t(s_t, a_t) =& \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k}(s_t, a_t) \\ A_t(s_t, a_t) =& G_t(s_t, a_t) - V(s_t; \theta^V)\end{aligned} \tag{3.31}$$

MPO is as an off-policy version of PPO, with better sample efficiency [84]. The creators boast it is robust, insensitive to hyperparameter changes, and a scalable of an on-policy algorithm while having off-policy sample efficiency due to its utilisation of a replay buffer. MPO decouples PPO into two steps:

1. Optimisation of a non-parametric target policy.
    - The intermediate policy with a probabilistic representation of the value function.
    - The temperature ν controls the steepness of distribution, in essence controlling the exploration-exploitation trade-off.

- It doesn't rely on a neural network and is computed for the value function, acting like a "soft target" or a reference policy.

2. Parametric policy updating for target approximation.
   - The actual policy which the agent uses to interact with the environment, parametrised through a neural network.
   - Updates through minimisation of the KL divergence with its non-parametric policy.

The target policy is updated through Equation 3.32 called the ***E-Step*** (expectation step). Note that the actor is stochastic, like SAC.

$$\zeta_{\text{ref}}(s_t) \propto \exp\left(\frac{Q(s,a)}{\eta}\right) \tag{3.32}$$

The parametric policy is updated through minimisation of the KL divergence of the target and parametric network, as shown in Equation 3.30, incorporating KL divergence constraint to stay within the trust region.

The temperature parameter is updated through dual optimisation of Equation 3.33, where ϵ^{ν} controls the variance of the weights for regularisation. Sequential Least Squares Programming (SLSQP) has been used to minimise the dual function concerning ν [85], with a softmax function used to normalise the new weights, as shown in Equation 3.34.

$$g(\eta) = \eta \cdot \kappa^{\eta} + \lambda \cdot \mathbb{E}_{s\sim\mathcal{R}}\left[\log\left(\frac{1}{N_b}\sum_{i=1}^{N_b}\exp\left(\frac{Q(s,a_i;\theta^Q)}{\eta^*}\right)\right)\right] \tag{3.33}$$

$$w_i = \frac{\exp(\frac{Q(s,a_i)}{\eta^*})}{\sum_j \exp(\frac{Q(s,a_j)}{\eta^*})} \tag{3.34}$$

The actor updates through the log-likelihood of the target and current policies, as is a multi-variate Gaussian distribution. Secondly, it has two Lagrangian multipliers λ^{μ} and λ^{σ} to maintain the constraints on the KL divergence between the target and current policy's Gaussian distributions. Pre-defined thresholds M^{μ} and M^{σ} define the ***trust region***. This process is shown in Equation 3.35.

$$\begin{aligned}
\mathcal{L}^{\sigma} &= \lambda^{\sigma} \cdot \left(M^{\sigma} - \mathcal{D}_{\text{KL}}(\zeta^{-}(s_t;\theta^{\sigma}) \parallel \zeta(s_t;\theta^{\sigma}))\right), \\
\mathcal{L}^{\mu} &= \lambda^{\mu} \cdot \left(M^{\mu} - \mathcal{D}_{\text{KL}}(\zeta^{-}(s_t;\theta^{\mu}) \parallel \zeta(s_t;\theta^{\mu}))\right), \\
\mathcal{L}^{\zeta,\text{unconstrained}} &= -\frac{1}{N_b}\sum_{i=1}^{N_b}\Big(\log P(a_i \mid \mu^{-}(s_t;\theta^{\mu}),\sigma(s_t;\theta^{\sigma})) \\
&\qquad + \log P(a_i \mid \mu(s_t;\theta^{\mu}),\sigma^{-}(s_t;\theta^{\sigma^{-}}))\Big).
\end{aligned} \tag{3.35}$$

The authors tested it on control tasks *Walker-2D*, *Acrobat*, and *Hopper*; it was shown to outperform DDPG and PPO in an extensive ablation study. However, there is limited evidence to suggest whether it can outperform more advanced DDPG algorithms, such as D4PG, SAC, or TD3.

The original paper implementation ran the MPO algorithm in a distributed form, with a single learner but multiple learners (agents) collecting data independently, with a chief collecting the gradients and performing a parameter update through averaging gradients. In other words, *distributed synchronous SGD*. MPO can be used in a distributed framework (DMPO) under Google's Acme framework, allowing the user to configure it as desired [86].

3.3.5. Learning stability improvements

Normalisation techniques have been investigated as a form of regularisation [87]. First, it was showed how input normalisation benefits most RL environments. For an environment like rocket landing, where altitude can vary tens of thousands of meters and pitch angle to a few degrees, input normalisation is required. As the critic takes in both action and observation, both need to be normalised. To bound the states, they can be normalised between -1 and 1 through selected normalisation functions per state. In this section, several types of normalisation shall be presented.

$\tanh$ **normalisation:** smooths saturation of large values to squish outliers between $(-1, 1)$. The k factor can be selected to give the maximum expected value equal to an output of say 0.8, such that there is room for outliers beyond that.

$$x' = \tanh(k \cdot x) \tag{3.36}$$

Min-max normalisation: is a simple linear scaling, which is interpretable, but is sensitive to outliers.

$$x' = \frac{x - \min(x)}{\min(x) - \max(x)} \tag{3.37}$$

Shifted normalisation: zero centres the data but does not reduce it to within a range.

$$x' = x - \mu(x) \tag{3.38}$$

Exponential scaling: amplifies large value while suppresses small values to increase contrast. This can be useful for some scenarios where reward shaping wants large values to be emphasised.

$$x' = e^{k \cdot x} \tag{3.39}$$

Logarithmic scaling: is used to compress large values and expand small ones to reduce the dynamic range. It is helpful to stabilise large-magnitude inputs, providing an alternative to $\tanh$ and clipping. Also, it can be bounded between 0 and 1 through its denominator.

$$x' = \frac{\log(1 + x)}{\log(1 + \max(x))} \tag{3.40}$$

Andrychowicz et al. reviewed **gradient clipping** for RL and note it as secondary importance to value function normalisation [87]. However, they showed it can benefit learning stability. Here, the magnitude of the gradients of backpropagation is bounded to provide a safety net against exploding gradients, which are needed in high-variance environments, such as those that can occur with the use of N-step rewards.

Lemma. Gradient clipping (global norm scaling)

Gradient clipping regularises training through the prevention of gradient explosion to stabilise it by bounding the update step. The global L2 norm of all gradients across the network is computed:

$$\|g\| = \sqrt{\sum_i \sum_j g_{ij}^2}$$

If this norm exceeds a user-defined threshold τ, all gradients are scaled uniformly by a factor:

$$\text{scale} = \min\left(1, \frac{\tau}{\|g\| + \varepsilon}\right)$$

Where ε is a small constant added for numerical stability. Each parameter's gradient g_i is then rescaled:

$$g_i \leftarrow g_i \cdot \text{scale}$$

Additionally, regularisation is a technique used in machine learning to reduce a network's overfitting and encourage its ability to generalise. This can reduce the risk of the critic overfitting to large outlying TD errors, improving its generalisation capabilities. **L2 regularisation**, also known as weight decay, is a popular regularisation technique that discourages large weights [88].

Lemma. L2 regularisation (weight decay)

L2 regularisation improves the generalisation of neural networks by penalising large weight magnitudes, through a quadratic penalty term added to the loss function, modifying the loss to:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{original}} + \lambda \|\theta\|^2$$

Where λ is an L2 regularisation coefficient hyperparameter that controls the strength of the regularisation. This penalty encourages the optimiser to find solutions with small weights, which become less sensitive to noise and decrease the likelihood of overfitting.

3.4. Launch vehicle model

To learn a policy for powered descent, an *environment* is needed in the form of a launch vehicle model. This study aims to provide a benchmark for determining the feasibility of learning a policy to perform powered descent. Since no proprietary model is available, a new model is constructed based on physics and existing literature. First, *Starship* is reviewed as a representative reusable launch vehicle in Subsection 3.4.1, and an optimal staging procedure for reusable launch vehicles in Subsection 3.4.2. Aerodynamic dynamic coefficients for a representative body and grid fins are presented in Subsection 3.4.3 and Subsection 3.4.4. Finally, atmosphere and gravity models for Earth are provided in Subsection 3.4.5.

3.4.1. Starship case study

To ensure the launch vehicle's first stage can land, it must be sized to be feasible. Currently, there are only a handful of proven launch vehicles with the ability to take off and land vertically; these are from *SpaceX* and *Blue Origin*. To determine reasonable parameters for a reusable launch vehicle, *Starship* was reviewed. This is a two-stage launcher with the first stage called the *Super Heavy Booster* and a second stage called its namesake, *Starship*. The mass of each stage is shown in Table 3.2[10], with its structural coefficients calculated through Equation 3.41. This launcher is capable of carrying 100-150 tonnes to LEO orbit and 27 tonnes to GEO orbit [89].

$$\epsilon = \frac{m_s}{m_s + m_p} \tag{3.41}$$

[10] `https://www.spacex.com/vehicles/starship/`. Accessed: 23/06/2025.

Table 3.2: Mass and performance parameters of *Super Heavy* and *Starship*

	Super Heavy booster	Starship
Total mass [t]	3,675	1,600 (excluding payload)
Propellant mass [t]	3,400	1,500
Structural mass [t]	275	100
Structural coefficient [-]	0.0748	0.0625

In terms of propellant, both stages use a fuel oxidiser mixture of liquid oxygen (LOX) and liquid methane (LCH_4) called methalox. The 2nd stage carries 1170 kg of LOX and 330 kg of LCH_4, giving an oxidiser-fuel ratio of 3.545; this is assumed to be the same for the first stage, as the same engines, Raptor 3, are used.

The Raptor 3 engines are top-of-the-range full-flow cycle engines, with two varieties. One is optimised for sea-level performance, which is used on the first stage. The second is optimised for vacuum performance, with the second stage taking three of these for ascent and three sea-level optimised engines for descent. From the specific impulse- a measure of rocket efficiency that indicates the thrust produced per unit flow of propellant- the exhaust velocities are calculated using Equation 3.42. As the engines are optimised for sea-level and vacuum, the nozzle exit pressures are assumed to be close to the pressure in those regimes to minimise the pressure losses, Equation 3.43. The parameters of the Raptor 3 engine variants are displayed in Table 3.3[11].

$$v_{ex} = I_{sp} \cdot g_0 \tag{3.42}$$

$$\Delta T_p = (p_e - p_a) \cdot A_e \tag{3.43}$$

Table 3.3: Parameters of the Raptor 3 engines

	Sea-level Raptor 3	Vacuum Raptor 3
Specific impulse [s]	350	380
Exit diameter [m]	1.3	1.3
Exit area [m^2]	1.327	2.3
Thrust [MN]	2.745	2
Engine mass (integrated) [kg]	1525 (1720)	1525 (1720)
Exhaust velocities [m/s]	3433.5	3727.8
Exit pressure estimate [kPa]	101	0
Engine Height [m]	3.1	4.6

The thrust-to-weight ratio (TWR) is a crucial parameter that affects how much acceleration the rocket can produce, especially during takeoff. The engines produce thrust, resulting in the TWR determining the number of engines on the launch vehicle. The TWR is calculated using Equation 3.44 to determine the stage's TWR as shown in Table 3.4.

$$TWR = \frac{T_i^e \cdot n_i^e}{m_i \cdot g_0} \tag{3.44}$$

3.4.2. Optimal staging of a reusable rocket

The method of Jo and Ahn is presented to follow the sizing of a two-stage rocket with a reusable first stage and an expendable second stage [17]. Jo and Ahn's method optimally stages a launch vehicle with expendable and reusable stages, taking into account the velocity increment (Δv) losses in descent

[11] `https://www.spacex.com/vehicles/starship/`. Accessed 23/06/2025.

Table 3.4: Starship's stage's thrust-to-weight ratios.

	Super Heavy booster	**Starship**
Number of engines [-]	33	6
Stage Thrust-to-Weight ratio [-]	2.51	0.7645

and ascent, as well as the loss-free velocity increment for descent. Here, their method will be detailed to size a rocket with an expendable second stage and a reusable first stage.

First, the total descent velocity increment is found through Equation 3.45.

$$\Delta v_{d,1} = \Delta v^*_{d,1} + \Delta v_{d,1,\text{loss}} \tag{3.45}$$

The optimal staging procedure, similar to the method for expendable rockets, sets a parameter (here κ) in a modified formulation of the general Tsiolkovsky equation of Equation 3.47. When this equation is solved for κ, the payload ratio is minimised, in turn minimising the take-off mass.

The optimal staging procedure is similar to that of an expendable launch vehicle. It started by introducing a Lagrange multiplier κ to perform constrained optimisation of the Tsiolkovsky rocket equation. This yielded Equation 3.47, which, when solved, produced κ to maximise the payload ratio; the fraction of the launch vehicle's take-off mass that consists of payload. To do this, the velocity increment required was set from the orbital velocity at the defined semi-major axis, as shown in Equation 3.46.

To ensure the feasibility of the Lagrange multiplier κ, the following constraints were added to ensure physical and mathematical validity:

- **Exhaust velocity bound:** ensured κ avoids singularities in the logarithmic expression.

$$0 < \kappa < \min(v_{ex,1}, v_{ex,2})$$

- **Logarithm argument positivity:** guaranteed that the logarithm's values are strictly positive to prevent undefined values.

$$\frac{v_{ex,i} - \kappa}{v_{ex,i} \cdot \epsilon_i} > 0 \quad \Rightarrow \quad \kappa < v_{ex,i}$$

$$\begin{aligned} a =& y_t + r_E \\ \Delta v_{req} =& \sqrt{\frac{\mu}{a}} \end{aligned} \tag{3.46}$$

$$\Delta v_{req} = v_{ex,1} \cdot \ln(\frac{v_{ex,1} - \kappa}{v_{ex,1} \cdot \epsilon_1}) + v_{ex,2} \cdot \ln(\frac{v_{ex,2} - \kappa}{v_{ex,2} \cdot \epsilon_2}) - \Delta v_{d,1} \tag{3.47}$$

From the Lagrange multiplier, κ, the optimal payload ratios were computed. Starting with the expendable second stage, Equation 3.48 provides the optimal payload ratio, which is used to determine the optimal loss-free velocity increment for the second stage through Tsiolkovsky's rocket equation, as shown in Equation 3.49.

$$\lambda^*_2 = \frac{\kappa \cdot \epsilon_2}{(1 - \epsilon_2) \cdot v_{ex,2} - \kappa} \tag{3.48}$$

$$\Delta v^*_2 = v_{ex,2} \cdot \ln(\frac{1 + \lambda^*_2}{\epsilon_2 + \lambda^*_2}) \tag{3.49}$$

One of the benefits of following Jo and Ahn's work is how the launch vehicle was sized considering velocity losses, which are not insignificant. Velocity losses can result from thruster pressure losses, gravity, steering, and drag, significantly affecting the launch vehicle's achievable velocity increment.

To determine the size, the optimal loss-free velocity increment of Equation 3.49 was combined with the velocity losses of the second stage to yield the total velocity increment of the stage, as shown in Equation 3.50. The optimal payload ratio, considering losses, was then updated through a modified Tsiolkovsky equation to yield Equation 3.51.

$$\Delta v_2 = \Delta v_2^* + \Delta v_{loss,2} \tag{3.50}$$

$$\lambda_2^{l^*} = \frac{\epsilon_2 \cdot e^{\frac{\Delta v_2}{v_{ex,2}}} - 1}{1 - e^{\frac{\Delta v_2}{v_{ex,2}}}} \tag{3.51}$$

The masses of the second stage were then calculated using Equation 3.52, based on the defined structural coefficient of the second stage and the calculated optimal payload ratio, considering losses.

$$m_{s,2} = \frac{\epsilon_2}{\lambda_2^{l*}} \cdot m_{\text{pay}} \tag{3.52a}$$

$$m_{p,2} = \frac{1 - \epsilon_2}{\lambda_2^{l*}} \cdot m_{\text{pay}} \tag{3.52b}$$

$$m_{0,2} = m_{s,2} + m_{p,2} \tag{3.52c}$$

The expendable second stage has been sized, allowing for the reusable first stage to be sized by a similar but slightly different procedure. The structural coefficient of the first stage was the product of its descent and ascent structural coefficients. Rearranging Equation 30 of [17] to Equation 3.53, gave the loss-free descent structural coefficient, used in Equation 3.54 to find the loss-free ascent structural coefficient.

$$\epsilon_{d,1} = \left(e^{\frac{\Delta v_{d,1}^*}{v_{ex}}} \right)^{-1} \tag{3.53}$$

$$\epsilon_{a,1} = \frac{\epsilon_1}{\epsilon_{d,1}} \tag{3.54}$$

Following this, the optimal loss-free payload ratio was computed by Equation 3.56.

$$\epsilon_{a,1} = \frac{\epsilon_1}{\epsilon_{d,1}} \tag{3.55}$$

$$\lambda_1^* = \frac{\kappa \cdot \epsilon_{a,1}}{(1 - \epsilon_{a,1}) \cdot v_{ex,1} - \kappa} \tag{3.56}$$

Next, this optimal loss-free payload ratio was transformed into an optimal payload ratio that considers losses. This procedure started by calculating the total ascent velocity increment, as calculated in Equation 3.57. Then the descent velocity increment of Equation 3.45 was used to find the descent structural coefficient considering losses in Equation 3.58, which was used to calculate the ascent structural coefficient considering losses in Equation 3.59. Finally, the optimal payload ratio considering losses was provided in Equation 3.60.

$$\Delta v_{a,1} = \Delta v_{a,1}^* + \Delta v_{a,1,loss} \tag{3.57}$$

$$\epsilon_{d,1}^l = e^{-\frac{\Delta v_{d,1}}{v_{ex,1}}} \tag{3.58}$$

$$\epsilon_{a,1}^{l} = \frac{\epsilon_1}{\epsilon_{d,1}^{l}} \tag{3.59}$$

$$\lambda_1^{l^*} = \frac{\epsilon_{a,1}^{l} \cdot e^{\frac{\Delta v_{a,1}}{v_{ex,1}}} - 1}{1 - e^{\frac{\Delta v_{a,1}}{v_{ex,1}}}} \tag{3.60}$$

The payload ratios and structural coefficients known allow for the mass components of the first stage to be computed. The payload carried by the first stage $m_{L,1}^{l^*}$ was calculated in Equation 3.61, then used to find the stages structural and propellant mass during ascent in Equation 3.62 and Equation 3.63 respectively. Here, the ascent structural mass was both the vehicle's actual structural mass and the propellant reserved for landing. Knowing this, the final mass components of the stage were found using the subequations of Equation 3.64

$$m_{L,1}^{l^*} = \left(\frac{1}{\lambda_2^{l^*}} + 1\right) \cdot m_{\text{pay}} \tag{3.61}$$

$$m_{s,a,1}^{l^*} = \frac{\epsilon_{a,1}^{l^*}}{\lambda_1^{l^*}} \cdot m_{L,1}^{l^*} \tag{3.62}$$

$$m_{p,a,1}^{l^*} = \frac{(1 - \epsilon_{a,1}^{l})}{\lambda_1^{l^*}} \cdot m_{L,1}^{l^*} \tag{3.63}$$

$$m_{d,1}^{l^*} = m_{s,a,1}^{l^*} \tag{3.64a}$$
$$m_{s,d,1}^{l^*} = m_{d,1}^{l^*} \cdot \epsilon_{d,1}^{l} \tag{3.64b}$$
$$m_{p,d,1}^{l^*} = m_{d,1}^{l^*} - m_{s,d,1}^{l^*} \tag{3.64c}$$
$$m_{s,1}^{l^*} = m_{s,d,1}^{l^*} \tag{3.64d}$$
$$m_{p,1}^{l^*} = m_{p,d,1}^{l^*} + m_{p,a,1}^{l^*} \tag{3.64e}$$

3.4.3. Aerodynamics

The aerodynamics of the launch vehicle are essential to its behaviour, as they exert both forces and moments on the body. The chosen method is to use lift and drag coefficients, which vary with Mach number and angle of attack. As experimental data on *SpaceX*'s or *Blue Origin*'s launch vehicles is confidential, the legacy data of Wernher von Braun's V2 rocket from World War II can provide a benchmark for preliminary modelling. However, this is significantly different from modern launch vehicles. This section discusses the trends observed in the V2 aerodynamic coefficient curves, shown in Figure 3.12, and relates them to the governing flow physics across subsonic, transonic, and supersonic conditions.

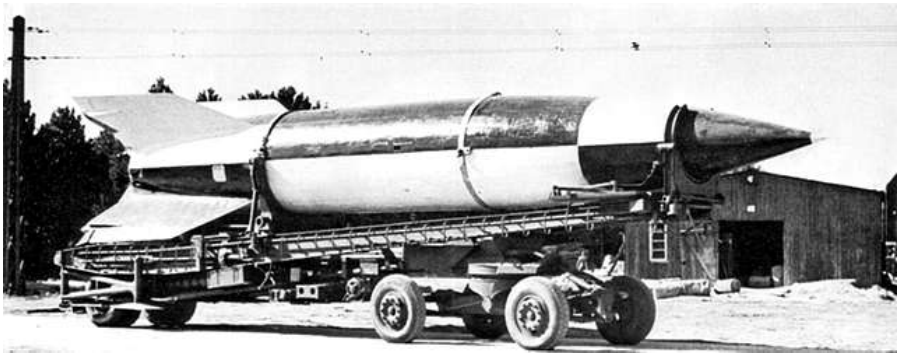


Figure 3.11: V2 rocket on Meillerwagen. Source: `https://timelessmoon.getarchive.net/media/v-2-rocket-on-meillerwagen-9effc8` (Accessed: 22 May 2025).

During the subsonic regime, under Mach 0.8, the drag is independent of the Mach number as compressibility effects are minimal. The lift coefficient initially increased with Mach number before decreasing, with the magnitude of the change observed being greater at higher angles of attack, as the circulation around the rocket increased.

For the transonic regime, between Mach 0.8 and 1.2, shock waves begin to form and propagate over the rocket's surface, causing *drag divergence*. Resulting in a rapid rise in drag approaching maximum drag and lift at 1.2. Strong shock and boundary layer interactions caused unsteady flow, leading to an increase in the coefficient.

