Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

M'hamed Bougara University of BOUMERDES

Faculty of Sciences

Computer Science Department

Autonomous Lunar Lander Optimization with Reinforcement Learning

Author(s):
Massil Meziane Mzir
Lyna Bouharoun
Alae Khelifi

Advisor:
Abdelhak Saouli

Examinator:
Hadjer Boughanem

May 22, 2025

Abstract

This study presents the development of an intelligent control system for an autonomous lunar lander using reinforcement learning techniques. The goal is to ensure safe, efficient lunar landings in dynamic and uncertain environments. The work explores several algorithms: starting with tabular Q learning, followed by Deep Q Learning (DQL), and Proximal Policy Optimization (PPO). These methods are tested in a physics based lunar simulation to evaluate performance based on landing precision, fuel efficiency, and reliability. The study compares the effectiveness and computational cost of each approach to support future autonomous lunar missions. c **Keywords:** Autonomous Lunar Lander, Reinforcement Learning, PPO (Proximal Policy Optimization), Q Learning, Q Table, Q Network, Deep Q Learning (DQL), Lunar Simulation.

Résumé

Cette étude présente le développement d'un système de contrôle intelligent pour un atterrisseur lunaire autonome, en utilisant des techniques d'apprentissage par renforcement. L'objectif est d'assurer des atterrissages lunaires sûrs et efficaces dans des environnements dynamiques et incertains. Le travail explore plusieurs algorithmes : en commençant par le Q learning tabulaire, suivi du Deep Q Learning (DQL), et de l'optimisation de politique proximale (PPO). Ces méthodes sont testées dans une simulation lunaire basée sur la physique pour évaluer les performances en termes de précision d'atterrissage, d'efficacité énergétique et de fiabilité. L'étude compare l'efficacité et le coût computationnel de chaque approche afin de soutenir les futures missions lunaires autonomes.

Mots-clés : Atterrisseur lunaire autonome, Apprentissage par renforcement, PPO (Optimisation de politique proximale), Q Learning, Table de Q, Réseaux Q, Deep Q Learning (DQL), Simulation lunaire.

Acknowledgments

As a preamble to this thesis, we would like to thank God, who has helped us and granted us patience and courage throughout these long years of study.

Our sincere thanks go to the jury member, Dr. Boughanem Hadjer, for having accepted to examine and evaluate this modest work.

We express our profound gratitude to Dr. SAOULI Abdelhak, our supervisor, for introducing us to a field of significant contemporary relevance, and for his trust, guidance, and continued support throughout the course of this project. His insightful advice and constructive feedback have been invaluable to our work. We sincerely acknowledge his unwavering commitment to student development and his dedication to the advancement of academic research.

We also extend our sincere thanks to all those who contributed to the development of our thinking and interpretation throughout this modest work. Our deep appreciation goes as well to all the faculty members and administrative staff of the Faculty of Sciences, whose dedicated efforts have provided us with a high-quality education.

Finally, we would like to express our gratitude to our friends and fellow classmates for the many enjoyable moments we shared together throughout our university journey.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
TVC	Thrust Vector Control
RCS	Reaction Control System
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
MDP	Markov Decision Process
Q-Value	Valeur Q
DQN	Deep Q-Network
PPO	Proximal Policy Optimization
SARSA	State Action Reward State Action (an RL algorithm)
DDPG	Deep Deterministic Policy Gradient
TD	Temporal Difference (learning method)
GPU	Graphics Processing Unit
CPU	Central Processing Unit
FPS	Frames Per Second
RLHF	Reinforcement Learning from Human Feedback
PID	Proportional-Integral-Derivative controller
NN	Neural Network
V(s)	Value Function of State
Q(s,a)	Action-Value Function (Q-Value)
s	State
a	Action
r	Reward
P	Transition Probabilities
R	Reward Function
S	Set of States
A	Set of Actions
B	Batch Size
C	Target Network Update Frequency
M	Maximum Number of Episodes
T	Maximum Steps per Episode
GAE	Generalized Advantage Estimation
TD error	Temporal Difference Error
arg max	Argument of Maximum (action selection operator)
ReLU	Rectified Linear Unit (activation function)
RGB	Red Green Blue (color channels in images)
GIF	Graphics Interchange Format (animated image format)
Deque	Double-ended Queue (data structure for frame stacking)