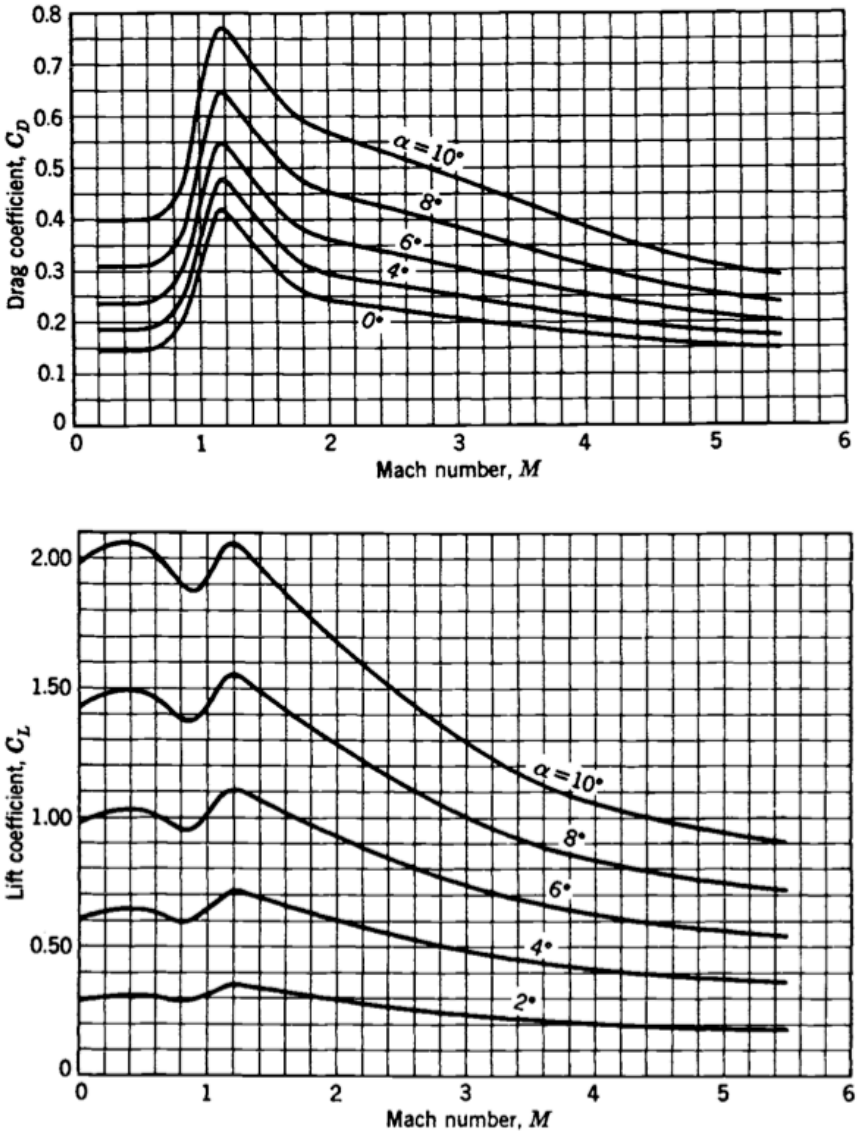


Figure 3.12: V2 rocket lift and drag coefficient curves [52].

The centre of pressure (CoP) is the point on the launch vehicle's body where the resultant aerodynamic force acts, in our case, the lift and drag forces. Reference [90] states that a launch vehicle has aerodynamic stability if its centre of pressure (CoP) is behind its centre of gravity (CoG). A launch vehicle may deviate, for instance, due to wind gusts, causing an increased angle of attack. A stable launch vehicle will correct for this by forcing it back to zero over time. As a result, the centre of pressure of the launch vehicle must be positioned so that it remains aerodynamically stable during both ascent and descent.

The lift and drag coefficients of Figure 3.12 can be converted to forces, lift and drag, through Equa-

tion 3.65 [91]. Where the area S is the cross-sectional area of the launch vehicle, so $S = \pi \cdot r_r^2$, and the dynamic pressure is $q = \frac{1}{2} \cdot \rho \cdot V^2$.

$$\begin{aligned} L =& \frac{1}{2} \cdot \rho \cdot V^2 \cdot C_L \cdot S \\ D =& \frac{1}{2} \cdot \rho \cdot V^2 \cdot C_D \cdot S \end{aligned} \tag{3.65}$$

For the ascent phase, the centre of pressure is behind the centre of gravity, which gives the freebody diagram Figure 3.13, with angle of attack α, the difference between the pitch and the flight path angle. Here, two frames were present:

- (x'', y''): The body frame originating from the centre of gravity, with y'' passing through the nose, and x'' perpendicular.
- (x', y'): The Local Vertical-Local Horizontal (LVLH) frame, where x' pointing East (right) , and y' the up.

From the freebody diagram, the equations of motion can be derived, Equation 3.66.

$$\begin{aligned} F_{x''} =& - L \cdot \cos(\alpha) - D \cdot \sin(\alpha) \\ F_{y''} =& L \cdot \sin(\alpha) - D \cdot \cos(\alpha) \\ M_z =& (d_{cg} - d_{cp}) \cdot (-L \cdot \cos(\alpha) - D \cdot \sin(\alpha)) \end{aligned} \tag{3.66}$$

The centre of pressure location can be checked to determine its stable position in relation to the centre of gravity. When an angle of attack is formed, a negative aerodynamic moment is desired to correct the angle of attack. Equation 3.67 derives the moment derivative with respect to angle of attack, before applying the small angle approximation. With a small α then $L \cdot \alpha << D$, and with the rest of the factors positive, the moment is restored.

$$\begin{aligned} \frac{dM_z}{d\alpha} =& (d_{cp} - d_{cg}) \cdot (L \cdot \sin(\alpha) - D \cdot \cos(\alpha) - C_{L_\alpha} q \cdot S \cdot \cos(\alpha) - C_{D_\alpha} \cdot q \cdot S \sin(\alpha)) \\ =& (d_{cp} - d_{cg}) \cdot (L \cdot \alpha - D - C_{L_\alpha} \cdot q \cdot S - C_{D_\alpha} \cdot q \cdot S \cdot \alpha) \end{aligned} \tag{3.67}$$

The descent reference from Figure 3.13 was used to derive the body frame forces from the aerodynamics in Equation 3.69. The effective angle of attack denotes the descent angle of attack, which was found using Equation 3.68.

$$\alpha_{eff} = \gamma - (\theta + \pi) \tag{3.68}$$

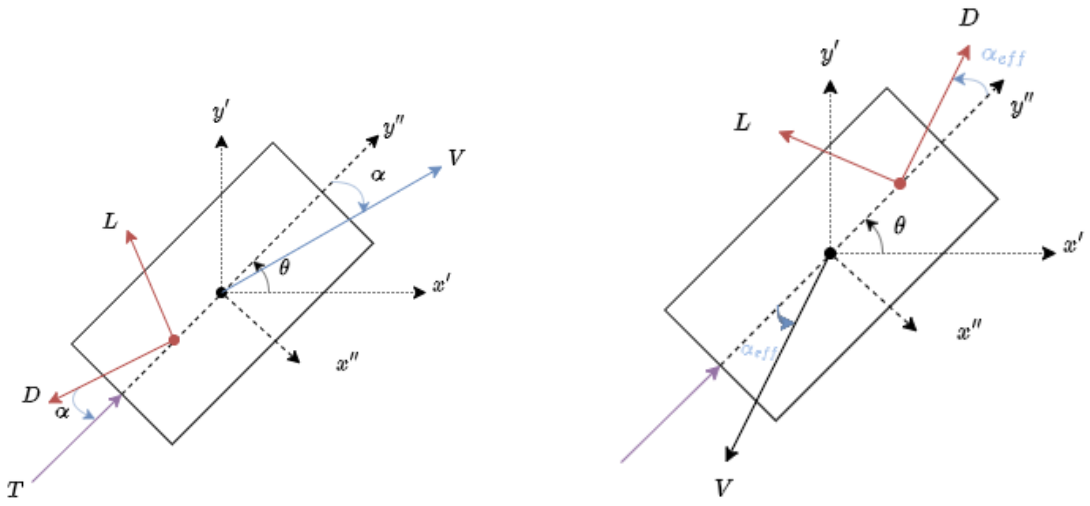
$$\begin{aligned} F_{x''} =& - D \cdot \sin(\alpha_{eff}) - L \cdot \cos(\alpha_{eff}) \\ F_{y''} =& D \cdot \cos(\alpha_{eff}) - L \cdot \sin(\alpha_{eff}) \\ M_z =& (d_{cg} - d_{cp}) \cdot (-D \cdot \sin(\alpha_{eff}) - L \cdot \cos(\alpha_{eff})) \end{aligned} \tag{3.69}$$

With the centre of pressure now moved to the top of the rocket, the aerodynamic stability can be checked in Equation 3.70. For aerodynamic stability, a positive restoring moment is required to minimise the angle of a. Equation 3.70 derives the moment derivative with respect to effective angle of attack before the small angle approximation is applied. A small angle of will make $L \cdot \alpha_{eff} << D$ so the applied moment is positive, as the rest of the terms are positive.

$$\begin{aligned} \frac{dM_z}{d\alpha_{eff}} =& (d_{cp} - d_{cg}) \cdot (C_{D_{\alpha_{eff}}} \cdot q \cdot S \cdot \sin(\alpha_{eff}) + C_{L_{\alpha_{eff}}} \cdot q \cdot S \cdot \cos(\alpha_{eff}) + D \cdot \cos(\alpha_{eff}) - L \cdot \sin(\alpha_{eff})) \\ =& (d_{cp} - d_{cp}) \cdot (C_{D_{\alpha_{eff}}} \cdot q \cdot S \cdot \alpha_{eff} + C_{L_{\alpha_{eff}}} \cdot q \cdot S + D - L \cdot \alpha_{eff}) \end{aligned} \tag{3.70}$$

Finally, the translation from the body to the LVLH frame is performed through Equation 3.71.

$$\begin{aligned} F_{x'} &= F_{y''} \cdot \cos(\theta) + F_{x''} \cdot \sin(\theta) \\ F_{y'} &= F_{y''} \cdot \sin(\theta) - F_{x''} \cdot \cos(\theta) \end{aligned} \tag{3.71}$$



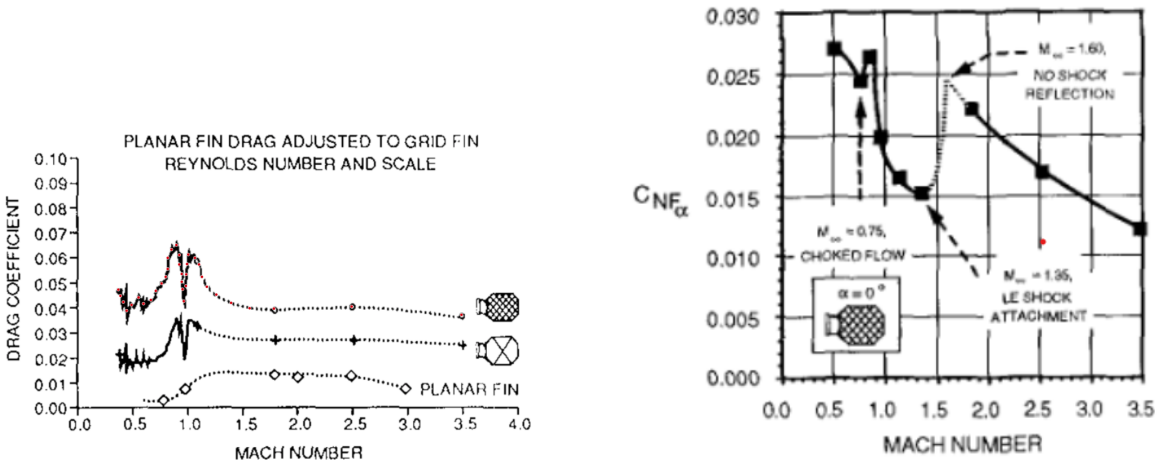
(a) Ascent. **(b)** Descent.

Figure 3.13: Aerodynamic Reference frames

3.4.4. Aerodynamic control surfaces

The aerodynamic control surfaces present on the first stage during descent were chosen to be grid fins, as shown in Subsection 3.1.1. Grid fins provide horizontal and angular control, while also giving some drag to slow down the rocket. Washington et al. performed experiments on grid fins and generated curves for their drag and normal force coefficients [92]. The curves represent a validated example of the change in aerodynamic coefficients with respect to angle of attack and Mach number, providing a representative representation of their dynamics suitable for a benchmark study.

At low angles of attack, the drag coefficient approximates the axial force coefficient for grid fins, as normal forces contribute minimally here. Figure 3.14a compares the drag coefficient for two grid fins and a planar fin; SpaceX's grids have a relatively dense lattice, resulting in the upper curve being the most representative. Furthermore, Figure 3.14b is a curve for the change in the normal force coefficient with local angle of attack. A derivation from these coefficients to forces and moments is provided in Section A.2.



(a) C_D of a fin with points selected for interpolation [92]

(b) C_{N_α} of a grid fin [92]

Figure 3.14: Side-by-side comparison of drag coefficient and normal-force gradient for a grid fin

3.4.5. Atmosphere and gravity models

Atmosphere dynamics influence the trajectory of the rocket, and as such, the control system design. For atmospheric models, the International Standard Atmosphere (ISA)[12] model is a commonly applied model in aerospace [93] to find the air density, atmospheric pressure and speed of sound as a function of altitude.

Gravitational acceleration varies as the launch vehicle moves from the Earth's surface; the inverse square law models this effect [94]. This is shown in Equation 3.72. A spherically symmetric and non-rotating Earth with uniform density, negligible oblateness and rotational centrifugal forces is assumed in this formulation. For precision, spherical harmonics, such as the J2 effect, can account for the Earth's equatorial bulge [94].

$$g = g_0 \cdot \left(\frac{r_e}{r_e + y} \right)^2 \tag{3.72}$$

3.5. Evolutionary algorithms for policy learning

EAs are population-based, gradient-free, black-box optimisation methods inspired by nature, evolution, and/or collective *swarming* intelligence [38]. These algorithms can solve problems across multiple domains, with the ability to scale to complex and non-linear optimisation problems, and manage a discontinuous and non-differentiable reward function. The managed stochastic nature of EA enables efficient exploration of a bounded parameter search space and avoids premature convergence with a sufficiently large population. This makes EA an attractive approach for performing *neuroevolution*, which involves tuning a neural network's parameters through EA, particularly in cases where a complex, non-linear reward (or objective) function is present. Through the non-gradient-based update procedure of ES, they can avoid local optima, a limitation typically constraining RL algorithms.

This section begins by reviewing two foundational papers that analyse and evaluate neuroevolution as an alternative to reinforcement learning in Subsection 3.5.1. Following this two selected EAs of genetic algorithms and particle swarm optimisation are presented with their extensions in Subsection 3.5.2 and Subsection 3.5.3. Finally, a trade-off between the algorithms is provided in Subsection 3.5.4.

3.5.1. Evolutionary algorithms as an alternative for reinforcement learning

Evolutionary algorithms perform stochastic population-based black box optimisation through having a group of candidate solutions (*population*) which are changed over each iteration (*generation*) through

[12] `https://www.engineeringtoolbox.com/international-standard-atmosphere-d_985.html`. Accessed: 23/06/2025.

operators (*perturbations*) inspired by nature, before their *fitness* is evaluated through an objective function, like RL's reward functions. Perturbations and information sharing encourage exploration of a bounded search space, particularly effective in non-convex and discontinuous optimisation problems [38],

OpenAI's landmark paper analysed Evolutionary Strategies (ES) as an alternative to reinforcement learning [20]. OpenAI's team used CMA-ES, as they said it was the most widely known. Compared to the Asynchronous Advantage Actor-Critic (A3C) RL algorithm, it performed better on 23 games while worse on 28, and showed better exploration than TRPO for MuJoCo's humanoid environment [40]. Salimans concluded that ES have benefits over RL, such as:

- Easily parallelise across CPUs, where candidates can be run across different nodes.
- Black box optimisation is not affected by the reward distribution, whereas RL can be slow in sparse reward environments.
- Works well with problems with long time horizons.
- RL can struggle from non-informative gradients of the policy's performance, which can come from a non-smooth reward or policy. As a result, value function variance can be large and thus difficult to learn. As ES don't use a value function approximator, this problem doesn't inhibit its abilities.

However, they noted that on "hard RL problems, it is often less effective than Q-learning. In summary, in context of rewards a problem with a long-time horizon, a non-smooth and/or sparse reward function can benefit from implementation of a EA as an alternative to RL.

A GA has been used to optimise a deep neural network's parameters and tested on RL benchmark environments of the Atari games [41]. It has been demonstrated that GAs can competitively compete with traditional RL methods for training neural networks. The authors noted in their discussion that, in specific scenarios, it may be worse to follow a gradient update than to perform local sampling of the parameter space to find a new solution. Still, they emphasised that this does not hold in all domains. Finally, a theory was presented for this: GAs can escape local minima, which simple gradient methods cannot.

3.5.2. Genetic Algorithm

A genetic algorithm initialises a *population* of candidate solutions (*genes*) with each parameter (*chromosome*) randomly initialised within its search space bounds [37]. In terms of a neural network, each gene is related to a specific weight or bias of one of its linear layers. The gene is initialised in the model, and the simulation is run. An *objective function* then evaluates the performance of the gene across the entire simulation. The resulting *fitness* aims to be maximised. Following this, *parent* gene combinations are *selected*, typically from a higher fitness, thus the most promising solutions. The parents share genetic information through *crossover*, exchanging portions of their genes to produce *offsprings* to promote diversity and exploration. Following the theory of natural evolution, a small part of the population has genes randomly *mutated* by a small amount of noise to unlock new solutions. Finally, if *elitism* is used, the best solutions are passed onto the next iteration (*generation*).

To promote diversity and discourage premature convergence to a local optimum, several solutions have been researched. One is the Island Genetic Algorithm (IGA), where the population is split into groups (*islands*) and runs independently. Periodic migration is performed between islands to provide new candidate solutions. This method has been motivated by its ability to preserve genetic diversity, as each island can follow a different search trajectory, thereby preventing premature convergence [95].

The Non-dominated Sorting Genetic Algorithm - II (NSGA-II) is an EA focused on multi-objective optimisation [96]. For example, making something as light as possible while making it as strong as possible. Instead of combining goals into a single fitness, NSGA-II looks for parameters that offer solutions to balance the trade-off between the goals. A good population of candidate solutions is maintained through comparison of how well they perform across all goals, and it prefers those that differ from other solutions to maintain a diverse population. Elitism is also used to balance performance and diversity.

NSGA-II sorts the population into layers based on Pareto dominance; a solution is said to dominate another when it is better in one objective and at least as good in the rest. The resulting best solutions

Algorithm 7 Genetic Algorithm (GA) with Elitism

Input: Population size N, crossover probability p_c, mutation probability p_m, maximum generations G, elitism count k
Initialize population P_0 with N individuals (randomly generated)
Evaluate fitness of each individual in P_0
for $generation = 0$ to $G - 1$ **do**
    Sort $P_{generation}$ by fitness (descending order)
    Elitism: $P_{generation+1} \leftarrow$ top k individuals from $P_{generation}$
    while $|P_{generation+1}| < N$ **do**
        Select parents p_1, p_2 from $P_{generation}$ using tournament selection
        Generate random $r \sim U(0, 1)$
        if $r < p_c$ **then**
            Crossover: $c_1, c_2 \leftarrow$ crossover(p_1, p_2)
        else
            Clone parents: $c_1 \leftarrow p_1$, $c_2 \leftarrow p_2$
        end if
        Mutation: $c_1 \leftarrow$ mutate(c_1, p_m)
        Mutation: $c_2 \leftarrow$ mutate(c_2, p_m)
        Add c_1 and c_2 to $P_{generation+1}$
    end while
    Evaluate fitness of new individuals in $P_{generation+1}$
end for

form the Pareto front, the trade-off boundary when no objective can be improved without worsening another. For diversity maintenance, crowding distance is used, measuring the difference between neighbouring solutions, keeping solutions in less-crowded areas to maintain diversity.

3.5.3. Particle swarm optimisation

Particle Swarm Optimisation (PSO) mimics the social behaviour of birds in a form of swarming intelligence [43]. A *swarm* of *particles* is randomly initialised within the search space bounds. Each particle is initialised within a separate but identical model, and its simulation is run, with an *objective function* evaluating the particle's performance to provide a *fitness*. Each particle stores its best fitness and *position* (the best solution), which is updated if the fitness improves. Also, a global best position and fitness are stored. These positions and fitness updates each particle's position $(\mathbf{x})$ and velocity $(\mathbf{v})$ through Equation 3.73, containing three weighted components:

- Inertia term: encouraging particles to search and explore by changing the position of the particle from its velocity. The inertia weight w_i is decayed over time to decrease exploration per generation.
- Cognitive term: pulls the particle back to its (local) best position $(\mathbf{x}^*)$ for exploitation, weighted by the cognitive coefficient c_1 and a random vector ξ_1.
- Social term: pulls the particle towards the global best position $(\mathbf{x}_s^*)$, weighted by the social coefficient c_2 and a random vector ξ_2.

$$\begin{aligned} \mathbf{v}_i \leftarrow & \underbrace{w\mathbf{v}_i}_{\text{inertia}} + \underbrace{c_1\,\boldsymbol{\xi}_1 \circ (\mathbf{x}_i^\star - \mathbf{x}_i)}_{\text{cognitive}} + \underbrace{c_2\,\boldsymbol{\xi}_2 \circ (\mathbf{x}_s^\star - \mathbf{x}_i)}_{\text{social}} \\ \mathbf{x}_i \leftarrow & \mathbf{x}_i + \mathbf{v}_i \end{aligned} \tag{3.73}$$

Algorithm 8 Particle Swarm Optimisation (PSO)

Input: Generations G, population size N, inertia bounds $w_{\min}, w_{\max}$, bounded domain projection operator $\Pi_{\mathcal{X}}$.
Initialise particles with random positions $\mathbf{x}_i$ and velocities $\mathbf{v}_i = \mathbf{0}$
Set $w \leftarrow w_{\max}$
Initialise global best fitness $f^\star \leftarrow -\infty$, position $\mathbf{x}^\star \leftarrow$ None
for each particle i **do**
    Set personal best $f_i^\star \leftarrow -\infty$, $\mathbf{x}_i^\star \leftarrow$ None
end for
for g in G **do**
    for i in N **do**
        Evaluate fitness $f_i \leftarrow f(\mathbf{x}_i)$
        if $f_i > f_i^\star$ **then**
            $\mathbf{x}_i^\star \leftarrow \mathbf{x}_i$
            $f_i^\star \leftarrow f_i$
        end if
        if $f_i > f^\star$ **then**
            $\mathbf{x}^\star \leftarrow \mathbf{x}_i$
            $f^\star \leftarrow f_i$
        end if
    end for
    for i in N **do**
        Sample $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2 \sim \mathcal{U}(0,1)^d$
        Update velocity: $\mathbf{v}_i \leftarrow w \cdot \mathbf{v}_i + c_1 \cdot \boldsymbol{\xi}_1 \circ (\mathbf{x}_i^\star - \mathbf{x}_i) + c_2 \cdot \boldsymbol{\xi}_2 \circ (\mathbf{x}^\star - \mathbf{x}_i)$
        Update position: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$
        Project $\mathbf{x}_i$ to domain: $\mathbf{x}_i \leftarrow \Pi_{\mathcal{X}}(\mathbf{x}_i)$
    end for
    Update inertia: $w \leftarrow w_{\max} - \frac{w_{\max} - w_{\min}}{G} \cdot g$
end for

Similar to the IGA implementation in Subsection 3.5.2, PSO can improve population diversity by splitting its swarm into multiple subswarms, a method employed in Cooperative PSO (CPSO) [97]. Here, the subswarms can cooperate by sharing solutions and migrating particles to enhance diversity and explore different solution paths.

3.5.4. Trade-off between Particle Swarm Optimisation and Genetic Algorithms

PSO and GA have been compared to optimise the parameters of artificial neural networks for controlling racecars in a simulation, to navigate around a track in the shortest time. The findings showed that PSO outperformed GA both in terms of speed and accuracy [98]. Moreover, PSO has shown faster convergence in designing a flexible AC transmission system-based controller for power system stability [99] and in the design of a reduced-scale two-loop pressurised water reactor core [100]. Furthermore, Stephen Chen, a Professor at York University, states that PSO has greater diversity and exploration, with momentum allowing faster convergence as it moves in the gradient's direction[13]. However, he also states that, as GA is inherently a discrete technique, and PSO is a continuous technique, it makes GA more suited for combinatorial problems. Chen's argumentation suggests that PSO, as a continuous technique, may be better suited to solve continuous problems, like fitting the parameters of a neural network; which is non-discrete.

[13] `https://www.researchgate.net/post/Can-you-please-list-main-advantages-of-PSO-over-GA`. Accessed: 06/22/2025.

4

Additional Results

The article presents the main results of the research, and this chapter presents supporting results to complement the main findings of the study. The results presented here also include verification tests and validation evaluations to highlight the flaws in the approach taken, while ensuring that the results align with expectations. First, in Section 4.1, the launch vehicle was staged following an optimal staging method for reusable launch vehicles of Subsection 3.4.2, with the resulting verification of the method's implementation conducted. Validation is evaluated through a comparison to industry launch vehicles. Following this, relations for the centre of gravity and inertia of the launch vehicle with respect to fuel consumption were derived in Section 4.2, and analysed, using *sanity checks* as verification. To conclude the sizing work package, the chosen values for the actuators are presented in Section 4.3. Afterwards, to generate an initial condition for policy learning of powered descent, a critical analysis of the performance of the preceding flight phase's controllers was conducted in Section 4.4, where the control laws, reference and dynamic response are presented and analysed. Section 4.5 presents the validation of the SAC reinforcement learning algorithm on the Lunar Lander benchmark problem. Finally, additional results to compliment the article regarding NE's performance on the powered descent problem, are presented in Section 4.6.

4.1. Launch vehicle staging: results, verification and validation

The staging procedure reviewed in Subsection 3.4.2 is used to optimally stage the launch vehicle considering velocity losses. This section covers the outcome of that process, with the first verification covered in Section 4.1, the results in Table 4.1, and finally, the validity of the results is discussed in Table 4.1 through a comparison with Starship.

Verification

The launch vehicle was staged using the method of Jo and Ahn [17], to verify the implementation of their process for a two-stage launch vehicle with a reusable first stage a reproduction of their Table 1 is performed. Errors were initially identified because the symbols used for variables in the table did not match the variables used in the equations. For example $m_{s,1}$ would be $m_{s,1}^{l^*}$ in paper, but more confusing is the payload ratio, as there is an optimal and an optimal loss free payload ratio, it was found that the table's λ is equal to λ^{l^*} and not λ^*.

The first step chosen was to calculate the procedure in reverse order and see if the κ (Lagrange multiplier) values align for the reusable and expendable stages, using an *Excel Spreadsheet*. The κ's were 0.0554% different, which was acceptable due to the three to four significant figures used in their value reporting. Table 4.1 presents the results of the spreadsheet's verification, with discrepancies arising from rounding errors in Jo and Ahn's reporting, which propagated throughout the calculation.

Table 4.1: Compared values from reverse order calculation of the optimal staging procedure for reusable launch vehicles considering velocity losses. The calculated values are given to the equivalent number of significant figures as Table 1 of [17]. The percentage difference is computed using non-rounded calculated values, and is to three decimal places.

Variable	Calculated	Jo and Ahn	Difference (%)
$\epsilon^{l}_{a,1}$	0.1421	0.1421	-0.017
$\lambda^{l^*}_{1}$	0.323	0.323	0.122
$\epsilon^{l}_{d,1}$	0.3606	0.3606	0
$m^{l^*}_{s,1}$	21600	21500	-0.263
$m^{l^*}_{p,1}$	399100	398600	-0.115

The next step of the verification procedure was to solve Equation 3.47 for κ correctly. κ was successfully reproduced with negligible difference. The Python implementation also produced the same values as the spreadsheet, with an insignificant difference.

Results

This section will document the results of the staging procedure. To be consistent with reusable launchers available in the industry, the staging shall be based on *Starship's* mission to Low Earth Orbit (LEO), as documented in Subsection 3.4.1. Therefore, 100 tonnes of payload shall be carried. However, for the second stage, reusability isn't considered, as it isn't applicable in this problem definition, which focuses on landing a launch vehicle's first stage, and Jo and Ahn don't provide their velocity losses for the second stage descent. The structural coefficients of the stages and exhaust velocities of the respective Raptor 3 engines are used to size the launch vehicle, with the velocity increments taken from Jo and Ahn's values as a first estimation. The results displayed in Table 4.3 include three subrockets, these are:

- Subrocket 0 is the full launch vehicle, used during the first stage burn for ascent.
- Subrocket 1 is the separated second stage containing the payload.
- Subrocket 2 is the separated first stage on its descent procedure.

Comparison to Starship

Table 4.2 shows a comparison between the values computed and Starships. First, it can be seen that the launch vehicle's total mass differs by only 2.75%. However, the second stage is nearly 50 % lighter; this could be because it is not reusable, so no extra propellant is held for landing, resulting in a lower structural mass and a lower overall mass. Also, the first stage is 25% larger, although it carries a lighter second stage. Telemetry data for Starship's flight[1] shows Super Heavy reaching 4500 km/hr, 1250 m/s on their 7th and 8th test flights, whereas Table 4.3 shows stage 1 with 2510 m/s loss-free velocity increment, this is increase in velocity increment causes the first stage to have a greater mass. Inaccurate velocity loss estimates will also cause the skewness between the values.

Table 4.2: Staging results comparison with SpaceX's Starship launch vehicle, to three significant figures.

Mass [t]	Table 4.3	Starship	Difference (%)
$m_{s,1}$	344	275	+25.2
$m_{p,1}$	4260	3400	+25.2
$m_{s,2}$	51.3	100	-48.7
$m_{p,2}$	770	1500	-48.7
$m_0 - m_{\text{pay}}$	5420	5280	+2.75

[1] `https://starship-spacex.fandom.com/wiki/Starship_Flight_Test_8#Telemetry` Accessed: 30-04-2025

Table 4.3: Key staging and mass breakdown values for the two-stage vehicle, to 4 significant figures.

Parameter	Symbol	Value	Unit
Lagrange staging multiplier	κ	2556	–
Optimal loss-free payload ratio stage 1	λ_1^*	0.6909	–
Optimal loss-free payload ratio stage 2	λ_2^*	0.1700	–
Optimal loss payload ratio stage 1	$\lambda_1^{l^*}$	0.2002	–
Optimal loss payload ratio stage 2	$\lambda_2^{l^*}$	0.1218	–
Structural coefficient ascent stage 1	$\epsilon_{1,a}$	0.1231	–
Structural coefficient descent stage 1	$\epsilon_{1,d}$	0.6079	–
Structural coefficient ascent stage 1 (loss)	$\epsilon_{1,a}^l$	0.1851	–
Structural coefficient descent stage 1 (loss)	$\epsilon_{1,d}^l$	0.4042	–
Optimal loss-free Δv stage 1	$\Delta v_{1,a}^*$	2510	m/s
Optimal loss-free Δv stage 2	Δv_2^*	6023	m/s
Initial launch vehicle mass	m_0	5521	t
Structural mass stage 1	$m_{s,1}$	344.3	t
Structural mass stage 2	$m_{s,2}$	51.32	t
Propellant mass stage 1	$m_{p,1}$	4257	t
Propellant mass stage 2	$m_{p,2}$	769.8	t
Structural mass stage 1 (ascent)	$m_{s,a,1}$	851.7	t
Structural mass stage 2 (ascent)	$m_{s,2}$	51.32	t
Propellant mass stage 1 (ascent)	$m_{p,a,1}$	3749	t
Propellant mass stage 2 (ascent)	$m_{p,2}$	769.8	t
Structural mass stage 1 (descent)	$m_{s,d,1}$	344.3	t
Propellant mass stage 1 (descent)	$m_{p,d,1}$	507.4	t
Actual structural mass stage 1	$m_{s,1}$	344.3	t
Actual structural mass stage 2	$m_{s,2}$	51.32	t
Stage 1 Mass	m_1	4601	t
Stage 2 Mass	m_2	821.1	t
Actual propellant mass stage 1	$m_{p,1}$	4257	t
Actual propellant mass stage 2	$m_{p,2}$	769.8	t
Initial mass of subrocket 0	–	5522	t
Initial mass of subrocket 1	–	921.1	t
Ascent burnout mass of subrocket 0	–	1773	t
Ascent burnout mass of subrocket 1	–	151.3	t
Mass at stage separation of subrocket 2	–	851.7	t
Mass at stage separation of subrocket 1	–	921.1	t

4.2. Inertia determination

As the launch vehicle burns propellant, the fill level of the tanks will decrease, resulting in a change of centre of gravity and inertia. To model this effect, functions for each of the sub-rocket are created to output the centre of gravity and inertia as a function of fuel consumption. A sub-rocket refers to the stages present on the launch vehicle at a certain point of flight.

First, the tanks are sized before the launch vehicle's sectioning model is presented as an approximation. Afterwards, the dry moment of inertia and centre of gravity for the stages are found. Finally, their wet values are showed with respect to fuel consumption.

Tank sizing

Each stage carries a tank for fuel and oxidiser, which combust in the engines to generate thrust. SpaceX stores the landing fraction of propellant in dedicated header tanks to ensure a stable propellant feed and minimise fuel sloshing effects during powered descent. However, to simplify the benchmark study, header tanks are excluded, with all propellant stored in the main tanks. Secondly, tanks typically have domes on the top and bottom, but this study approximates the tanks as cylindrical. Using the constants defined in Table 4.4, the procedure of Equation 4.1 is followed to generate the lengths of the tanks and masses of the propellant components for each stage, giving the final results in Table 4.5.

$$m_{LCH_4} = \frac{m_{prop}}{(1 + \frac{O}{F})}, \qquad m_{LOX} = m_{prop} - m_{LCH_4} \tag{4.1a}$$

$$h_f = \frac{m_{LCH_4}}{\rho_{LCH_4}} \cdot \frac{1}{\pi \cdot (r_r - t_{wall})^2}, \qquad h_{ox} = \frac{m_{LOX}}{\rho_{LOX}} \cdot \frac{1}{\pi \cdot (r_r - t_{wall})^2} \tag{4.1b}$$

Table 4.4: Tank sizing constants

Variable [unit]	Value
ρ_{LOX} $[kg/m^3]$	1200
ρ_{LCH_4} $[kg/m^3]$	450
$\frac{O}{F}$ $[-]$	3.545
t_{wall} $[m]$	0.01

Table 4.5: Tank sizing results to three significant figures.

Variable [unit]	Stage 1	Stage 2
m_{LOX} $[t]$	3320	600
m_{LCH_4} $[t]$	937	169
h_{LOX} $[m]$	25.8	4.67
h_{LCH_4} $[m]$	19.4	3.51

Rocket sectioning

The launch vehicle is sectioned in Table 4.2 to allow estimation of the centre of gravity and inertia. This low-fidelity method is valid for this study, although not exact, as it provides a representation of the inertia and centre of gravity changes a launch vehicle would experience. The first stage has the structural mass divided among a pre-allocated engine mass, dependent on the number of engines, using the Raptor 3 mass, lower and upper sections, along with structural mass surrounding the propellant: the tank.

This low-fidelity approximation enables the determination of a representative inertia and mass distribution without the added complexity of a CAD-based model or structural design of the launch vehicle. While it does not exactly capture the response, it provides sufficient resolution to show the dynamic changes in these variables throughout the launch vehicle's trajectory, as this is only a preliminary study into using policy optimisation methods for first stage descent control.

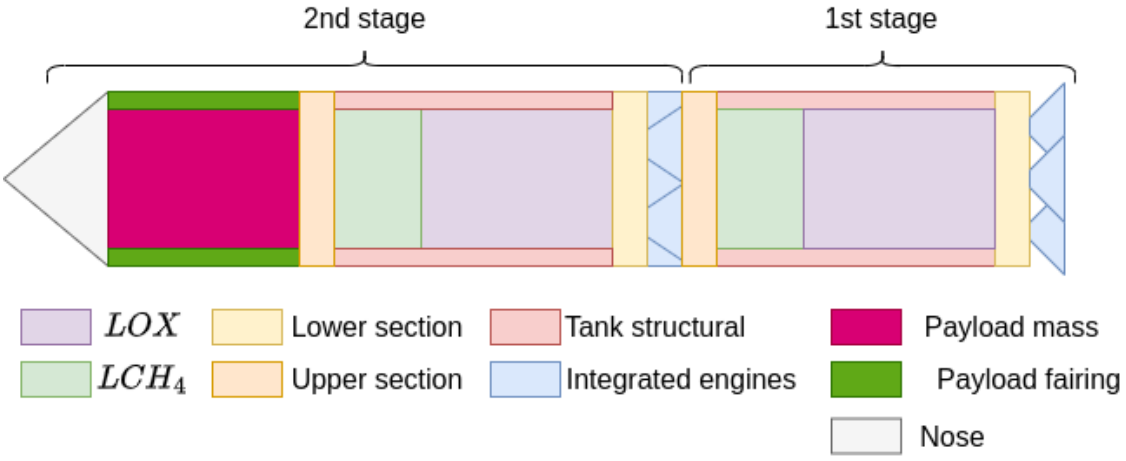


Figure 4.1: Sectioning of launch vehicle.

First stage dry inertia calculation

The masses of each of the first stage are calculated through Equation 4.2, using cylindrical tanks. The Λ_{ul} is the factor of mass remaining in the structural mass distribution between the upper and lower section heights; for this study, an equal distribution is assumed.

$$\begin{aligned} m_{s,tanks} &= \pi \cdot (h_f + h_{ox}) \cdot (r_r^2 - (r_r - t_{wall})^2) \cdot \rho_{304L} \\ m_{e,stage} &= n_e \cdot m_{e,integrated} \\ m_{upper} &= (m_s - m_{s,tanks} - m_{e,stage}) \cdot \frac{1}{1+\Lambda_{ul}} \\ m_{lower} &= (m_s - m_{s,tanks} - m_{e,stage}) \cdot \frac{\Lambda_{ul}}{1+\Lambda_{ul}} \end{aligned} \tag{4.2}$$

$$\begin{aligned} h_{upper} &= \frac{m_{upper}}{\rho_{upper}} \cdot \frac{1}{\pi \cdot r_r^2} \\ h_{lower} &= \frac{m_{lower}}{\rho_{lower}} \cdot \frac{1}{\pi \cdot r_r^2} \end{aligned} \tag{4.3}$$

$$y''_{dry} = \frac{-m_{e,stage} \cdot \frac{h_e}{2} + m_{lower} \cdot \frac{h_{lower}}{2} + m_{s,tanks} \cdot (h_{lower} + \frac{h_f + h_{ox}}{2}) + m_{upper} \cdot (h_{lower} + h_f + h_{ox} + \frac{h_{upper}}{2})}{m_s} \tag{4.4}$$

Now, the dry inertia is calculated using the cylinder's moment of inertia equation.

$$\begin{aligned} I_{e,stage} &= \frac{1}{12} \cdot m_{e,stage} \cdot h_e^2 - m_{e,stage} \cdot (y''_{dry} + \frac{h_e}{2})^2 \\ I_{lower} &= \frac{1}{12} \cdot m_{lower} \cdot h_{lower}^2 + m_{lower} \cdot (\frac{h_{lower}}{2} - y''_{dry})^2 \\ I_{upper} &= \frac{1}{12} \cdot m_{upper} \cdot h_{upper}^2 + m_{upper} \cdot (h_{lower} + h_{ox} + h_f + \frac{h_{upper}}{2} - y''_{dry})^2 \\ I_{s,tanks} &= \frac{1}{12} \cdot m_{s,tanks} \cdot (h_f + h_{ox})^2 + m_{s,tanks} \cdot (h_{lower} + \frac{h_f + h_{ox}}{2} - y''_{dry})^2 \\ I_{dry} &= I_{e,stage} + I_{lower} + I_{upper} + I_{s,tanks} \end{aligned} \tag{4.5}$$

Second stage dry moment of inertia

The payload structural mass is first computed using Equation 4.6, similar to the tank structural masses. Equation 4.2 is used to get the stage's total integrated engines mass and the tank structural mass.

Using the volume of the cone, the nose mass is calculated in Equation 4.7, allowing for the left-over mass to be applied to the second stage's lower and upper stages. Here, the nose is approximated as a cylindrical section, as in a launch vehicle; the nose is usually curved around the payload area, which provides a complexity reduction.

$$\begin{aligned} h_{pay} =& \frac{m_{pay}}{\rho_{pay}} \cdot \frac{1}{\pi \cdot (r_r - t_{fairing})^2} \\ m_{s,pay} =& \pi \cdot h_{pay} \cdot (r_r^2 - (r_r - t_{fairing})^2) \cdot \rho_{304L} \end{aligned} \tag{4.6}$$

$$\begin{aligned} m_{s,nose} =& t_{\text{fairing}} \cdot \pi \cdot r_r^2 \cdot \rho_{304L} \\ m_{s,upper} =& (m_s - m_{s,pay} - m_{s,tanks} - m_{e,stage} - m_{s,nose}) \cdot \frac{1}{1+\Lambda_{ul}} \\ m_{s,lower} =& (m_s - m_{s,pay} - m_{s,tanks} - m_{e,stage} - m_{s,nose}) \cdot \frac{\Lambda_{ul}}{1+\Lambda_{ul}} \end{aligned} \tag{4.7}$$

The dry center of gravity for the second stage including payload mass becomes:

$$\begin{aligned} y''_{cog} =& [-m_{e,stage} \cdot \frac{h_e}{2} + m_{lower} \cdot \frac{h_{lower}}{2} + m_{s,tank} \cdot (h_{lower} + \frac{h_f + h_{ox}}{2}) \\ &+ m_{upper} \cdot (h_{lower} + h_f + h_{ox} + \frac{h_{upper}}{2}) + (m_{pay} + m_{s,pay}) \cdot (h_{lower} + h_f + h_{ox} + h_{upper} + \frac{h_{pay}}{2}) \\ &+ m_{nose} \cdot (h_{lower} + h_f + h_{ox} + h_{upper} + h_{pay} + \frac{t_{\text{fairing}}}{2}] \cdot \frac{1}{m_s + m_{pay}} \end{aligned} \tag{4.8}$$

Now the dry moment of inertia can be found, taking $I_{e,stage}$, I_{lower}, I_{upper} and $I_{s,tanks}$ from Equation 4.5.

$$\begin{aligned} I_{pay} =& \frac{1}{12} \cdot (m_{pay} + m_{s,pay}) \cdot h_{pay}^2 + (m_{pay} + m_{s,pay}) \cdot (h_{lower} + h_f + h_{ox} + h_{upper} + \frac{h_{pay}}{2} - y''_{dry})^2 \\ I_{nose} =& \frac{3}{80} \cdot m_{s,nose} \cdot (5 \cdot r_r^2) + m_{s,nose} \cdot (h_{lower} + h_f + h_{ox} + h_{upper} + h_{pay} + \frac{r_r}{4} - y''_{dry})^2 \\ I_{dry} =& I_{e,stage} + I_{lower} + I_{upper} + I_{s,tanks} + I_{pay} + I_{nose} \end{aligned} \tag{4.9}$$

Wet moment of inertia and centre of gravity

Propellant in the form of fuel and oxidiser is consumed as the thrusters burn, resulting in a shift in the centre of mass and inertia of the launch vehicle. This cannot be neglected, as it affects the rotation of the vehicle due to changing moment arms. First, the wet centre of gravity is calculated with the oxidiser tank being placed underneath the fuel tank. f^l refers to the fill level of the tanks, and the tilde superscript $\tilde{x}$ is the adjusted position. The remaining oxidiser and fuel masses and heights calculated in Equation 4.10 are used to find the propellant centre of gravity with respect to the tank bottom in Equation 4.10. h_{lower} is the height from the bottom of the oxidiser tank to the base of the launch vehicle.

$$\tilde{h}_{ox} = h_{ox} \cdot f^l \qquad \tilde{h}_f = h_f \cdot f^l \tag{4.10a}$$

$$\tilde{m}_{ox} = m_{ox} \cdot f^l \qquad \tilde{m}_f = m_f \cdot f^l \tag{4.10b}$$

$$\tilde{y}_{prop} = \frac{\tilde{m}_{ox} \cdot (h_{lower} + \frac{\tilde{h_{ox}}}{2}) + \tilde{m}_f \cdot (h_{lower} + h_{ox} + \frac{\tilde{h}_f}{2})}{\tilde{m}_{ox} + \tilde{m}_f} \tag{4.11}$$

The inertia for the propellant is computed around the propellant's centre of gravity, modelling the tanks as cylinders and using the parallel axis theorem in Equation 4.12.

$$\tilde{I}_{ox} = \frac{1}{12} \cdot \tilde{m}_{ox} \cdot \tilde{h}_{ox}^2 + \tilde{m}_{ox} \cdot (h_{lower} + \frac{\tilde{h}_{ox}}{2} - \tilde{y}_{prop})^2 \tag{4.12a}$$

$$\tilde{I}_f = \frac{1}{12} \cdot \tilde{m}_f \cdot \tilde{h}_f^2 + \tilde{m}_f \cdot (h_{lower} + h_{ox} + \frac{\tilde{h}_f}{2} - \tilde{y}_{prop})^2 \tag{4.12b}$$

$$\tilde{I}_{prop} = \tilde{I}_{ox} + \tilde{I}_f \tag{4.12c}$$

The wet centre of gravity is calculated in Equation 4.13, which gives the final inertia values denoted by a hat subscript in Equation 4.14.

$$\tilde{y}_{wet} = \frac{m_s \cdot y_{dry} + \tilde{m}_{prop} \cdot \tilde{y}_{prop}}{m_{dry} + \tilde{m}_{prop}} \tag{4.13}$$

$$\hat{I}_{dry} = I_{dry} + m_s \cdot (y_{dry} - \tilde{y}_{wet})^2 \tag{4.14a}$$

$$\hat{I}_{prop} = I_{prop} + \tilde{m}_{prop} \cdot (\tilde{y}_{prop} - \tilde{y}_{wet})^2 \tag{4.14b}$$

$$\hat{I}_{stage} = \hat{I}_{dry} + \hat{I}_{prop} \tag{4.14c}$$

Changes with fuel consumption

Figure 4.2 shows the change in the centre of gravity with propellant consumption[2]. The graphs show the centre of gravity moves aft as propellant is consumed. This is logical, as propellant is held near the base of the stage, and burning propellant will remove propellant from the top of the tank. The oxidiser tank is located below (aft of) the fuel tank and has a greater mass due to its high oxidiser-to-fuel ratio. As a result, the mass loss occurs near the base.