General Introduction

A spaceship that lands safely and efficiently is one of the major challenges in aerospace engineering and space exploration. The process requires navigating through complex environmental conditions that can significantly affect security and accuracy. Whether landing on the dusty plains of Mars, returning to the rocky surface of the Moon, or in the dense environment of the Earth, each astronomical body presents unique challenges. These challenges consist of intense gravitational forces, erratic weather conditions, changes in the atmosphere, and uneven landscapes that require accurate calculations and immediate adjustments.

In the early days of space exploration, engineers relied on basic landing techniques such as parachutes and airbags to cushion the impact of spacecraft. For example, the Mars Pathfinder mission used airbags that allowed the lander to bounce to a stop, shielding it from the force of landing. However, this method was risky and often caused significant damage, leading to mission failures and jeopardizing delicate equipment. In fact, these engineering challenges led experts to develop reliable landing methods, which resulted in the creation of propulsive landing technology. **SpaceX** for instance, gave us an important technological advancement in 2015 with Falcon 9 using rocket engines to manage its landing for accurate and secure ones. The **Chopstick** approach is one of the advanced landing technologies that uses robotic arms to catch spacecraft mid-air for a gentle and controlled landing. The shift from traditional airbag landings to self-operating catching mechanisms represents a significant advancement in space exploration technology.

The current wave of advances in AI across industries marks the next phase in spaceship landing technology, which will operate completely autonomously without human intervention. Reinforcement learning **(RL)** emerges as a changing force for landing technologies. It allows spacecraft to adjust their navigation to any environment, leading to the development of fully autonomous landings that can operate in any terrain or weather condition. The recent discovery promises to revolutionize aerospace exploration by initiating a new era in space history while ensuring safer and more effective interplanetary travel. The capacity of RL agents to learn through experience and evolve over time positions them as perfect tools for managing unpredictable and dynamic conditions in space environments.

This excerpt outlines a research project applying Reinforcement Learning to spacecraft landing technology using the OpenAI Gym Lunar Lander environment. The researchers aim to develop an RL agent that can master complex landing scenarios, demonstrating RL's potential to advance autonomous landing systems for space exploration with broader industry applications. The thesis is organized into four chapters ,chapter(1) fundamental rocket landing physics and challenges in low-gravity environments; chapter(2) a review of relevant RL approaches including Q-Learning, Deep Q-Learning, and PPO; chapter(3) methodology covering simulation environments, learning strategies, and agent design; and chapter(4) experimental results and analysis. The work concludes with a summary and directions for future research.

Chapter 1

Introduction

1 Introduction

As we embark on this journey, our focus will unravel the intricate challenge of rocket landing, exploring the realms of Artificial Intelligence (AI), Machine Learning, Reinforcement Learning, and Optimization. This chapter serves as the gateway to our discussion, emphasizing the critical importance of achieving precise and efficient rocket landings, particularly in the context of space exploration and autonomous control systems.

2 Fundamentals of Rocket Landing

2.1 Principles of Rocket Landing in Low Gravity Environments

A low gravity environment is a setting or an area characterized by a gravitational force weaker than the 9.807 m/s^2 experienced on Earth. This includes both low gravity and microgravity environments. A microgravity environment is one where the gravitational force is nearly zero, although it is not absolute zero gravity; rather, it is a condition in which gravity is very weak, where it indicates accelerations equivalent to one millionth (10^{-6}) of the force of gravity at Earth's surface [6] . However, a low gravity environment is much weaker than Earth but not an absolute zero. These conditions can be established in various environments that we are familiar with, such as the Moon, which represents 1/6 of Earth's gravity, and Mars, which has about 38% of Earth's gravity.