Focusing on stage 1, the stage at which the burn is considered in this study, with a height of 50.4m. First, with no fuel, the centre of gravity is located just below its centre. As the engines are positioned below the base of the launch vehicle, they shift the dry centre of gravity to just below the midpoint. As fuel is consumed, the centre of gravity drops to just under 21.5m at about 57.5% fuel consumed, before rising to 22.6m at 100% consumption. Starting with a complete launch vehicle, the centre of gravity is below the dry value because the oxidiser is denser than the fuel and is primarily located below the centre of gravity. As the propellant burns, the levels of oxidiser and fuel decrease, shifting the centre of gravity aft. As the oxidiser tank's fill level moves past the centre of gravity, the centre of gravity moves forward.

[2]The complete launch vehicle takes in the first stage propellant consumption.

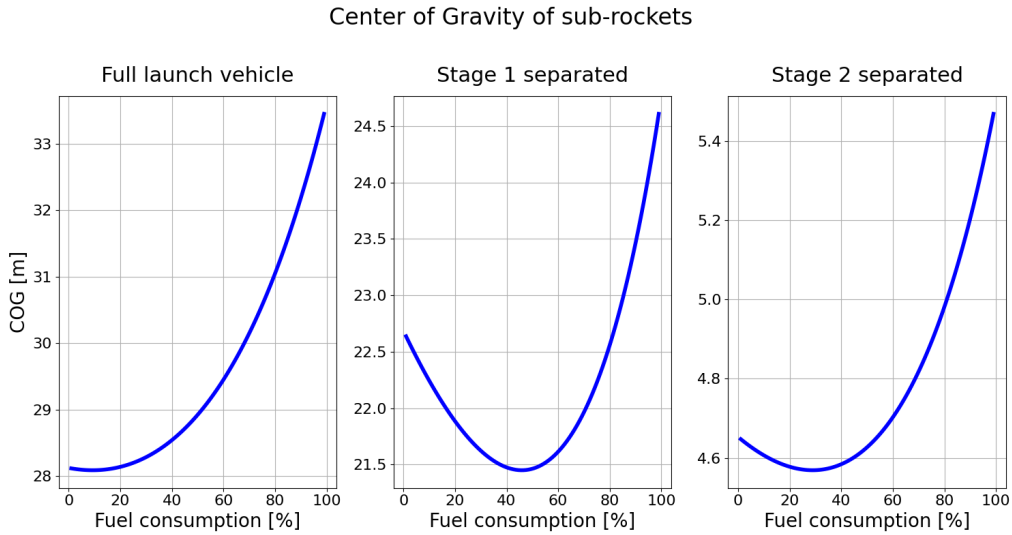


Figure 4.2: Centre of gravity change with fuel consumption.

Figure 4.3 shows the change of inertia with fuel consumption. Focusing, on the separated first stage, the inertia decreases as fuel is consumed.

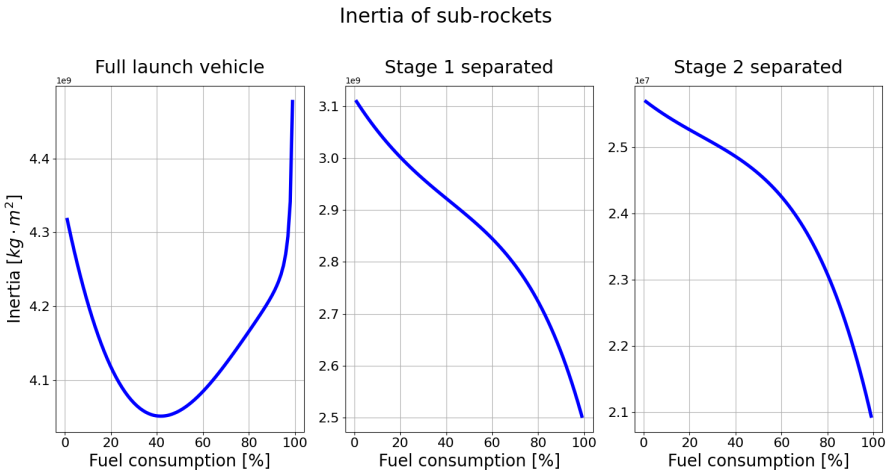


Figure 4.3: Inertia changes with fuel consumption.

4.3. Actuators sizing: results, verification and validation

Launch vehicle staging isn't the only part of launch vehicle sizing; the number of engines must also be determined to provide a radius for the launch vehicle. The actuators considered in this problem are grid fins, thrusters, and cold gas thrusters, with the thrusters based on the Raptor 3 parameters documented in Table 3.3.

Number of engines

After launch vehicle staging, the number of engines was determined by Equation 4.15 based on Starship's TWR for each respective stage. The number of engines was used to determine the launch vehicle's radius using Equation 4.16. Super Heavy Booster, which features a design with 20 fixed engines in the outer ring, followed by an inner ring of 10 gimballed engines and a central triangle of gimballed engines, totalling 20 fixed engines and 13 gimballed engines. Taking the three centrally gimballed engines as fixed, independent of the launch vehicle's configuration, the number of engines within the rings is altered. It is known that two extra outer ring engines are needed to accommodate one extra gimballed engine, resulting in a 2:1 ratio.

Note that the launch vehicle's radius was kept constant across the stages, similar to Starship, and sized through the first stage as more engines were required there. The results of this sizing are summarised in Table 4.6.

$$\begin{aligned} T_i^{req} =& m_i(0) \cdot g_0 \cdot TWR_i \\ n_i^e =& [\frac{T_i^{req}}{T_i^e}] \\ T_i^{max} =& n_i^e \cdot T_i^e \\ \dot{m}_i^{max} =& \frac{T_i^{max}}{v_{ex}^i} \end{aligned} \tag{4.15}$$

$$r_r = r_{\text{superheavy}} \cdot (1 + \frac{n^e_{\text{outer}} - n^e_{\text{outer,superheavy}}}{n^e_{\text{outer,superheavy}}}) \tag{4.16}$$

Table 4.6: Number of engines results, to three significant figures for non-integer values.

Variable	Stage 1	Stage 2
Number of engines [$-$]	42	4
Maximum thrust [MN]	115	8.00
Maximum mass flow [t/s]	33.6	2.15
Rocket radius [m]	5.85	5.85

Stage two had two fewer engines than Starship's 6; however, Table 4.2 shows it was 48.7% lighter. The number of engines was rounded up, so the calculated 3.09 becomes 4.0, for an extra margin. The first stage has 42 engines, which were arranged in an inner triangle of 3 gimballed engines, a surrounding perimeter of 13 engines, and an outermost perimeter of 26 engines. In total, this was nine more engines than the Super Heavy booster, but the sized stage was heavier, so it required more engines to match the TWR ratio.

Thruster modelling

To improve simulation fidelity, pressure losses were included through Equation 3.43: as the rocket moves through the atmosphere, atmospheric pressure changes, altering the true thrust the thrusters exert on the launch vehicle. Furthermore, a minimum of 40% throttle per thruster has been enacted to keep inline with available thrusters. However, during descent, the number of rocket engines active will decrease; this switching isn't modelled, but the central three thrusters are assumed always active, as seen with the Super Heavy booster.

The gimbal angle is defined as counter-clockwise to keep conventions uniform; this gives the body frame forces from the thrusters in Equation 4.17, with Equation 3.71 providing the inertial frame translation.

$$\begin{aligned} F_{x''} =& -T^g \cdot \sin(\theta^g) \\ F_{y''} =& T^{ng} + T^g \cdot \cos(\theta^g) \\ M_z =& -(d_{cg} - d_t) T^g \cdot \sin(\theta^g) \end{aligned} \tag{4.17}$$

Reaction control system

The RCS, in this case, is comprised of cold gas thrusters positioned on either side of the rocket at the top and bottom, for a scenario that only considers pitch control. They were selected to be 1m away from the top and base of the rocket, to give a large moment arm. Each group was modelled to apply a singular force of $5kN$ at their respective points. Minimal sizing was performed for the RCS procedure, as it had a minimal influence on the outcome of the research questions.

Grid fins

The grid fins were selected to be 2.5m by 2m to ensure they are representative and provide a high level of control authority. The grid fins' aerodynamic curves, as told in Subsection 3.4.4, were taken from Washington's work [92]. From these curves, points were selected and then interpolated, enabling the extraction of coefficients. The curves do not provide the full Mach number range. So, the Mach number range was linearly extrapolated to its maximum, and C_{n_α} was capped at zero. The lower bound extrapolation kept the coefficient value constant.

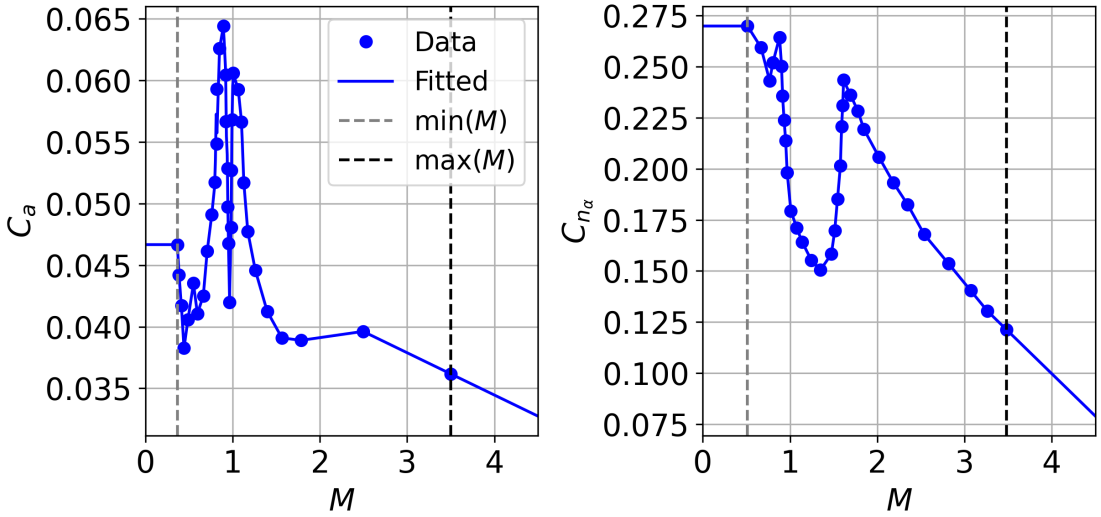


Figure 4.4: C_{n_α} and $C_a \approx C_D$ interpolated and extrapolated curves from Washington's work [92].

4.4. Controller performance: generation of an initial condition for powered descent and ensuring feasibility.

To start learning an optimal policy for powered descent, an initial state must be presented to the environment. This section generated the initial condition by performing unoptimized control manoeuvres, bringing it to the start of the powered descent phase. Starting with the first-stage ascent burn in Subsection 4.4.1, the launch vehicle was taken to a state where RTLS can be achieved by adhering to RETALT's flight envelope constraints. Next, the flip-over manoeuvre and boostback burn were simulated in Subsection 4.4.2 to take the booster to an arbitrary horizontal velocity, which can be optimised in later work to meet the landing pad[3]. The high-altitude ballistic arc control strategy and results are then presented in Subsection 4.4.3, providing the initial conditions for powered descent. Finally, Subsection 4.4.4 validates that a feasible solution space is present for powered descent policy learning.

4.4.1. Ascent control

Compared to conventional fully expendable launch vehicles, launch vehicles with a reusable first stage have a steeper ascent when pushing a payload to LEO and performing an RTLS landing. The lower horizontal speed and downrange distance result in less fuel required for the boostback burn, which in turn constrains the vertical and horizontal speed limits of the first-stage ascent burn for RTLS landing. Figure 4.5 illustrates these constraints for the RETALT1 concept launch vehicle, with a reusable first stage, highlighting that the terminal vertical and horizontal speeds of the first stage's ascent burn impact the orbit it can reach and the landing scenario. Taking the ascent flight path in the diagram, although for the AFLS scenario, it is followed until the dynamic pressure limitation region, where it stops at a vertical speed of 900 m/s. Table 4.7 records the associated flight path angles for this trajectory as a function of vertical speed.

[3] Section 4.6 manually tunes this target horizontal velocity to guide the launch vehicle to 333m away from the landing pad.

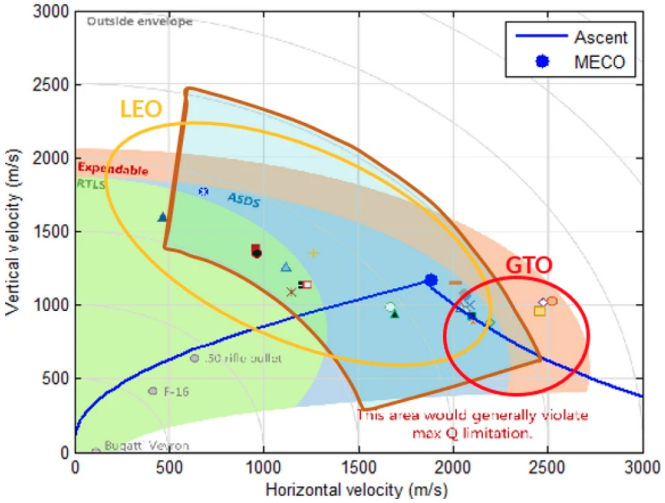


Figure 4.5: RETALT1's range covered from LEO and GTO orbits, giving RTLS, ASDS (Away From Launch Site), and expendable first stage outcomes [7].

Table 4.7: Vertical velocity versus flight-path angle for the ascent description of RETALT1's flight [7]

$v_y\ (m/s)$	Flight-path angle (°)
0	90.00
100	82.55
200	79.68
300	73.43
400	66.27
500	58.98
600	52.55
700	47.31
800	42.90
900	39.22

A Proportional-Derivative (PD) controller follows the pitch reference, which was the associated desired flight path angle concerning altitude of Table 4.7—providing a moment throttle output saturated between 0 and 1. An estimate for the maximum feasible moment over the flight was generated, and when multiplied by the moment throttle, it gives the desired pitch moment generated by the thrusters.

A heavy 30 kPa constraint on the dynamic pressure was applied during descent, in line with industry launch vehicles like the Falcon 9; this was lower due to the stage fairings often having lower limits. To manage this constraint, a Mach number reference was provided to a PD controller, which outputs a non-nominal throttle above a base 0.5 throttle. The Mach number reference was the Mach number corresponding to maximum dynamic pressure, as shown in Equation 4.18.

$$M_{\text{max}} = \sqrt{\frac{2 \cdot q_{\text{max}}}{\rho}} \cdot \frac{1}{a} \tag{4.18}$$

The total throttle is used to calculate the engine thrust through Equation 3.43, which, when multiplied by the number of gimballed and non-gimballed engines, gives the respective thrust for their blocks T^g and T^{ng}. From this, the desired gimbal angle is calculated using Equation 4.19.

$$M_z = T_g \cdot \sin(\theta_g) \cdot l_{th-cg} \longrightarrow \theta_g = \sin^{-1}(\frac{M_z}{T_g \cdot l_{th-cg}}) \tag{4.19}$$

Figure 4.6 shows the resulting flight profile for ascent. The launch vehicle rose to an altitude of 34 km and moved downrange to 20 km. The Mach number was well controlled by the throttle controller, with the launch vehicle satisfying the 30 kPa dynamic pressure constraint during ascent. This is controlled through a PD controller, which determines the maximum speed the launch vehicle can fly at to achieve the desired Mach number for an efficient yet safe ascent. Initially, the full throttle was used to raise the launch vehicle's speed as soon as possible, until it was throttled back to avoid overshooting the maximum Mach number. From here, the throttle gradually increased as the air density decreased due to the launch vehicle's climb, allowing for a faster speed. Eventually reaching full throttle again, as the Mach number reference is not achievable for the launch vehicle, the dynamic pressure drops below the 30 kPa limit.

The gimbal angle was the result of a moment reference generator and the throttle. As can be seen, the launch vehicle had a minimal pitch angle error. The gimbal angle begins with an initial rise to a positive angle after the launch vehicle clears the launch tower, marking the vertical rising phase. Following this,

it lowers to a negative angle to stabilise the pitch, before rising and oscillating for fine tracking. The gimbal angle drops to nearly negative six degrees during the throttling period, before rising above zero and slowly converging back to zero.

The pitch angle diverges from the angle of attack, as indicated by a final pitch angle error of nearly six degrees. Interestingly, it starts at zero before dropping into negative values as the launch vehicle heavily pitches over, then increases to just above zero as the vehicle throttles. Here, the aerodynamics has time to correct for the change in pitch angle, which is notably lower due to throttling. Finally, as the launch vehicle throttles again, the angle of attack decreases to negative six, plateauing near the end. This illustrates that the aerodynamic stability of the launch vehicle is insufficient to correct the angle of attack; therefore, additional control mechanisms, such as variable thrust over the engines or controllable aerodynamic surfaces, are required to ensure a decoupled controllable pitch moment and horizontal force.

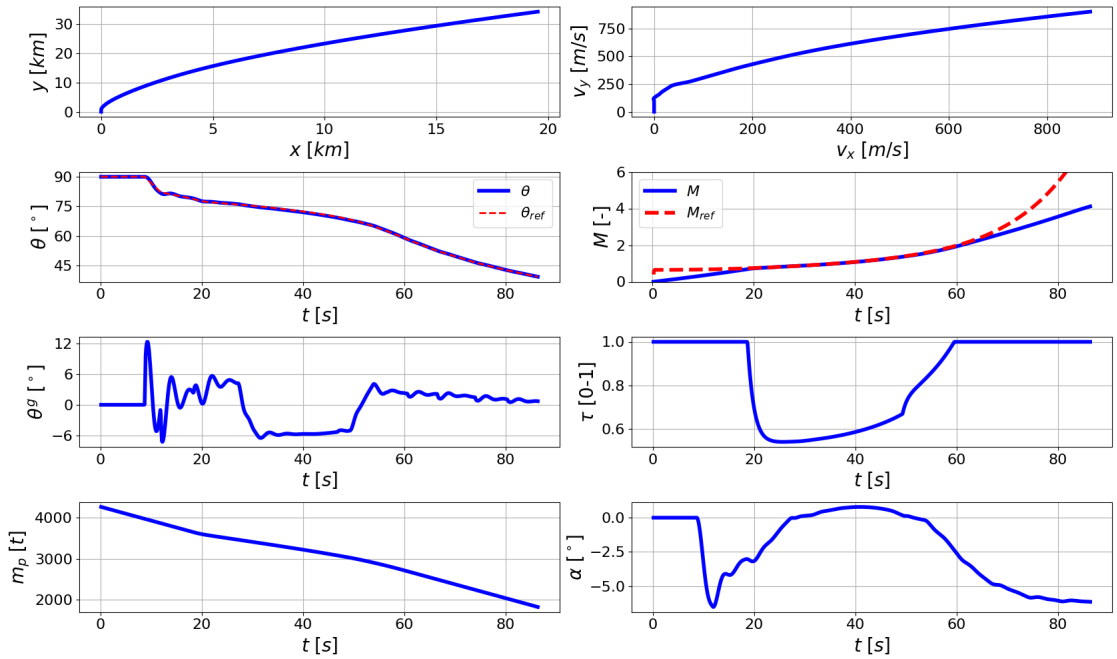


Figure 4.6: Simulation mission profile of the complete launch vehicle ascent to stage separation.

4.4.2. Flip over manoeuvre control

The flip-over manoeuvre rotated the launch vehicle's pitch to horizontal using TVC on a select number of gimballed engines. Here, a PD control took the pitch error as input to output a commanded gimbal engine, which was then filtered through a low-pass filter to simulate actuator delay. Figure 4.7 shows the results of the combined flip-over manoeuvre followed by the boostback burn to an unoptimised terminal horizontal velocity of $100m/s$. The thruster's gimbal quickly to a negative angle, causing a positive pitch rate by applying a force in positive x body axis to the rocket from its base to generate a positive moment. Then, the gimbal angle decreased and became positive to reduce the pitch rate, before decreasing and increasing again to finalise the pitch at its desired angle with negligible overshoot or steady-state offset.