What makes these environments particularly interesting is the impact of such conditions on objects and their movements, where they behave very differently compared to Earth. In low gravity environments, landing a spacecraft is significantly different from landing on Earth, as illustrated in Figure 1.1. With weaker gravitational forces, spacecraft land slowly, which requires careful control of propulsion to prevent excessive bouncing or drifting when reaching the surface. Since atmospheric drag is often minimal or absent, rockets must depend entirely on thrusters to decelerate and stabilize during landing. To achieve a precise landing, spacecraft use retropropulsion, firing downward facing thrusters to counteract gravity and control landing speed. In addition, thrust vectoring and reaction control systems help to orient halfway through the landing to maintain the lander on a stable trajectory. Fuel economy is also significant. Even a slight variation in velocity (ΔV) can become significant in low gravity environments. As such, every maneuver must be carefully designed so that fuel is utilized to the maximum while providing a controlled and smooth descent. Understanding the unique characteristics of

low gravity environments is the starting point for discovering how spaceships and other systems

operate in space. With this background, we can now narrow our focus to the specific issue of a spaceship landing in such environments,

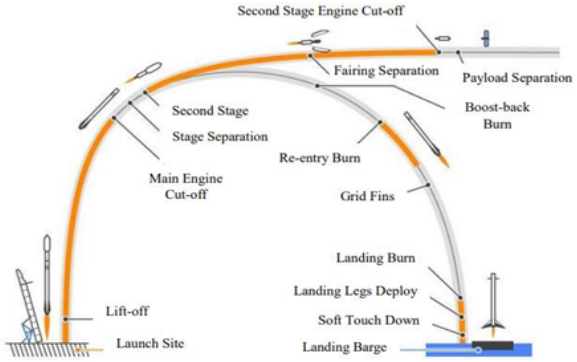

Figure 1.1: Landing a spaceship [4]

2.2 Landing Spacecraft in Low Gravity

Spacecraft landing in low gravity is highly challenging as opposed to the Earth's. It requires specialized techniques and systems to ensure a controlled, safe landing. There are several challenges that make the process greatly complicated. Less gravity makes descending different, and the lack of atmosphere on bodies like the Moon eliminates air resistance to slow it down. Landing stability is harder to guarantee with the less potent forces that are involved and requires advanced technology to offset even slight deflections. The explanation is laid out below, and its impact is discussed in relation to landing in space.

The most critical challenge is the more gradual fall through weaker gravity. The weaker gravity pull makes the spacecraft fall much more slowly than on Earth. This requires longer and larger burns by the engines to properly control the descent. Another problem is the absence of aerodynamic drag. In most low gravity environments, such as the Moon, there is no atmosphere. Without air resistance, the spacecraft cannot utilize parachutes or aerobraking to slow down. Instead, it will have to depend only on propulsion systems for slowing down. Stability is a concern as well. With fewer restoring forces, it is more difficult to stabilize the spacecraft. Small translations or rotations must be corrected by advanced systems such as:

- **TVC**: Adjusts the direction of engine thrust to stabilize. see Figure 1.2 for a representation.[17]
- **RCS**: Makes fine position and attitude corrections using small thrusters in the spacecraft.[16]

Even minor disturbances cause serious orientation problems. Other factors that make landings difficult include:

- **Uneven terrain and barriers**: Boulders, slopes, or rough terrain increase the likelihood of a failure landing.
- **Live location of landing site**: The space vehicle must survey and make accommodations for potential landing sites along the way to the landing.

Weakened gravity also creates additional challenges:

- **Increased longer burning of engines**: Additional longer thrust required to counteract low gravity, consuming more fuel.

- **Fuel control**: Excessive fuel usage may lead to overheating, whereas low fuel will result in a crash.

Successful landing in a low gravity environment requires precise engineering, careful planning, and using complex stabilization systems. in order to surpass these problems.

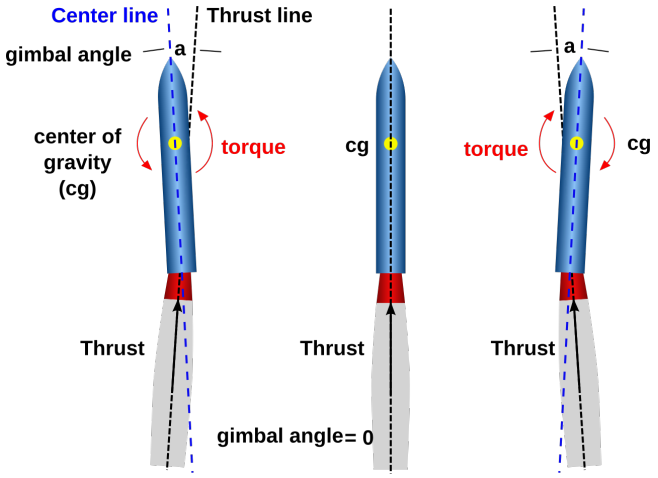

Figure 1.2: TVC [15]

3 Reinforcement Learning for Rocket Landing

3.1 Deep Learning

Deep learning is a branch of artificial intelligence and a subset of machine learning, inspired by how the human brain processes information As shown in Figure 1.3. The brain consists of interconnected neurons that recognize patterns, make decisions, and learn from experience. In artificial intelligence, this concept is applied through neural networks, which allow machines to process vast amounts of data and extract meaningful patterns without explicit programming. These networks have revolutionized AI by enabling models to automatically recognize features in data, making them highly effective for complex tasks.

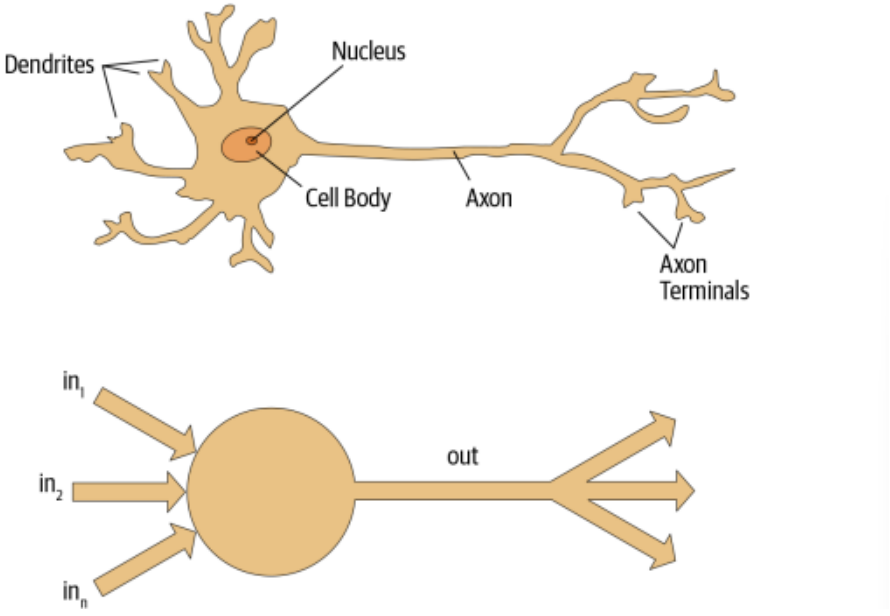

Figure 1.3: Natural and artificial neurons [5]

Neural networks come in different types, each designed for specific applications. Artificial Neural Networks (ANNs) are the most basic form, consisting of multiple layers of interconnected nodes that process information and identify patterns. While effective for simpler problems, they struggle with high dimensional data that requires deeper feature extraction. To address this limitation, Convolutional Neural Networks (CNNs) were developed, primarily for image processing. CNNs use specialized layers to automatically detect important features like edges, textures, and shapes, making them highly effective in tasks such as object detection, medical image analysis, and facial recognition. Their ability to extract spatial features has made them the backbone of modern computer vision applications.