The proportional term provided the main movement of the gimbal, and damping happened from the derivative term. The gimbal angle induced a change in the moment, with a negative gimbal providing a positive moment, thereby altering the angular acceleration of pitch. An integral term is not needed, as no steady-state error is observed, thereby simplifying the system's implementation to a PD controller with saturated outputs at the gimbal's maximum deflection of $10°$.

The gains will be negative as they align with the actuator conventions; a negative gimbal angle induces a positive pitch moment, which increases pitch acceleration and, consequently, pitch angle when the

initial pitch rate is zero. Recognising that negative gains are necessary, a constrained search space for gains was employed to identify the gains.

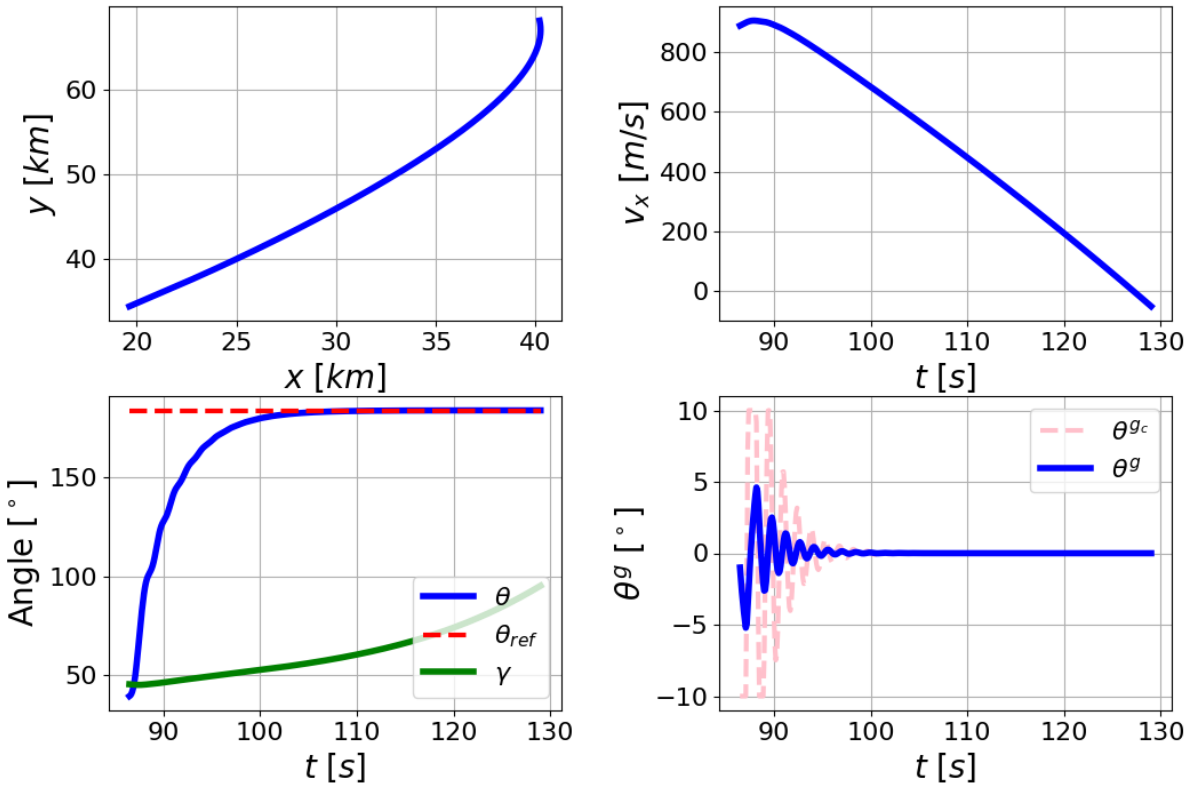


Figure 4.7: Mission profile of the flip-over manoeuvre followed by a boostback burn.

Particle Swarm Optimisation for Tuning the Controller of the Flip-Over Manoeuvre

A vanilla PSO algorithm was used to search for optimised gains for the PD controller performing the flip-over manoeuvre, under the cost function of Equation 4.20. The PSO fine-tuned the gains to achieve a faster response, resulting in an increase in performance. The comparison metrics and gains of Table 4.8 show a 4s reduced settling time, a 10.5% reduced cost, and a neglible final pitch error.

$$J = \sum_{i=0}^{N-1} \Bigg(\underbrace{\frac{|e_\theta(i)|}{10}}_{\text{response time}} + \underbrace{\frac{|\theta^g(i)|}{20}}_{\text{control effort}} + \underbrace{\frac{|\dot{\theta}(i)|}{5}}_{\text{heavy pitch rates}} + \underbrace{H(-e_\theta(i)\cdot\dot{\theta}(i))\cdot\frac{|\Delta\dot{\theta}(i)|}{2}}_{\text{pitching away from reference}} \Bigg) + \underbrace{10\cdot|e_\theta(N-1)|}_{\text{final error}} \tag{4.20}$$

Table 4.8: Result of manually and PSO tuned gains for a PD controller performing the flip-over manoeuvre, given to three significant figures.

Gains	K_p	K_d	Settling time [s]	Final pitch error [°]	J
Manual	-18.0	-5.00	17.5	1.53	914
PSO	-22.0	-4.80	13.5	1.47	818

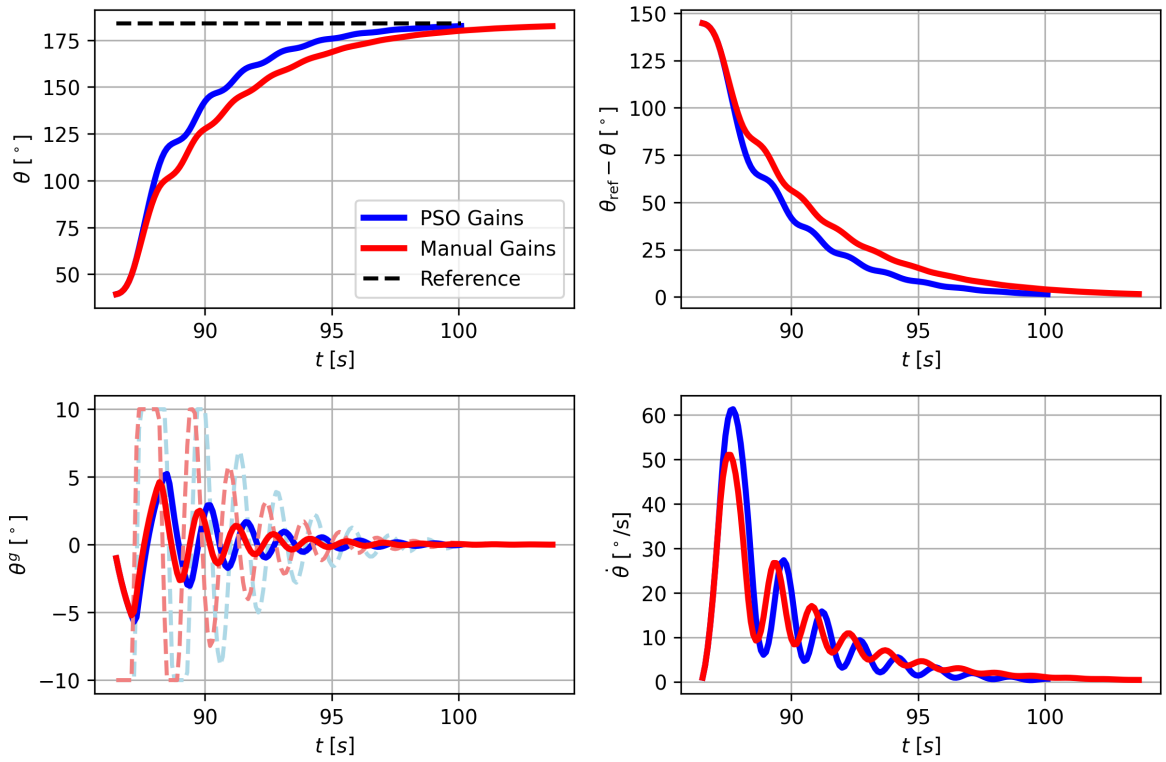


Figure 4.8: Comparison of a manually tuned (red) and particle swarm optimised (blue) controller gains.

4.4.3. High altitude ballistic arc control

The RCS was used to orient the rocket to achieve a low angle of attack after it reached its apogee. Here, a PD controller minimised the rate-limited angle of attack once the flight path angle reached $200°$. It began with a dominating proportional term, causing the RCS throttle to provide its maximum negative moment to achieve a negative pitch rate, until it approached the reference. At this point, the throttle turned positive to increase the pitch rate to positive, correcting for the overshoot before it was damped out.

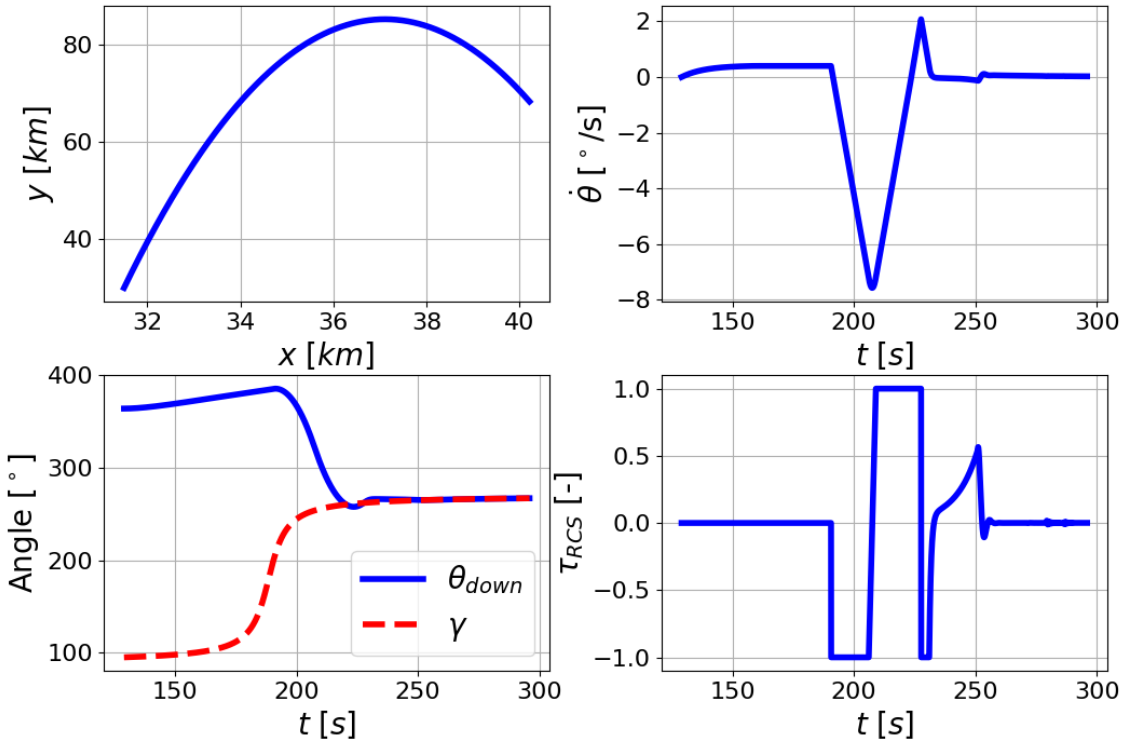


Figure 4.9: Simulation results from the high altitude ballistic arc.

4.4.4. Powered descent feasibility verification

This section covers the procedure used to ensure landing feasibility through throttle modulation. A feasible solution space was essential to guarantee that the policy optimisation methods were given a solvable problem, such that a feasible policy could be learnt. As explained in Chapter 2, the stage was subject to a $65kPa$ dynamic pressure constraint during descent, implicitly combining the re-entry and landing burns into a continuous powered descent burn, thereby reducing the problem complexity.

The purpose of this reference was not to generate an ideal solution but to validate that a throttle-only controller can, in principle, satisfy the dynamic pressure constraint across the entire landing phase. This ensured that the learning problem was not ill-posed and that the launch vehicle possessed sufficient control authority and propellant to regulate its descent under aerodynamic and terminal constraints. Once feasibility was established, this reference was discarded, and the policy was trained towards more optimal solutions.

A second-order polynomial function was fitted to generate this reference, constrained to pass through (start at) the initial descent condition and to a zero velocity at the ground. The polynomial was curved to have a point tangent with a $50kPa$ dynamic pressure limit, ensuring a trajectory within the flight envelope's constraints, giving a gradual reduction of speed. Figure 4.10 shows the polynomial fit in the red line, and the velocity corresponding to the dynamic pressure limit in blue.

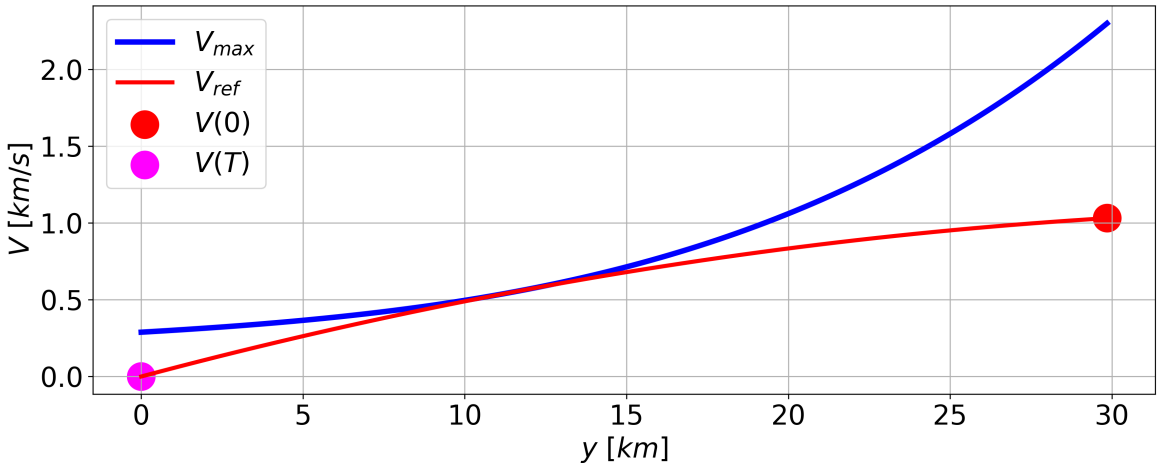


Figure 4.10: Initial powered descent velocity reference guess. V_{max} corresponds to the maximum speed the rocket can travel at to not breach the $50kPa$ dynamic pressure constraint.

A simple proportional controller satisfied speed reference tracking, eliminating the need for integral or derivative terms. In Chapter 2, this was replaced with an optimally learnt policy. As shown in Figure 4.11, the launch vehicle started throttling to maximum before down-ramping after the dynamic pressure peak was reached. During the phase of maximum throttle, the dynamic pressure limit of $50kPa$ was briefly exceeded to $60kPa$, even though the speed reference wasn't achievable at maximum throttle, meaning that zero-error tracking was unachievable. However, the dynamic pressure remained below the $65kPa$ threshold, showing the launch vehicle had sufficient thrust. The launch vehicle stopped $0.964m$ above the ground with -1.07 m/s vertical speed and a positive $1.669m/s$ horizontal speed. The low speeds cause the angle of attack to oscillate due to numerical integration errors, resulting in the horizontal speed overshooting $0m/s$ and then rising to a positive speed.

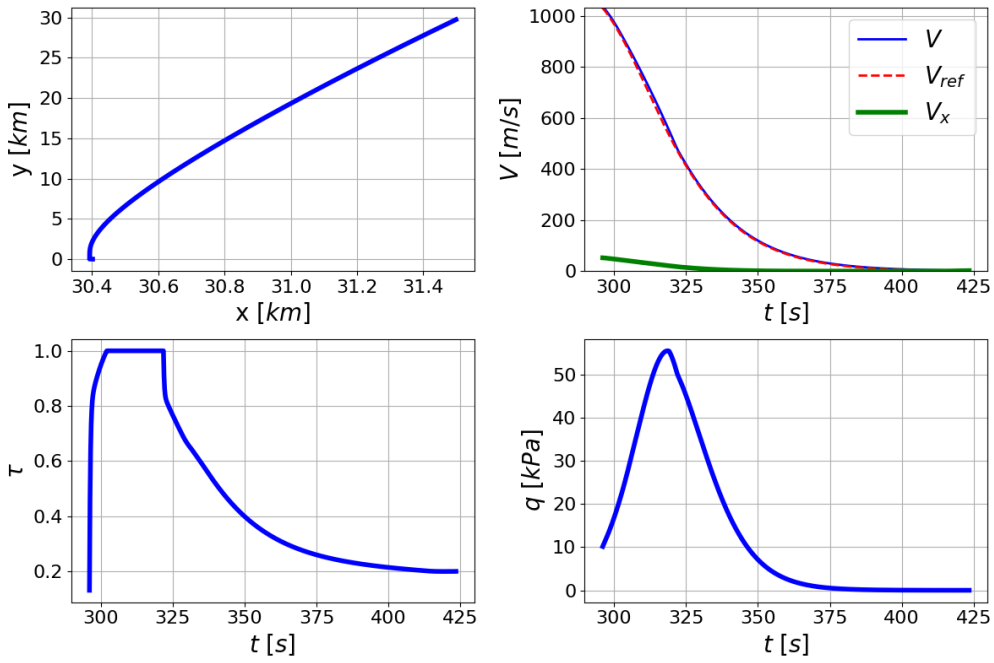


Figure 4.11: Landing burn dynamics using classical controllers to follow the velocity reference of Figure 4.10.

4.5. Reinforcement learning verification

This section aims to verify the SAC's implementation using a uniform replay buffer through a system test that is independent of the custom simulation environment. The *Lunar Lander v3* continuous environment, an OpenAI Gymnasium benchmark problem [101], has been used in literature to evaluate the success of RL algorithms due to its moderate complexity and terminal state rewards. Verification was performed by testing the performance of the SAC with unoptimised hyperparameters in the Lunar Lander environment to analyse convergence behaviour and validate that satisfactory exploration and exploitation are present.

The Lunar Lander environment provides a -100 and +100 reward for crashes and successful landings, respectively, along with lower magnitude rewards that guide success and optimality. Truncation (early stopping) of the episode occurs when the lander exits the frame, crashes or exceeds a maximum 1000-step limit, to improve learning efficiency.

The results are presented in Figure 4.12 using standard hyperparameters documented in Table 4.9. For a successful episode, the reward must be above 200 [4], which can be seen from 600 episodes in both the training and evaluation rewards, indicating that the SAC successfully solved the Lunar Lander problem.

The training rewards rose from -500 to 0.0, with some errors dropping to below -1000, indicating that the policy was still exploring. At 200 to 500 episodes, the problem was typically truncated due to the maximum number of steps constraint, thereby mitigating the negative 200 reward given for a crash landing. At 500 episodes, the reward rose to over 200, indicating a successful episode as a new experience was unlocked. This caused stochasticity as the agent adapted to the new transitions, before stabilising just after 600 episodes, resulting in continual successful landings.

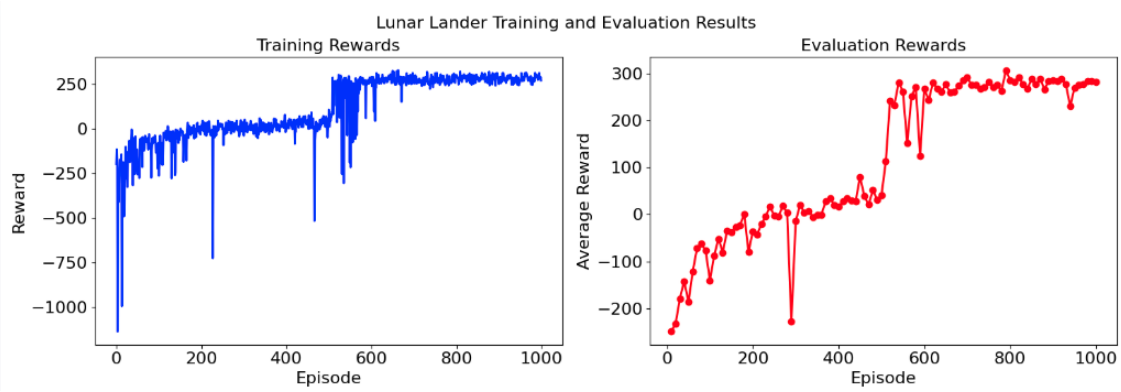


Figure 4.12: SAC agent's performance on the Lunar Lander v3 continuous benchmark problem from OpenAI Gymnasium [101].

Table 4.9: SAC hyperparameters used for the Lunar Lander verification test of Figure 4.12.

Actor layers	Critic layers	ν_0	γ	τ	N_b	η_Q	η_π	η_ν
2x256	2x256	0.2	0.99	0.005	256	0.001	0.001	0.001

Extra findings. The SAC's first implementation was with the JAX machine learning library with CUDA GPU acceleration, JAX uses Just-In-Time (JIT) compilation, to reduce computational time from compiling functions at runtime for accelerated computation. However, this resulted in a significant amount of code due to the creation of an inference class to facilitate logging of variables.

JAX's implementation was taking a long time to run episodes, at around 10-14 iterations per second for the initial 200 episodes. Therefore, blog posts were consulted to determine the number of episodes

[4] `https://gymnasium.farama.org/environments/box2d/lunar_lander/`. Accessed 03-06-2025

typically required to solve the Lunar Lander problem. These blogs commonly used PyTorch for the SAC, and the number of lines of code was a fraction of the current JAX implementation. Fewer lines of code provide easier maintainability. As a result, a quick PyTorch implementation with CUDA GPU acceleration was created, showing an average speed increase of over ten times. Although JAX had TensorBoard logging, which could slow it down slightly, the leading theory behind JAX's speed decline was incorrect JIT compilation at each step, causing significant compilation overhead. This motivated the SAC to be run on PyTorch. Finally, when extensive logging was enabled this time through `pandas`, the speed was halved, but it was still over five times faster than the original JAX-based implementation.