Nowadays, science hasn't stopped here. While deep learning models typically rely on large, labeled datasets to improve performance through training, we can now work even when such data isn't available. Reinforcement Learning (RL) provides a different approach. Instead of learning from pre existing data, RL enables models to learn through direct interaction with an environment, making decisions based on trial and error while optimizing for rewards. This approach has been crucial in developing AI for robotics, self driving cars, financial modeling, and even training autonomous agents to perform complex tasks like playing strategic games or landing spacecraft. By combining deep learning techniques with reinforcement learning, AI continues to evolve, allowing machines to achieve remarkable levels of autonomy and intelligence.

3.2 Reinforcement Learning

Reinforcement learning (RL) is a machine learning approach where an agent learns to make decisions by interacting with an environment. The agent takes actions, receives feedback in the form of rewards, and adjusts its strategy to maximize cumulative rewards over time.

In RL, the main components are:

- **Agent** – The decision-maker that interacts with the environment.
- **Environment** – The world in which the agent operates.
- **State (s)** – The current situation or condition of the environment.
- **Action (a)** – The decision the agent takes at a given state.
- **Reward (r)** – The feedback received after taking an action (positive for good actions, negative for bad ones).
- **Policy (π)** - The strategy that the agent follows to choose actions based on states.
- **Value Function (V(s))** – Estimates the long-term reward expected from a state.
- **Q Value (Q(s, a))** – Estimates the long-term reward of taking an action in a given state.

The goal is to find an optimal policy that maximizes the total reward, allowing the agent to learn effective landing strategies through trial and error.[12]

3.3 Markov Decision Process in Lunar Lander

The Lunar Lander environment is modeled using a Markov Decision Process (MDP), a mathematical framework for decision making in situations where outcomes are partly random and partly under the agent's control. An MDP is defined by a tuple (S, A, P, R, γ):

- **S (States)** – All possible configurations of the Lunar Lander, including its horizontal and vertical positions, velocities, angle, and angular velocity.
- **A (Actions)** – The set of actions available to the agent — such as firing the main engine (upward thrust), firing the left or right thrusters, or doing nothing.
- **P (Transition probabilities)** – The probability of moving from one state to another given a certain action. The dynamics of the lander's movement follow physical laws, but external forces like gravity and wind add uncertainty.
- **R (Rewards)** – The immediate feedback received after each action. For the Lunar Lander:
  - A positive reward for landing within a target zone.
  - A negative reward for crashing or drifting too far from the landing pad.
  - Additional penalties for excessive fuel use or erratic movements.
- γ **(Discount factor)** – A value between 0 and 1 that balances the importance of immediate versus future rewards. A higher γ encourages long term planning, essential for precise landings.

The Markov property implies that the next state depends only on the current state and action, not on the sequence of states that preceded it. This allows the agent to make decisions based solely on its present state, simplifying the learning process.[12]

3.4 Exploration vs. Exploitation

A big challenge in reinforcement learning is the exploration exploitation trade off. In fact, the agent must find the right balance between :

- **Exploration** : Trying out new actions to discover potentially better strategies, even if they result in lower rewards in the short term.
- **Exploitation** : exploit known actions that have produced high rewards to maximize immediate gains.

In rocket landing applications, reinforcement learning agents navigate the exploration exploitation dilemma by balancing information gathering with reward maximization. Exploration examines diverse thruster configurations, landing angles, and descent velocities to understand environmental dynamics, while exploitation leverages established strategies like steady descent control and horizontal drift compensation. The ϵ ϵ-greedy algorithm facilitates this balance by selecting optimal actions with probability (1-ϵ) and random actions with probability ϵ, often implemented with decay schedules that progressively reduce ϵ to transition from exploration to exploitation. This calibrated approach prevents both premature convergence to suboptimal solutions and excessive resource expenditure on unproductive exploration paths.[12]

3.5 Role of Reward Functions in Successful Landings

In reinforcement learning, the reward function guides the agent's learning by assigning values to actions and outcomes. Designing an effective reward function is crucial for training an RL model capable of landing a rocket safely and efficiently. The Lunar Lander environment employs a carefully balanced system of positive rewards and penalties to shape the agent's behavior. These incentives work together to encourage successful landings while discouraging wasteful or dangerous actions.