4.6. Additional neuroevolution findings regarding the powered descent problem.

The boostback burn used in the article and Section 3.1 landed the first stage 30 km Eastward of the landing pad. To provide an answer to RQ3 (Section 1.3), the boostback burn's terminal horizontal velocity was tuned to $-215m/s$, taking it to 333 m away from the landing pad using the initial guess velocity profile and throttle controller of Section 3.1. The powered descent phase then had an $818.6t$ starting mass, whereas previously it was $1630t$.

Figure 4.13 shows the best and average fitness of the subswarms and swarms as a whole. A clear trend of increasing average fitness was evident for subswarm one, whereas subswarm two had a delayed fitness increase. The main driver of the increase in fitness was the number of successful landings within the swarm. The best fitness values showed a first feasible solution at the 27^{th} generation for the first subswarm, with further fitness increases after reducing propellant consumption by 143 t, an 18.8% decrease in consumption. Subswarm two explored a different search trajectory, achieving the first feasible solution at the 44^{th} generation. However, it consumed a significant amount of fuel. This subswarm matched subswarm one's performance (within three kilograms) by the 150^{th} generation.

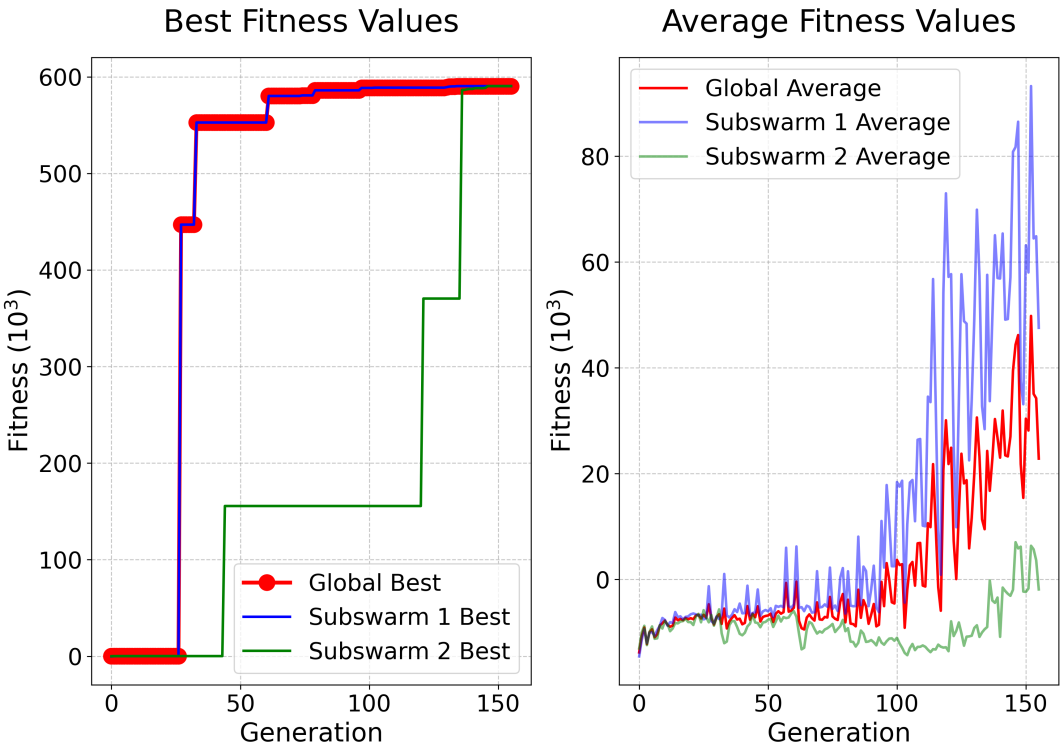


Figure 4.13: Best fitness and average of two subswarms performing PSO to fit a neural network over 150 generations to perform powered descent. The left plot displays the best fitness values, with the global best indicated by a red dot. The right plot shows the average fitness of the subswarms, with the average value indicated in red.

The best candidate solution's resulting trajectory is shown in Figure 4.13. The stage lands 157m East-

ward of the landing pad, the dynamic pressure peaks just below the 65kPa constraint. Following this, the throttle initially increased before decreasing to reduce velocity as the stage approached the landing pad. Ultimately, resulting in a successful landing.

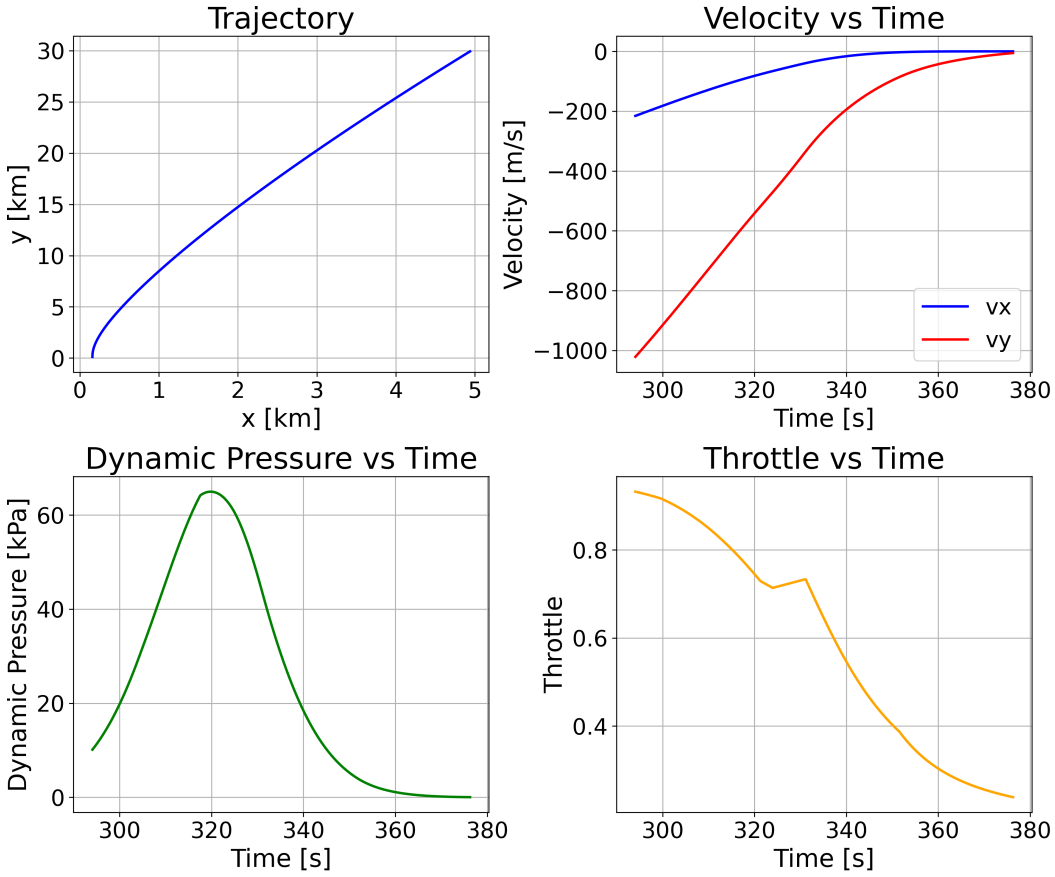


Figure 4.14: Trajectory of the final candidate solution from the PSSO performing NE of Figure 4.13.

5

Conclusion

This report aimed to provide an investigation into the performance of SAC and PSSO methods for learning a policy to perform the powered descent procedure for a launch vehicle's first stage. Secondly, it aimed to identify improvements in the iterative optimal staging and trajectory optimisation procedure for reusable launch vehicle sizing, particularly in terms of powered descent trajectory prediction and estimation of key design parameters, such as the moment of inertia. This section summarises and evaluates the findings of this report, relating directly to the research objectives, questions, and hypotheses presented in Chapter 1.

The motivation for linking policy optimisation methods for powered descent with reusable launch vehicle staging and sizing, and evaluating the performance of both SAC and PSSO for guidance and control was:

- Traditional methods for performing guidance and control of powered descent, like SCvx or MPC, require inner-loop controls to correct for errors in model linearisation. For example, changes in aerodynamic coefficients or moment of inertia are inherently nonlinear. As a result, Gaudet proposed utilising DRL to learn a policy directly that works with nonlinear dynamics, integrating guidance and control. However, Gaudet's work only considered a very small rocket in the final stages of powered descent[1] [14].
- The iterative staging-trajectory optimisation procedure presented by Jo and Ahn [17] would require new design parameters, such as the number of engines, to be determined for each staging. Additionally, with a different plant[2], the inner-loop controllers would need retuning due to changes in many plant variables, requiring either manual inspection or automatic tuning with custom cost functions.
- As explained in the Subsection A.1.2 and Chapter 2, the RL algorithms (SAC and TD3[3]) struggled to learn a solution; as a result, PSSO was considered as an alternative. This was supported by theoretical literature regarding benchmark comparisons of NE and DRL, where NE outperformed on problems with local minima (e.g., arising from a non-convex reward function), long time horizons, and high value function variance [20], [41], all present in the problem formulation. As a result, a comparison between gradient and non-gradient based methods' performances for this problem formulation was conducted.
- NE is the use of an EA to optimise a neural network's parameters, topology or learning algorithm coefficients. EAs are algorithms inspired by evolution [38], SI algorithms like PSSO, on the other hand, are inspired by the collective behaviour of animals, and can be said to perform NE in the

[1] Note that Gaudet's work was for a Mars mission, as such a 2km powered descent phase may be feasible. However, the Falcon 9 from SpaceX starts a re-entry burn at 50-60km, before a landing burn at 5km, to manage thermal and dynamic pressure constraints.

[2] For a control system, the plant refers to the process being controlled, i.e. the simulation model.

[3] TD3's results are not reported in this work but only included in the planning as reporting of only SAC's results directly answered the research questions more concisely.

place of EAs. The focus of NE here was only to optimise the parameters, keeping it coherent with implementations from the literature motivating its use over DRL [20], [41].

Key pieces of work were created, and insights were found. The optimal staging procedure for reusable launch vehicles considering velocity losses was reproduced from Jo and Ahn's work [17], complete with verification and validation through a comparison with *Starship*'s values in Section 4.1. From the masses found for each stage, the rocket's dimensions were estimated to create a variable moment of inertia and centre of gravity model, allowing for variable moment arms, as shown in Section 4.2. Furthermore, aerodynamic coefficient relationships for the launch vehicle body and manoeuvrable grid fins were incorporated to provide a representative change in aerodynamic forces with angle of attack and Mach number in Subsection 3.4.5, Subsection 3.4.4 and Section 4.3. Additionally, representative initial conditions for the powered descent phase were generated using mock references and classical controllers, as described in Section 4.4.

Chapter 2 conducted a comparison of using SAC and PSSO to learn a throttle profile for the powered descent problem, complete with an extensive hyperparameter study for the SAC to ensure a fair comparison. The PSSO quickly found a feasible solution and continued optimising for fuel consumption minimisation, resulting in a 90 t reduction in the article and a 143 t reduction for the best-performing swarm in the scenario of Section 4.6. For the PSSO, a gated reward function was easily created to guide the swarm first to avoid the dynamic pressure constraint, then to take it to the ground, and finally to achieve a successful landing, as well as minimise propellant consumption. By comparison, the SAC was unable to generate a solution for either sparse and non-sparse reward functions, due to non-informative gradients stemming from poor value estimation resulting from a non-convex reward function in the non-sparse reward case, and a lack of learning signals, which caused the sparse reward case to become trapped in a local optimum. NE performs non-gradient black-box optimisation with a population of solutions, mitigating these issues.

5.1. Key limitations of the work.

This section summarises the key limitations of the work and the reasoning behind them. Starting with the staging procedure validation of Section 4.1, a difference of 2.75% was found for the launch vehicle compared to *Starship*; however, the first stage was nearly 25% heavier, while the second stage was 50% smaller. The discrepancy was theorised to be due to the second stage in this work being modelled as expendable, whereas *Starship* is reusable. Secondly, the velocity losses and descent increment have not been updated, but instead taken from the work of Jo and Ahn.

Section 4.3 finds that the staged rocket of this work has nine additional engines (27%) more engines for the first stage, but it is heavier, and two fewer engines for the lighter non-reusable second stage. As a result, the radius is 30% larger. Furthermore, empirical data from proprietary sources are required to obtain accurate estimations of Λ_{ul} and ρ_{upper}, thereby providing representative lengths for the stages, and thus more realistic moment arms, centre of mass and inertia. However, for this study, this procedure is adequate as it illustrates the dynamics of changes in inertia, moment arms and centre of mass concerning fuel consumption.

Regarding the modelling, the main simplification is that only three DoF dynamics are used - lateral and longitudinal translation with pitch rotation. However, when reviewing key work on lossless convex optimisation, SCvx and DRL for the powered descent problem in Table 5.1, it can be seen that all the work considers a point mass model, and most of the work does not account for rotational motion. As a result, taking a three DoF model with rotational motion is comparable in complexity to similar methods. Furthermore, all the previously performed powered descents started from under 3 km above the surface as they were based on Mars. In contrast, this work began at an altitude of around 30 km to manage a significant dynamic pressure constraint.

Table 5.1: Degrees of freedom for previous work in generating guidance for the powered descent problem on Mars.

Reference	Year	Description
[3], [5], [11], [12]	2007, 2010, 2013, 2016	Considered three DoF, but did not consider the translational and rotational motion to be coupled, and modelled the rocket as a lumped mass.
[13]	2018	6 DoF point mass model.
[14]	2018	6 DoF point mass model, but without direct yaw control.

Creating a full-fidelity launch vehicle model was deemed infeasible within the project timeframe, as reasoned in Subsection 1.2.2, and is therefore presented in Section 5.5 as future work.

Regarding the flight phase controllers preceding the descent phase, first, the ascent phase controller obtained a reference profile from RETALT1 to enable a feasible RTLS launch. This profile was not optimised for the launch vehicle itself. As such, the propellant allocated for first-stage ascent, as assigned by the optimal staging procedure, was not fully consumed. Furthermore, the angle of attack peaks at 6.147 at the end of the flight, as uniform gimbal angles and throttling are assumed across the engines, causing a coupling between the pitch moment and the perpendicular force. To fully correct for this, non-uniform throttle, additional aerodynamic control surfaces are required.

Although the control for the flip-over and boostback burn performs well, the optimal burn time or terminal horizontal velocity of the boostback burn is required to ensure a fuel-optimal landing at the landing pad. This was circumvented by tuning the terminal velocity manually to guide the launch back to within 333 metres from the landing pad, in Section 4.6. Finally, Chapter 2 reduces the final landing burn phases to two phases[4] and neglects aerodynamic heating for simplicity.

Finally, there were computational limitations as presented in Subsection 1.2.2. These were partially mitigated through implementing a multithreaded solution for the PSSO and GPU acceleration for the SAC.

5.2. Hypotheses.

This section evaluates the hypotheses presented in Subsection 1.2.3.

HP.1. Reinforcement learning algorithms can successfully control the powered descent phase of a launch vehicle's first stage, achieving a stable landing.

From the experiments conducted in Chapter 2, the SAC controller did not find a solution to powered descent; however, this does not mean that other RL approaches cannot achieve a successful powered descent. Accordingly, HP.1 is not rejected.

HP.2. The selected neuroevolution algorithm will require fewer training episodes to learn an effective control policy for powered descent, compared to the selected reinforcement learning method, thereby demonstrating greater learning efficiency.

Chapter 2 shows that the PSSO finds a feasible solution, whereas the SAC fails to do so within the same number of episodes; thus, HP.2. is not rejected.

HP.3. Policy optimisation methods can be effectively applied to determine velocity increments for optimal staging in reusable launch vehicles, yielding consistent performance when hyperparameters are held constant across optimisation iterations.

Section 4.6 provides evidence not to reject HP.3. as a successful solution if found with a different initial powered descent mass.

HP.4. The developed methodology will estimate the inertia of a launch vehicle stage when provided with a specified stage mass at staging.

Section 4.2 developed this methodology, passing this hypothesis. However, it would benefit from validation with proprietary data.

[4] Falcon 9 and RETALT1 [7] follow a mission profile with a re-entry burn, followed by a ballistic arc, and finally a landing burn. Whereas, the SuperHeavy Booster has a single landing burn.

5.3. Research questions.

This section evaluates the research questions presented in Section 1.3.

RQ.1 How can policy optimisation methods be applied to find effective high-level trajectory planning and mid-level guidance control strategies for the powered descent of a reusable launch vehicle's first stage?

- RQ.1.1 What policy optimisation algorithms are most suitable for performing powered descent?
- RQ.1.2 To what extent can the selected algorithms achieve a successful landing under representative powered descent scenarios on Earth?
- RQ.1.3 How do the learning efficiency and stability of the different algorithms compare in this context?

Traditionally, powered descent utilises convex optimisation (covered in Section 3.1) to provide guidance, with inner-loop tracking error controls. Policy optimisation methods offer the ability to join the guidance and control into a single system [14]. Section 3.2, focusing on model-free algorithms, deduced that an online algorithm was preferred as the policy will initially not be sufficient for landing and such online learning (within a simulation) is needed to unlock new solutions. Moreover, an off-policy method was chosen because it allows the reuse of previous transitions through a shared replay buffer, which improves sample efficiency. To work on a continuous domain, five actor-critic formulations were reviewed: DDPG [72], D4PG [80], SAC [28], TD3 [81], and MPO [84]. SAC was deemed the most promising as DDPG, D4PG, and TD3 use state-independent Gaussian action space noise, whereas SAC learns a stochastic policy to manage exploration. The state-dependent noise modulation of SAC enabled targeted exploration during stages within the powered descent phase, in contrast to a fixed Gaussian noise, which can destabilise nominal trajectories. Furthermore, two buffer types were chosen for trial: the standard uniform replay buffer and the more advanced PER buffer [70] to improve sample efficiency.

Two mainstream algorithms compatible with NE were reviewed in Section 3.5: GA and PSO (with their extensions). PSO was chosen over GA as it has shown faster convergence on three examples [98]–[100], and is better suited to fitting the parameters of a neural network, as GA is an inherently discrete algorithm. The PSO was extended to a PSSO to increase diversity and explore different search trajectories. The application of PSSO performing NE was demonstrated in Chapter 2 to show a powered descent with a single policy, without the need for linearization model simplification and internal tracking error controls. The policy optimiser, through training on non-linear dynamics, learns a policy that accounts for non-linear dynamics, such as variable aerodynamic coefficients and pressure losses on thrusters. As a result, the solution first provided a feasible powered descent trajectory and then reduced propellant consumption during the flight phase.

Furthermore, the gradient-based SAC RL algorithm was validated in the LunarLander environment, as shown in Section 4.5, achieving successful completion within 600 episodes. However, when added to a more representative model in Chapter 2, it struggled with a non-smooth reward function, causing difficulty in value function estimation. However, this problem could be solved by providing a velocity reference or through inverse reinforcement learning. However, they both would then require non-policy-optimisation methods to generate this reference. Alternatively, the non-gradient-based PSSO algorithm, which performs NE (see Subsection 3.5.1), provided a feasible powered descent solution, independent of a velocity reference or an inverse reinforcement-learned reward function. As a result, the non-gradient-based policy optimisation algorithm is suitable for performing powered descent.

To summarise the answers:

- **RQ.1** Policy optimisation can provide an integrated guidance and control solution.
- **RQ 1.1** SAC was deemed the best model-free RL algorithm, and PSO was preferred over GA for performing NE.
- **RQ 1.2** PSSO successfully lands the rocket, whereas SAC fails to land it within the period of PSSO.

- **RQ 1.3** PSSO is inherently stable as it does not rely on gradient-based updates, which cause instability in SAC. Also, PSSO efficiently finds a solution within four generations.

RQ.2: What model fidelity is required to accurately represent powered descent flight for a reusable launch vehicle's first stage as a feasibility study for policy optimisation's effectiveness?

- RQ.2.1 How can a representative initial condition for powered descent be generated based on a realistic mission profile?
- RQ.2.2 What are the key mathematical equations necessary for an adequate simulation model of powered descent?
- RQ.2.3 Which model extensions (e.g., aerodynamics, fuel sloshing) could most improve fidelity for future research?

Section 3.4 presents a rocket model incorporating non-linear features. For example, aerodynamic coefficients are taken from the V2 rocket's curves to provide a representation of their changes with angle of attack and Mach number. Moreover, a variable centre of gravity and inertia model is presented in Section 4.2 to model their changes with fuel propellant consumption, which has often been overlooked in work on rocket landing control. The actuators comprised of gimballed and fixed thrusters with pressure losses, grid fins with variable aerodynamic coefficients (see Subsection 3.4.4) and a reaction control system comprised of cold gas thrusters. A clear mathematical description of the model is shown in Chapter 2. The final recommendation of Section 5.5 includes extending the model to 6-DoF and incorporating fuel sloshing dynamics.

The initial condition generation for the powered descent flight phase starts with a verified implementation of an optimal staging procedure for reusable launch vehicles (see Subsection 3.4.2 and Section 4.1). Following this, the launch vehicle has parameters such as the number of engines and dimensions, providing a first iteration of staging for a launch vehicle. This launch vehicle follows a mock ascent reference from RETALT1's mission profile, utilising PD controls and a Mach number reference constrained by dynamic pressure limitations. Then, a PD controller performs a flip-over manoeuvre, which proceeds to the boostback burn and a high altitude ballistic arc past its apogee. The initial condition is checked for feasibility in Subsection 4.4.4 by a mock velocity reference constrained by the maximum dynamic pressure being tracked by a throttle controller, giving a successful soft landing under the terminal conditions of Chapter 2.

To summarise the answers:

- **RQ.2** Subsection 1.1.1 introduces the need for non-linear features such as variable inertia, centre of mass and aerodynamic coefficients; limitations of previous work ([3], [5], [11], [12], [14]), and Section 3.4 presents how non-linear features can be incorporated.
- **RQ.2.1** Section 4.4 presents the generation of a representative initial condition for a scenario of a two-stage launch vehicle travelling to LEO with a reusable first stage under a RTLS landing scenario.
- **RQ.2.2** The mathematical model is presented in Chapter 2.
- **RQ 2.3** Recommended model extensions are the inclusion of fuel sloshing and extending to a six DoF model.

RQ.3: Do changes in launch vehicle parameters necessitate re-tuning of hyperparameters for learning a policy to perform powered descent?