Positive Rewards	**Negative Rewards (Penalties)**
Successfully landing within the designated landing pad	Crashing the lander
Smooth descents with minimal oscillations	Excessive fuel consumption
Maintaining proper orientation during descent	Drifting away from the target landing zone
Controlled velocity reduction before touchdown	Excessive tilting or rotation during descent

Table 1.1: Comparison of Positive and Negative Rewards in Lunar Lander

A poorly designed reward function can mislead the agent. For example, sparse rewards that only provide feedback for successful landings make learning difficult as the agent receives infrequent guidance. Similarly, conflicting rewards such as excessive penalties for fuel use without balanced rewards for controlled landings might cause the agent to avoid using thrusters altogether, resulting in crashes. Therefore, crafting a reward function that carefully balances safety, accuracy, and efficiency is key to successful training. Often, fine tuning these rewards through iterative experimentation helps the agent converge toward optimal landing policies.

4 Conclusion

In conclusion, the first chapter lays a strong foundation for our project, emphasizing the crucial role that artificial intelligence (AI) and machine learning (ML) will play. By highlighting these advanced,we showed that reinforcement learning is important for improving rocket landing. The fundamental concepts of rocket descent dynamics, control strategies,the integration of RL algorithms provide a comprehensive understanding of the challenges and solutions in this domain. This chapter introduces AI driven landing systems, helping make space missions more efficient and reliable.

Chapter 2

State of Art

1 Introduction

Landing a spaceship safely and on its own is a key challenge in today's space engineering, with big effects on exploring space and reusing expensive rockets. Since the 1950s, tossing rockets and space gear into Earth's oceans has shown why we need systems we can use again, like what SpaceX is doing grabbing their rockets back and launching them again in just hours. Old school landing tricks, based on set paths and human control, don't handle surprises well like wild weather or strange landing spots pushing us toward smart tech, especially reinforcement learning. RL is a learning method where the spaceship figures things out by trying stuff and learning from what works, which is perfect for the tough job of landing.

This chapter digs into the latest ways RL is used for these tasks, looking at how it grew from simple Q learning and Q tables where the rocket learns what moves are best step by step to fancier tools like Proximal Policy Optimization (PPO), great for smooth control jobs. Studies have moved from basic tests, like OpenAI's Lunar Lander game, to complex setups copying real stuff, such as SpaceX's Falcon 9 with its thrust tricks and gas jets. Plus, ideas from other areas like landing small drones using Deep Q Networks (DQNs) and smart memory tricks give us hints for dealing with messy situations, like a rocket dropping through stormy air or onto weird alien ground. All this shows how RL can make landings sharper, more flexible, and cheaper for space trips.

2 Reinforcement Learning Algorithms for Rocket Landing

Reinforcement learning (RL) algorithms are computational techniques that enable agents to learn optimal behaviors through interaction with an environment, rather than relying on explicit programming. These algorithms operate based on a reward-driven system, where agents receive feedback in the form of rewards or penalties, allowing them to refine their decision-making over time.

Reinforcement learning approaches can be broadly classified into value based methods, which focus on estimating the optimal value function (e.g., Q learning), and policy based methods, which directly optimize the policy that determines an agent's actions (e.g., Proximal Policy Optimization (PPO)). Value function methods, also known as critic- only methods, operate by first discovering the optimal value function such as a Q function through learning, and then deriving the optimal policy from it. Actor critic methods combine both value and policy based approaches to improve learning efficiency.