- RQ.3.1 How do the hyperparameters change the learning/optimisation stability and efficiency regarding the chosen RL algorithm?
- RQ.3.2 How can policy optimisation be incorporated into the optimal staging procedure to handle variations in launch vehicle parameters?
- RQ.3.3 How can a low-fidelity methodology estimate key physical parameters (such as inertia and actuator sizes) for a launch vehicle following a staging procedure?
- RQ.3.4 How similar are the launch vehicle parameter estimations compared to the *Starship* launch vehicle?

Section 4.6 documents a PSSO performing NE with the same hyperparameters as the solution in Chapter 2, Section 4.6 starts with a different initial condition and still provides a feasible solution to powered descent, before minimising propellant consumption. This demonstrates that the hyperparameters are independent of the launch vehicle's starting mass at powered descent, and therefore, no retuning of hyperparameters is necessary. A key drawback of the optimal staging procedure for reusable launch vehicles [17] was identified in Subsection 1.2.1, namely that for each new rocket configuration, classical controllers require their gains to be retuned, which often necessitates human feedback. Conversely, NE provides a solution which can be run autonomously. As a result, it is proposed to replace the classical controllers with policy optimisation methods. Finally, Chapter 2 conducted an extensive ablation study on the SAC's hyperparameters. Covering: actor, critic and temperature learning rates, and the Polyak averaging coefficients. Furthermore, for the flip-over manoeuvre, an example of tuning the PD controller with PSO is provided in Subsection 4.4.2.

Section 4.2 presents an inertia estimation model, and Section 4.3 determines the number of engines required based of *Starship*. The staging procedure is compared to *Starship* in Section 4.1 and Section 5.1 discusses the discrepancy - being due to first iteration velocity increments and a non-expendable second stage, leading to a 25% heavier first stage and 50% smaller second stage, but an overall difference of 2.75%.

To summarise the answers:

- **RQ 3** PSSO, the successful method, has been shown to converge with 50% lower starting mass in Section 4.6.
- **RQ 3.1** An extensive ablation study of the SAC was conducted in Chapter 2.
- **RQ 3.2** PSSO NE can provide a method for generating velocity increments for powered descent independent of the launch vehicle's parameters. Additionally, Chapter 2 recommends extending this to use NSGA-II for learning the boostback burn's terminal velocity and the policy for powered descent.
- **RQ 3.3** Section 4.2 and Section 4.3 developed a model to estimate the variable inertia and centre of mass, and Section 4.3 estimated the number of engines.
- **RQ 3.4** Section 4.1 compared the staging values to *Starship*.

5.4. Reflection

This study contributes to the primary research objective:

To investigate the performance of gradient and non-gradient based policy optimisation methods for first-stage landing during the powered descent phase.

The study finds that the non-gradient-based PSSO outperforms the gradient-based SAC in learning a throttle mapping for powered descent. This could motivate future research into utilising NE for reusable launch vehicle guidance and control, over RL. Moreover, it provides a *cautionary tale* on the applicability of RL for continuous control problems, showing how, for specific problems, it can struggle to converge. For instance, SAC works well without hyperparameter tuning in the simple Lunar Lander environment, but was unable to converge on a higher-fidelity model with dynamic pressure. As a result, when choosing to use neural network-based control, backed by previous literature [41], the author recommends:

If the reward function is non-convex with a considerable number of local optimums, NE is preferred over RL.

Regarding the secondary research objective:

To develop and evaluate improvements in the iterative process of optimal staging and trajectory optimisation for reusable launch vehicles, with a focus on the powered descent trajectory prediction and the estimation of key design parameters, to support more efficient and reliable launch vehicle design.

This study presents an estimation of key design parameters and provides powered descent trajectory prediction without the need for manual retuning of controllers. This can be extended to the full descent procedure as recommended in Chapter 2.

5.5. Recommendations for future research

The article of Chapter 2 proposed three avenues for future work regarding the descent guidance problem. Firstly, utilising NSGA-II to optimise all the descent flight phases in combination [96]. Secondly, to incorporate a known horizontal wind into the physics and the provision of additional actuators. Finally, to try model-based RL [51], reward shaping [102], or inverse reinforcement learning [62], [63]. Below are extra recommendations and future work directions.

- Benchmark problems play a crucial role in evaluating the performance of reinforcement learning and neuroevolution algorithms. For example, OpenAI's gymnasium has Atari games and MuJoCo robotics environments, covering continuous and discrete control tasks. Additionally, the Farama Foundation maintains additional open-sourced environments configured the same as gymnasium; these include a UAV flight control environment and car racing simulators. Adding a benchmark first-stage landing environment with the ability to select which flight phase here would enable future work regarding the performance of these algorithms for this problem.
- NE can learn a policy for powered descent under a horizontal wind profile, as proposed in the article. The resulting trajectory can be used to create a velocity tracking reward for an RL algorithm operating in an environment with wind gusts, modelling uncertainty, and additional disturbances. This approach is similar to that of Gaudet, who used a velocity target to guide the agent [14]. To ensure robustness Robust Adversarial Reinforcement Learning (RARL) can be utilised [103].
- Generative Adversarial Imitation Learning (GAIL) of SCvx solutions by a neural network [104] to allow for faster solutions[5]. This can then be learned as part of the curriculum with an inner-loop neural-based controller.
- First, extend the model to 6-DoF, incorporate fuel sloshing and a variable centre of pressure. Following this, consider the incorporation of a launch vehicle's bending modes and aerodynamic coefficients for modern launch vehicles over the V2 curves.

[5] This approach would require additional computational resources than used to conduct this study.

References

[1] Jet Propulsion Laboratory, California Institute of Technology, "Surveyor iii mission report: Part i. mission description and performance," Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, NASA Technical Report 32–1777, Sep. 1967.

[2] M. D. Holley, W. L. Swingle, S. L. Bachman, C. J. LeBlanc, H. T. Howard, and H. M. Biggs, "Apollo Experience Report – Guidance and Control Systems: Primary Guidance, Navigation, and Control System Development," National Aeronautics and Space Administration, Lyndon B. Johnson Space Center, Washington, D.C., NASA Technical Note TN D-8227, May 1976.

[3] B. Açıkmeşe 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. DOI: 10.2514/1.27553.

[4] A. R. Klumpp, "Apollo lunar descent guidance," *Automatica*, vol. 10, no. 2, pp. 133–146, 1974, ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(74)90019-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0005109874900193.

[5] B. Açıkmeşe, J. M. C. III, and L. Blackmore, "Lossless convexification of nonconvex 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. DOI: 10.1109/TCST.2012.2237346.

[6] L. Blackmore, "Autonomous precision landing of space rockets," *The Bridge*, vol. 46, no. 4, pp. 16–22, 2016.

[7] A. Botelho, M. Martinez, C. Recupero, A. Fabrizi, and G. D. Zaiacomo, "Design of the landing guidance for the retro-propulsive vertical landing of a reusable rocket stage," *CEAS Space Journal*, vol. 14, pp. 551–564, 2022. DOI: 10.1007/s12567-022-00423-6. [Online]. Available: https://doi.org/10.1007/s12567-022-00423-6.

[8] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 12, no. 1, pp. 1–27, 2012.

[9] J. Yu and J. Wei, *Rocket landing control with grid fins and path-following using mpc*, 2024. arXiv: 2405.16191 [cs.AI]. [Online]. Available: https://arxiv.org/abs/2405.16191.

[10] J. Jang, C. Lee, and S. He, "Convex programming approach of robust powered descent guidance through dynamic tube mpc," English, *ICAS Proceedings*, 2024, Publisher Copyright: © 2024, International Council of the Aeronautical Sciences. All rights reserved.; 34th Congress of the International Council of the Aeronautical Sciences, ICAS 2024 ; Conference date: 09-09-2024 Through 13-09-2024, ISSN: 1025-9090.

[11] L. Blackmore, B. Açıkmeşe, 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. DOI: 10.2514/1.47202.

[12] M. Szmuk, B. Acikmese, 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*. 2018. DOI: 10.2514/6.2016-0378. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2016-0378. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2016-0378.

[13] M. Szmuk and B. Acikmese, "Successive convexification for 6-dof mars rocket powered landing with free-final-time," in *2018 AIAA Guidance, Navigation, and Control Conference*, American Institute of Aeronautics and Astronautics, Jan. 2018. DOI: 10.2514/6.2018-0617. [Online]. Available: http://dx.doi.org/10.2514/6.2018-0617.

[14] B. Gaudet, R. Linares, and R. Furfaro, “Deep reinforcement learning for six degree-of-freedom planetary powered descent and landing,” *arXiv preprint arXiv:1810.08719*, 2018, Accessed from arXiv. [Online]. Available: https://arxiv.org/abs/1810.08719.

[15] E. E. H. SCHURMANN, “Optimum staging technique for multistaged rocket vehicles,” *Journal of Jet Propulsion*, vol. 27, no. 8, pp. 863–865, 1957. DOI: 10.2514/8.12965.

[16] A. D. Koch, “Optimal staging of serially staged rockets with velocity losses and fairing separation,” *Aerospace Science and Technology*, vol. 88, pp. 65–72, 2019, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2019.03.019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963818325641.

[17] B.-U. Jo and J. Ahn, “Optimal staging of reusable launch vehicles considering velocity losses,” *Aerospace Science and Technology*, vol. 109, p. 106 431, 2021, ISSN: 1270-9638. DOI: https://doi.org/10.1016/j.ast.2020.106431. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1270963820311135.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing atari with deep reinforcement learning*, 2013. arXiv: 1312.5602 [cs.LG].

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

[20] T. Salimans, J. Ho, X. Chen, and I. Sutskever, “Evolution strategies as a scalable alternative to reinforcement learning,” *arXiv preprint arXiv:1703.03864*, 2017.

[21] R. Bellman, *Dynamic Programming*. Dover Publications, 1957, ISBN: 9780486428093.

[22] R. Sutton, “Learning to predict by the method of temporal differences,” *Machine Learning*, vol. 3, pp. 9–44, Aug. 1988. DOI: 10.1007/BF00115009.

[23] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[24] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, pp. 229–256, 1992.

[25] V. Konda and J. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[26] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, “Trust region policy optimization,” in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, JMLR, 2015, pp. 1889–1897.

[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018, pp. 1861–1870.

[29] P. Werbos and P. John, “Beyond regression : New tools for prediction and analysis in the behavioral sciences /,” Jan. 1974.

[30] A. M. Turing, “Computing machinery and intelligence,” English, *Mind*, New Series, vol. 59, no. 236, pp. 433–460, 1950, ISSN: 00264423. [Online]. Available: http://www.jstor.org/stable/2251299.

[31] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, Apr. 1992, ISBN: 9780262275552. DOI: 10.7551/mitpress/1090.001.0001. [Online]. Available: https://doi.org/10.7551/mitpress/1090.001.0001.

[32] J. H. Holland, “Genetic algorithms and the optimal allocation of trials,” *SIAM J. Comput.*, vol. 2, no. 2, pp. 88–105, 1973. DOI: 10.1137/0202009. [Online]. Available: https://doi.org/10.1137/0202009.

[33] J. H. Holland, “Genetic algorithms,” *Scholarpedia*, vol. 7, no. 12, p. 1482, 2012. DOI: 10.4249/SCHOLARPEDIA.1482. [Online]. Available: https://doi.org/10.4249/scholarpedia.1482.

[34] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[35] E. Ronald and M. Schoenauer, “Genetic lander: An experiment in accurate neuro-genetic control,” in *Parallel Problem Solving from Nature — PPSN III*, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 452–461, ISBN: 978-3-540-49001-2.

[36] X. Yao, “A review of evolutionary artificial neural networks,” *International journal of intelligent systems*, vol. 8, no. 4, pp. 539–567, 1993.

[37] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[38] T. Bäck, *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996, ISBN: 9780195099713.

[39] K. O. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/?stanley:ec02.

[40] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[41] F. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, “Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning,” *arXiv preprint arXiv:1712.06567*, 2017.

[42] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: Optimization by a colony of cooperating agents,” *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, vol. 26, no. 1, pp. 29–41, 1996.

[43] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *Proceedings of ICNN’95 - International Conference on Neural Networks*, IEEE, 1995.

[44] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence from Natural to Artificial Systems* (Santa Fe Institute Studies in the Sciences of Complexity). Oxford University Press, 1999, ISBN: 0195131592. [Online]. Available: http://www.us.oup.com/us/catalog/general/subject/LifeSciences/Neurobiology/?view=usa%5C&%5C#38;ci=0195131592.

[45] D. Freitas, L. G. Lopes, and F. Morgado-Dias, “Particle swarm optimisation: A historical review up to the current developments,” *Entropy*, vol. 22, no. 3, 2020, ISSN: 1099-4300. DOI: 10.3390/e22030362. [Online]. Available: https://www.mdpi.com/1099-4300/22/3/362.

[46] V. Gudise and G. Venayagamoorthy, “Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks,” in *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS’03 (Cat. No.03EX706)*, 2003, pp. 110–117. DOI: 10.1109/SIS.2003.1202255.

[47] R. Mendes, P. Cortez, M. Rocha, and J. Neves, “Particle swarms for feedforward neural network training,” vol. 2, Feb. 2002, pp. 1895–1899, ISBN: 0-7803-7278-6. DOI: 10.1109/IJCNN.2002.1007808.

[48] E. Mooij, *Re-entry Systems* (Springer Aerospace Technology), English. Springer, 2024, ISBN: 978-3-031-62173-4. DOI: 10.1007/978-3-031-62174-1.

[49] M. Reyhanoglu, “Modeling and control of space vehicles with fuel slosh dynamics,” in *Advances in Spacecraft Technologies*, J. Hall, Ed., Published online February 14, 2011, InTech, 2011, pp. 1–26, ISBN: 978-953-307-551-8. [Online]. Available: http://www.intechopen.com/books/advances-in-spacecraft-technologies/modelling-and-control-of-space-vehicles-with-fuel-slosh-dynamics.

[50] G. F. McDonough, W. D. Murphree, J. C. Blair, *et al.*, “Wind effects on launch vehicles,” The Advisory Group for Aerospace Research and Development, NATO, Slough, England, Tech. Rep. 115, 1970, Library of Congress Catalog Card No. 69-10027.

[51] T. M. Moerland, J. Broekens, and C. M. Jonker, “Model-based reinforcement learning: A survey,” *CoRR*, vol. abs/2006.16712, 2020. arXiv: 2006.16712. [Online]. Available: https://arxiv.org/abs/2006.16712.

[52] G. P. Sutton and O. Biblarz, *Rocket Propulsion Elements*, 9th. Hoboken, NJ: John Wiley & Sons, 2016, ISBN: 978-1-119-17120-4.

[53] D. M. Gaspar, “A Tool for Preliminary Design of Rockets,” M.S. thesis, Faculty of Aerospace Engineering - Technico Lisboa, Lisbon, Portugal, 2014.

[54] A. C. Morelli, C. Giordano, R. Bonalli, and F. Topputo, *Characterization of singular arcs in spacecraft trajectory optimization*, 2023. arXiv: 2311.04123 [math.OC]. [Online]. Available: https://arxiv.org/abs/2311.04123.

[55] G. De Zaiacomo, G. Blanco Arnao, R. Bunt, and D. Bonetti, “Mission engineering for the retalt vtvl launcher,” *CEAS Space Journal*, vol. 14, pp. 533–549, 2022. DOI: 10.1007/s12567-021-00415-y.

[56] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, 2nd. Nob Hill Publishing, 2017, ISBN: 978-0975937743.

[57] S. Tahir, J. Wang, M. Baloch, and G. Kaloi, “Digital control techniques based on voltage source inverters in renewable energy applications: A review,” *Electronics*, vol. 7, p. 18, Feb. 2018. DOI: 10.3390/electronics7020018.

[58] C. Wang and Z. Song, “Convex model predictive control for rocket vertical landing,” in *Proceedings of the 37th Chinese Control Conference*, Wuhan, China, 2018, pp. 9837–9843.

[59] S. Boyd, “Second-order cone programming,” Stanford University, Tech. Rep., 2003, Accessed: 2024-12-02. [Online]. Available: https://web.stanford.edu/~boyd/papers/pdf/socp.pdf.

[60] Y. Mao, D. Dueri, M. Szmuk, and B. Acıkmeşe, “Successive convexification of non-convex optimal control problems with state constraints,” *arXiv:1701.00558v2*, 2017, Accessed: 2024-12-02. [Online]. Available: https://arxiv.org/abs/1701.00558v2.

[61] Z. Shen, S. Zhou, and J. Yu, “Real-time computational powered landing guidance using convex optimization and neural networks,” *arXiv preprint arXiv:2210.07480*, 2022.

[62] S. Adams, T. Cody, and P. A. Beling, “A survey of inverse reinforcement learning,” *Artificial Intelligence Review*, vol. 55, pp. 4307–4346, 2022. DOI: 10.1007/s10462-021-10108-x.

[63] A. Y. Ng and S. J. Russell, “Algorithms for inverse reinforcement learning,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2000, pp. 663–670.

[64] C. M. University, *Recitation 5: A comparative overview of dp vs. mc vs. td*, 2021.

[65] S. Gu, T. P. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, “Q-prop: Sample-efficient policy gradient with an off-policy critic,” *CoRR*, vol. abs/1611.02247, 2016. arXiv: 1611.02247. [Online]. Available: http://arxiv.org/abs/1611.02247.

[66] Y. Jiang, Y. Yang, Z. Lan, *et al.*, “Rocket landing control with random annealing jump start reinforcement learning,” 2024. [Online]. Available: https://arxiv.org/abs/2407.15083.

[67] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[68] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. de Freitas, “Dueling network architectures for deep reinforcement learning,” in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, PMLR, 2016, pp. 1995–2003.

[69] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI Press, AAAI Press, 2016, pp. 2094–2100.

[70] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[71] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1707.01495.

[72] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015. [Online]. Available: https://arxiv.org/abs/1509.02971.

[73] F. Sehnke, C. Osendorfer, T. Rückstieß, *et al.*, "Parameter-exploring policy gradients," *Neural Networks*, vol. 23, no. 4, pp. 551–559, 2010. DOI: 10.1016/j.neunet.2009.12.004.

[74] T. Rückstieß, M. Felder, and J. Schmidhuber, "State-dependent exploration for policy gradient methods," in *European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2008, pp. 234–249. DOI: 10.1007/978-3-540-87481-2_31.

[75] M. Plappert, R. Houthooft, P. Dhariwal, *et al.*, "Parameter space noise for exploration," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://arxiv.org/abs/1706.01905.

[76] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.

[77] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Exploration by random network distillation," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://arxiv.org/abs/1810.12894.

[78] M. Fortunato, M. G. Azar, B. Piot, *et al.*, "Noisy networks for exploration," *CoRR*, 2017. arXiv: 1706.10295. [Online]. Available: http://arxiv.org/abs/1706.10295.

[79] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, JMLR Workshop and Conference Proceedings, vol. 32, 2014, pp. 387–395.

[80] G. Barth-Maron, M. W. Hoffman, D. Budden, *et al.*, "Distributed distributional deterministic policy gradients," in *Proceedings of the International Conference on Learning Representations (ICLR)*, DeepMind, 2018.

[81] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, PMLR, 2018, pp. 1587–1596.

[82] A. Wahid, A. Stone, K. Chen, B. Ichter, and A. Toshev, "Learning object-conditioned exploration using distributed soft actor critic," *arXiv preprint arXiv:2007.14545*, 2020.

[83] D. Akimov, "Distributed soft actor-critic with multivariate reward representation and knowledge distillation," *arXiv preprint arXiv:1911.13056*, 2019.

[84] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," *arXiv preprint arXiv:1806.06920*, 2018.

[85] H. Prior, *Jax-rl: Mpo.py*, https://github.com/henry-prior/jax-rl/blob/master/jax_rl/MPO.py, Accessed: 22-Nov-2024, 2024.

[86] M. W. Hoffman, B. Shahriari, J. Aslanides, *et al.*, "Acme: A research framework for distributed reinforcement learning," *arXiv preprint arXiv:2006.00979*, 2020. [Online]. Available: https://arxiv.org/abs/2006.00979.

[87] M. Andrychowicz, A. Raichuk, P. Stanczyk, *et al.*, "What matters in on-policy reinforcement learning? a large-scale empirical study," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=r1etN1rtPB.

[88] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization," in *Advances in Neural Information Processing Systems*, vol. 4, Morgan Kaufmann, 1992, pp. 950–957. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1991/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf.

[89] Space Exploration Technologies Corp., *Starship users guide, Revision 1.0*, Accessed 4 May 2025, SpaceX, Hawthorne, California, Mar. 2020. [Online]. Available: https://www.spacex.com/media/starship_users_guide_v1.pdf.

[90] Society of Amateur Rocketry and Space Modelling, “TIR-33 rocket stability and control manual,” Space Modeling Association, Technical Information Report TIR-33, 1998, Accessed: 2025-05-06. [Online]. Available: https://www.spacemodeling.org/jimz/manuals/tir-33.pdf.

[91] J. D. Anderson, *Fundamentals of aerodynamics*, 5th. McGraw-Hill, Feb. 2011. [Online]. Available: http://www.worldcat.org/isbn/9780073398105.

[92] W. D. Washington and M. S. Miller, “Grid fins – a new concept for missile stability and control,” in *Proceedings of the 31st Aerospace Sciences Meeting & Exhibit*, Reno, Nevada, 1993. DOI: 10.2514/6.1993-35.

[93] J. D. A. Junior, *Introduction to Flight*, 8th. McGraw-Hill Education, 2020, ISBN: 978-1260012383.

[94] R. Serway and J. Jewett, *Physics for Scientists and Engineers with Modern, Chapters 1-46*. Cengage Learning, 2009, ISBN: 9781111788957. [Online]. Available: https://books.google.nl/books?id=FRkFAAAAQBAJ.

[95] D. Whitley, S. Rana, and R. B. Heckendorn, “Island model genetic algorithms and punctuated equilibria,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 1999, pp. 548–555.

[96] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[97] F. Bergh and A. Engelbrecht, “A cooperative approach to particle swarm optimization,” *Evolutionary Computation, IEEE Transactions on*, vol. 8, pp. 225–239, Jul. 2004. DOI: 10.1109/TEVC.2004.826069.

[98] B. Clow and T. White, “An evolutionary race: A comparison of genetic algorithms and particle swarm optimization for training neural networks.,” Jan. 2004, pp. 582–588.

[99] S. Panda and N. P. Padhy, “Comparison of particle swarm optimization and genetic algorithm for facts-based controller design,” *Applied Soft Computing*, vol. 8, no. 4, pp. 1418–1427, 2008, Soft Computing for Dynamic Data Mining, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2007.10.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494607001330.

[100] C. A. Souza Lima, C. M. F. Lapa, C. M. do N.A. Pereira, J. J. da Cunha, and A. C. M. Alvim, “Comparison of computational performance of ga and pso optimization techniques when designing similar systems – typical pwr core case,” *Annals of Nuclear Energy*, vol. 38, no. 6, pp. 1339–1346, 2011, ISSN: 0306-4549. DOI: https://doi.org/10.1016/j.anucene.2011.02.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306454911000612.

[101] F. Foundation, *Gymnasium: A standard api for reinforcement learning*, https://github.com/Farama-Foundation/Gymnasium, Accessed: 2025-05-28, 2022.

[102] A. Y. Ng, *Shaping and policy search in reinforcement learning*. University of California, Berkeley, 2003.

[103] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, “Robust adversarial reinforcement learning,” *arXiv preprint arXiv:1703.02702*, 2017.

[104] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, Curran Associates, Inc., 2016, pp. 4565–4573. [Online]. Available: http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf.

[105] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, N. Ernestus, and M. Dormann, *Stable-baselines3: Reliable reinforcement learning implementations*, https://github.com/DLR-RM/stable-baselines3, Accessed: 2025-05-28, 2021.

A

Appendix

A.1. Project Planning

This section covers the planning process of the project. The work was divided into two main research phases, culminating in distinct milestones: a midterm and greenlight review. Research Phase I aimed to demonstrate feasibility in answering the research questions and build an environment for policy learning. Phase II aimed to answer the research questions directly.

First, the milestones for writing an MSc thesis at the Faculty of Aerospace Engineering, for the Control and Simulation profile, are outlined in Subsection A.1.1, before the Work Packages (WPs) of each phase are covered in Subsection A.1.2. Finally, the Gantt charts illustrating the project's progression are presented in Subsection A.1.3.

A.1.1. Milestone planning

The research project has scheduled milestones to guide the student to a structured thesis. They are presented in order of appearance below

1. **Initial research proposal**

   A research proposal document was drafted that proposed the application of reinforcement learning to the task of rocket landing, highlighting the project's relevance. Later, it went on to briefly outline broad groups of methods or tools that could be used, along with an initial Gantt chart. The research proposal enabled a more targeted literature review and ensured the development of novel findings.

2. **Literature study**

   The literature study reviewed academic articles relevant to the topic and documented reusable rockets active in the industry. The first draft of the literature study was purposely made broad to give the author a wide view of the methods available to help solve the rocket landing control problem.

3. **Research proposal**

   Conducting the literature review provided valuable insights for finalising the research proposal and project planning. Here, the research questions, Gantt chart, and Work Breakdown Structures (WBS) were developed to provide a structured framework for the project. Also, options were provided in case implementation was easier or more complicated than expected.

4. **Mid term review**

   The end of the first research phase culminated in a *Midterm review*, where the work was presented to the project supervisor. The mid-term review aimed to show the initial feasibility of the project. Following the midterm review, the final plan was updated, and the research questions were reviewed in light of the insights uncovered.

5. **Greenlight review**

   The greenlight review was the presentation of the draft thesis, which aimed to be as complete as possible. The research questions had to be addressed for the greenlight review, and an additional focus was to be placed on the culminating weeks of writing the article and discussing the results.

A.1.2. Work packages

Following the draft of the literature study, an initial plan was made. Here, the work was split into two phases. Phase I lasted until the mid-term review, to provide proof of concept for the method and demonstrate its potential to answer the research questions. Following this, Phase II would fully implement the technique to answer the research questions. This section will cover the initial WPs devised for each phase, as well as the findings and supervisor feedback that led to alterations to the WPs.

Project kick-off and literature study

The thesis began by identifying and selecting an application of EAs or RL within the aerospace domain that could yield novel findings. Consequently, the broad idea of applying RL to reusable launch vehicles was decided upon and presented at the *kick-off meeting*, along with a project description crafted in tandem. The initial RL solution aimed to cover fault-tolerance and interpretability; however, feedback from the kick-off meeting indicated that the targeted task was too broad for this timeframe.

The literature study began by dividing the survey into three groups, following the creation of a preliminary research definition —a smaller version of the research proposal — to identify topics that would be covered in the literature study.

- **LIT.1**: Reinforcement learning. Aiming to cover the main groups of methods used within RL, starting from a foundational basis. Secondly, to research methods which increase learning efficiency and stability due to constrained computational resources. Finally, to culminate in an initial algorithm decision, best suited to the research problem.
- **LIT.2**: Rocket landing GNC. Firstly, it would cover techniques and launch vehicles used in the industry, before covering trajectory optimisation methods for a rocket landing control system.
- **LIT.3**: Launch vehicle modelling. This would begin with research models from the literature that are suitable for this study. After that, higher-fidelity modelling opportunities, such as variable aerodynamics and fuel sloshing, would be covered.

Research phase I

WP1: Algorithm feasibility study. Following the literature review, an MPO skeleton was chosen due to the author's statement [84] on its invariance to hyperparameter choices. The PER buffer was chosen as the buffer for increased sample efficiency. This provided *Level A* of the algorithm, to be tested on the continuous Lunar Lander environment for a functionality test. Afterwards, n-step returns were to be included to aid policy learning with longer time horizon problems, creating *Level B* to conduct feasibility testing, first on the Lunar Lander environment and then on MuJoCo's more complex inverted pendulum, to assess performance in an environment with more complex dynamics. Finally, *Level C* aimed to provide robustness testing on MuJoCo's inverted pendulum through static parameter perturbations using Gaussian distributions, and dynamic perturbations via Robust Adversarial Reinforcement Learning (RARL). Ultimately, the aim was to prove the MPO's ability to scale to environments with complex dynamics and disturbances.

Time: 18 working days were assigned, excluding a three-week break over Christmas.

Changes: First, following the literature review, it was recommended to reduce the implementation complexity by removing RARL. Second, although the authors motivate MPO from its insensitivity to hyperparameters, it was found to be difficult to set initial hyperparameters. This resulted in SAC being trialled due to its more straightforward implementation, more proven applications to continuous control problems and fewer hyperparameters. Moreover, each test environment may have required independent hyperparameter tuning, while not replicating the flight dynamics or constraints of the rocket landing problem. Consequently, the decision was made to directly test the implementation in a reduced-fidelity rocket landing environment, following the Lunar Lander

problem. The Lunar Lander problem exhibited increasing rewards but did not converge due to insufficient runtime. This was later solved in WP6 and documented in Section 4.5.

WP2: Primary simulation environment. This WP aimed to build a 6-DoF simulation in stages, gradually increasing the model's fidelity. First, a low-fidelity implementation was to be built for a 3-DoF environment, considering a 2D plane (x, y) with only pitch rotation. This would remove roll control from the problem. Next, actuator dynamics were to be implemented for the rocket, consisting of grid fins, TVC, and cold-gas thrusters. The atmosphere and gravity models were to be taken from literature for three environments of the Earth, Mars and the Moon. Finally, the WP incorporated higher-fidelity dynamics of a 6-DoF model, fuel sloshing, and variable aerodynamic coefficients. An additional task of integrating with the RARL's dynamic disturbances was included too.

Time: 18 working days were assigned for this task, to be ran in parallel with WP3 of verification and validation.

Changes: The primary struggle of creating the environment was finding realistic parameters and initial conditions for a feasible and representative rocket landing. Open-source models lacked the necessary fidelity and the ability to interface easily with a reinforcement learning agent. As such, a custom environment had to be built, requiring rocket staging and sizing, followed by a mock ascent trajectory that provided the initial conditions for the first-stage descent. This took a total of 16 full working days to create the initial environment. Additionally, the ascent reference profile originally took 4 working days to create, through a non-optimised ascent reference representative of a reusable launch vehicle's flight to LEO [7]. Finally, following feedback on the project proposal, the environment was simplified to only Earth.

WP3: Verification and Validation. The three subtasks were writing a verification and validation plan, performing verification and performing validation. This WP was to be performed in parallel with WP2.

Changes: Verification and validation were found to be a continual work package throughout the project, with a formal documentation stage near the end of Research Phase II. However, in hindsight, it would have been best to have documentation run in parallel to all the tasks, too. Moreover, to validate the sources of failure, the SAC from StableBaselines3 ([105]), a validated implementation, was used to provide a comparison and showed the same problems of non-convergence as the original SAC algorithm on the ascent problem, implying the issue was not necessarily tied to the SAC implementation or inadequate verification. To validate the source of failure, a non-gradient-based PSSO algorithm was trialled in place of the RL algorithm to tune the parameters of a neural network. This reduced the problem to the reward function and truncation logic. The end solution was an additional tool to decouple from gradient-based policy learning. This approach follows the principle of *modular falsification*, a form of systemic isolation in which one part is replaced with a working part to isolate the problem. This helped deduce faults in an unreachable reference in the reward function and the truncation logic of the ascent phase.

Research Phase II

WP4: Baseline controllers. One of the initial research questions was to determine the feasibility of mimicking traditional trajectory optimisers through RL, such as lossless convex optimisation and MPC. MPC was chosen to be mimicked because, in theory, a finite time horizon requires less computation for a solution. As such, the first step was to validate an MPC for trajectory tracking, before determining if computational resources were sufficient to mimic it.

Time: 8 days were assigned, with 5 assigned to validate the MPC's implementation, 1 to determine feasibility and an extra 2 days to document the work.

Changes: The literature study on which the original project plan was based lacked sufficient information on the individual flight phases of a rocket during descent. More specifically, objectives, actuators and operational constraints dependent on each phase were not adequately addressed. Consequently, WP4 was subdivided due to the need for independent controllers for each phase. This required 3 days for the ascent phase controllers and then 10 days for the descent controllers, including one day for incorporating the ACS model. Regarding the mimicking of an MPC, it was noted how long RL takes. Given the limited computational resources available (Subsection 1.2.2),

it was deemed too time-intensive to imitate and could be moved to a future work recommendation.

WP5: Agent and environment incorporation. The agent and environment were to be combined, with a reward function set up to mimic the MPC.

Time: 9 days were assigned, with 5 for the combination, 2 for metric implementation, 2 for additional verification and validation, and finally 1 day for documentation.

Changes: As WP2 transitioned to directly testing the RL algorithm in a simple environment, these features were already combined along with metrics, which were required to demonstrate progress during weekly progress meetings. Additional verification and validation were conducted, taking 17 days, which was significantly longer than initially estimated. Furthermore, supervisory learning of controllers to kick-start learning was tested as an alternative to apprenticeship imitation learning of MPC. However, there was limited policy improvement. Furthermore, two days were assigned to validate a feasible powered descent scenario.

WP6: Experiments. These were divided into three groups:

- Perfect model performance.
- Robustness testing to static and dynamic disturbances.
- Agent's adaptability to a higher fidelity model.

Time: 25 days were assigned, with the majority, 16 days, allocated to facilitate the higher-fidelity model transfer experiments.

Changes: Experiments regarding the powered descent phase were conducted during the final five weeks, focusing on achieving perfect model performance with higher fidelity features such as variable aerodynamic coefficients. SAC struggled to learn a policy, as explained in Chapter 2 and Chapter 5, resulting in the PSSO being experimented with, as motivated by literature [20], [41]. Additionally, a boostback terminal velocity procedure was developed, and a wind disturbance model was incorporated; however, insufficient time was left, making it infeasible to test them within the environment.

A.1.3. Gantt charts

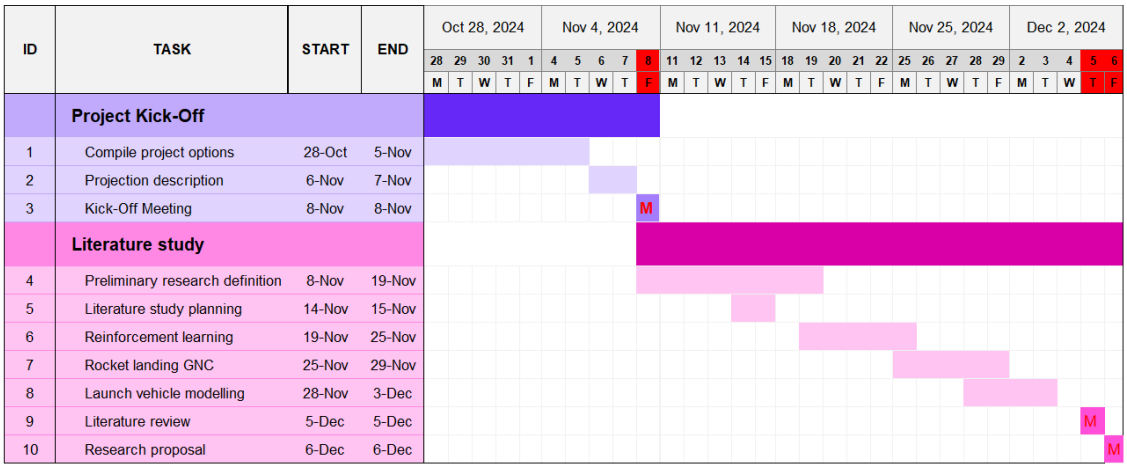


Figure A.1: Gantt chart from the beginning of the project to the literature review. Showcasing key milestones of the kick-off meeting, literature review and research proposal hand-in.

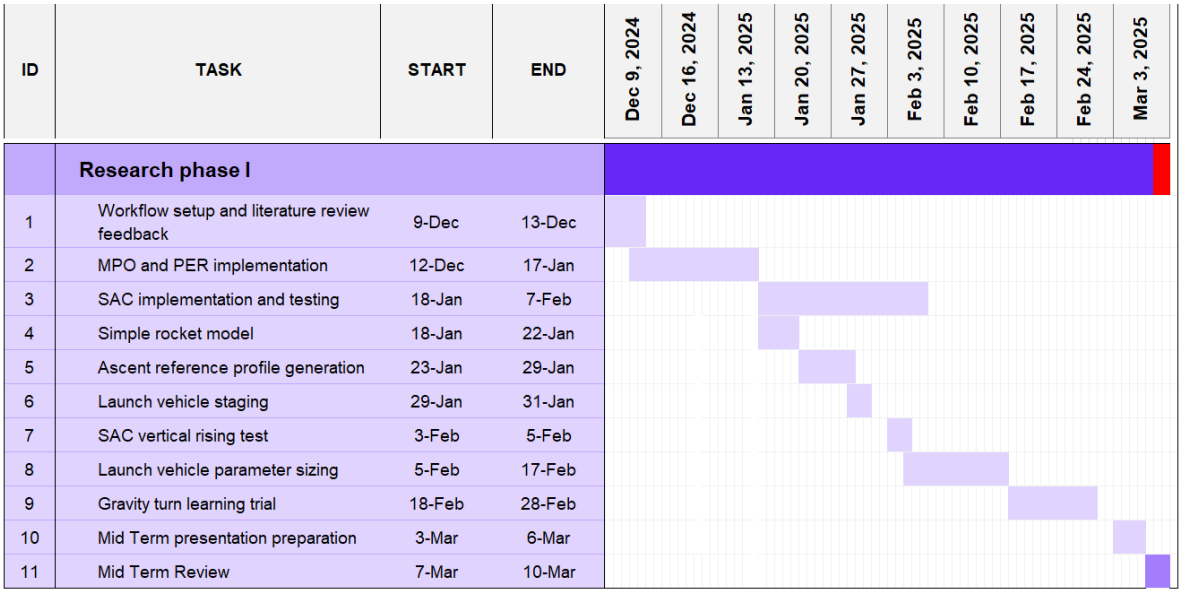


Figure A.2: Gantt chart followed for Research Phase I, finalising with the midterm review.

ID	TASK	START	END
	Research Phase II		
1	StableBaslines3 verification	17-Mar	18-Mar
2	PSO and GA implementations	18-Mar	20-Mar
3	Classical controllers for first stage ascent burn	21-Mar	24-Mar
4	PSO and GA NE for first stage ascent burn	24-Mar	28-Mar
5	First stage descent classical controllers and mock references, excluding final powered descent phase	31-Mar	11-Apr
6	ACS preliminary modelling, and RCS modelling	7-Apr	8-Apr
7	Supervisory learn controllers untill powered descent phase	14-Apr	16-Apr
8	Immitation learning of controls through SAC	17-Apr	22-Apr
9	Formal verification and validation	17-Apr	28-Apr
10	TD3 implementation	28-Apr	1-May
11	RL algorithm: reward shaping, normalisation, hyperparameter tuning and truncation logic.	30-Apr	8-May
12	Landing burn feasibility check through control systems.	6-May	8-May
13	Landing burn supervisory learning to kick start test	8-May	9-May
14	Validated aerodynamics and ACS model upgrades. Aerodynamic stability check.	12-May	15-May
15	Boostback burn terminal velocity derivation	15-May	16-May
16	Powered descent RL (experiments, ablation study and post-processing)	19-May	5-Jun
17	Neuroevolution	28-May	2-Jun
18	Wind model	29-May	29-May
19	Documentation (at least 50% of day)	15-May	9-Jun
20	Thesis first draft	10-Jun	10-Jun

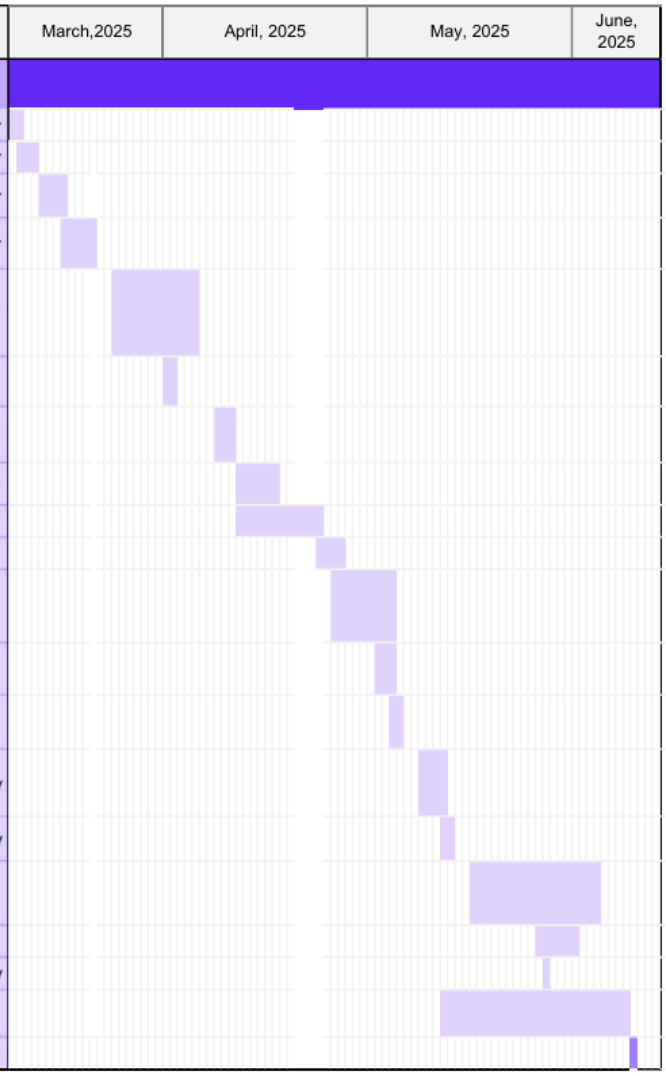


Figure A.3: Gantt chart followed for Research Phase II, finalising with a draft thesis submission.

A.2. Manoeuvrable grid fin model

The local angle of attack is the angle of the grid fin to the perpendicular direction of the flow. Equation A.1 shows the local angle of attack for the left and right grid fins, dependent on the effective angle of attack (descent angle of attack) and their respective deflection angles. The deflection angles go counter-clockwise, so an upward right grid fin deflection and a negative left grid fin deflection are positive.

$$\begin{aligned} \alpha_{l,R} &= \alpha_{eff} - \delta \\ \alpha_{l,L} &= \alpha_{eff} - \delta \end{aligned} \tag{A.1}$$

With the angle of attack acting on each grid fin defined, the normal and axial forces can be found from Figure A.4. First, the axial forces are the same for each grid fin as they are only dependent on Mach number. Second, the grid fins in the centre of our 2D rocket are only modelled to give axial force, as the third dimension is not considered.

$$\begin{aligned} F_a &= q \cdot S \cdot C_a(M) \\ F_{N,L} &= q \cdot S \cdot C_n(M, \alpha_{l,L}) \\ F_{N,R} &= q \cdot S \cdot C_n(M, \alpha_{l,R}) \end{aligned} \tag{A.2}$$

The freebody diagram of the rocket during descent with grid fins is shown in Figure A.4. From this, the body frame (x", y") forces are derived in Equation A.3.

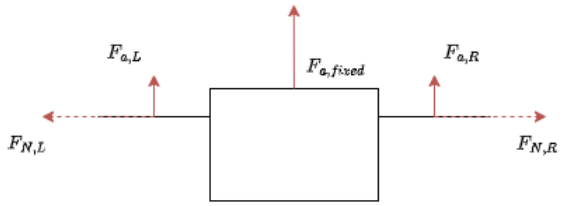


Figure A.4: Aerodynamic Control Surfaces freebody diagram during descent.

$$\begin{aligned} F_{x''} &= F_a \cdot (\sin(\delta_R) - \sin(\delta_L)) - F_{N,L} \cdot \cos(\delta_L) + F_{N_R} \cdot \cos(\delta_R) \\ F_{y''} &= F_a \cdot (2 + \cos(\delta_L) + \cos(\delta_R)) - F_{N,L} \cdot \sin(\delta_L) + F_{N,R} \cdot \sin(\delta_R) \\ M_z &= -(d_{gf} - d_{cg}) \cdot \left(F_a \cdot (\sin(\delta_R) - \sin(\delta_L)) - F_{N,L} \cdot \cos(\delta_L) + F_{N_R} \cdot \cos(\delta_R) \right) \\ &\quad + r_r \cdot \left(F_a \cdot (\cos(\delta_R) - \cos(\delta_L)) + F_{N,R} \cdot \sin(\delta_R) - F_{N,L} \cdot \sin(\delta_L) \right) \end{aligned} \tag{A.3}$$