Model free RL algorithms, such as Deep Q Networks (DQN) and Advantage Actor Critic (A2C), learn policies without prior knowledge of the environment's dynamics, while model based RL methods leverage predictive models of the environment to enhance decision-making. Deep

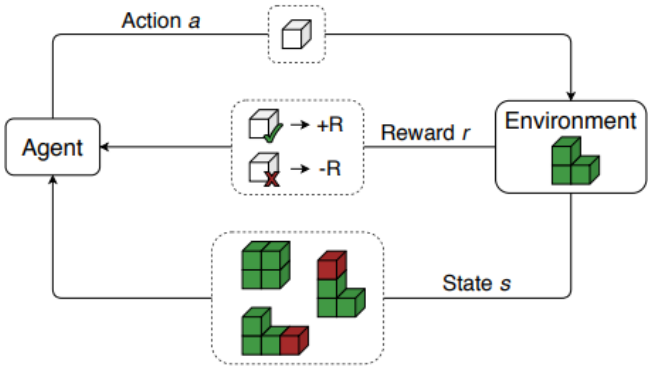

Figure 2.1: Basic Principle of Reinforcement Learning [1]

reinforcement learning (DRL) integrates artificial neural networks to approximate functions, enabling agents to handle complex, high dimensional state spaces. These algorithms are widely applied in robotics, autonomous control, and game playing AI.

2.1 Q Learning: A Model Free Reinforcement Learning Algorithm

Q learning is a **model free** reinforcement learning algorithm where agents learn optimal policies through environmental interaction without prior knowledge of dynamics. The algorithm centers on the **Q function**, which estimates expected rewards for state-action pairs, updated using the **Bellman equation**:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{2.1}$$

where:

- $Q(s,a)$: Value estimate for action a in state s
- r: Reward after action a
- γ: Discount factor for future rewards
- $\max_{a'} Q(s',a')$: Highest estimated value for next state
- α: Learning rate

As an **off policy** algorithm, Q learning learns the optimal policy independently of current action selection, often employing ε**-greedy** strategy to balance exploration and exploitation.

The algorithm initializes a **Q table** (Figure 2.2) as a matrix where rows represent states, columns represent actions, and cells contain expected rewards. With no prior knowledge, these values start arbitrarily. During learning, the agent:

1. Observes current state
2. Selects an action (using ϵ-greedy policy)
3. Receives reward feedback
4. Observes transition to next state
5. Updates Q values using the Bellman equation

Through continuous iteration, the Q values converge toward optimal values, enabling the agent to make increasingly better decisions based on accumulated experience.

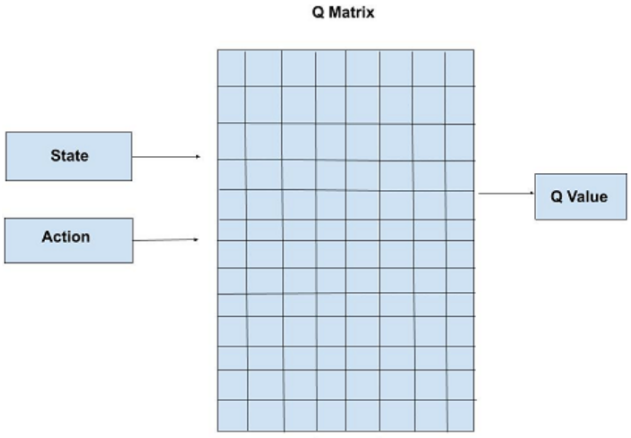

Figure 2.2: Structure of a Q-Table [10]

Recent work by **Wilson and Riccardi (2023)**.[18] applied Q learning to a 3-DOF spacecraft powered descent problem, demonstrating that discretizing action spaces and automated hyperparameter tuning (via TPE) could achieve a 98% success rate with shaped state representations. Their results highlighted the importance of state space design: agents using raw states (position, velocity, mass) failed to land safely due to reward hacking, while shaped states (velocity error, altitude) enabled efficient learning. Though their Q learning agent trained faster (4,000 episodes) than PPO (30,000 episodes), it sacrificed landing precision a trade off relevant to our study of DQN and PPO in Gym Lunar Lander.

2.2 Deep Q learning

To address the limitations of traditional Q Learning, researchers introduced **Deep Q Networks (DQNs)**, which combine Q Learning with deep neural networks. Instead of maintaining a table of Q values for each state action pair, DQNs approximate the Q value function using a weight parameterized neural network θ. The network takes a state as input and outputs Q values for all possible actions.

The Deep Q Networks (DQN) algorithm was introduced by Mnih et al. to address the limitations of traditional Q learning. This algorithm combines the Q learning framework with deep neural networks (DNNs). As is well known in artificial intelligence, DNNs are powerful non linear function approximators. Instead of maintaining a Q table, DNNs are used to approximate the Q function, allowing the agent to handle large and continuous state spaces.(see Figure 2.3).

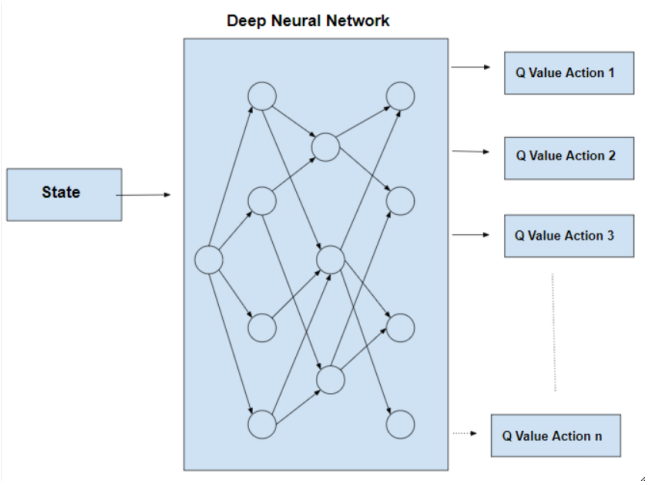

Figure 2.3: DQN architecture [10]

In a study by **Hicham Bouchana** et al.[14] , the authors propose an intelligent control method for launch vehicle landing using deep reinforcement learning. Their approach combines Long Short Term Memory (LSTM) networks with reinforcement learning to enhance the adaptability and precision of landing maneuvers. The method demonstrates superior performance in handling dynamic conditions and achieving high landing accuracy, further validating the potential of reinforcement learning in autonomous rocket landing systems. This work complements our research by highlighting the effectiveness of advanced neural architectures in improving real time decision-making for complex aerospace applications

2.3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm introduced by OpenAI in 2017, designed to improve the stability and efficiency of policy optimization methods. It is widely used in deep reinforcement learning, particularly for tasks involving high dimensional state and action spaces. PPO builds on previous policy gradient approaches and is an approximation of Trust Region Policy Optimization (TRPO), offering a more computationally efficient alternative while maintaining robust performance, finding a smarter, faster way to get things done without losing quality.

Proximal Policy Optimization (PPO) is an iterative reinforcement learning algorithm designed to optimize a policy while maintaining stability and efficiency. The agent begins by interacting with the environment, collecting trajectories.

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$$

which are then used to estimate the expected return and the advantage function $A(s, a)$. To update the policy, PPO introduces a probability ratio between the new and old policies, defined as

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

which measures how much the new policy deviates from the previous one. Instead of enforcing a strict constraint like in Trust Region Policy Optimization (TRPO), PPO employs a clipped surrogate objective function given by

$$J(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

where ϵ is a small hyperparameter that controls how much the policy can change in a single update. This clipping mechanism prevents excessively large updates, ensuring more stable training. The policy parameters θ are then updated using gradient ascent to maximize $J(\theta)$, while the value function $V_\pi(s)$ is simultaneously updated to improve state value estimations by minimizing the loss

$$L(\theta) = \mathbb{E}\left[(V_\pi(s) - R_t)^2\right]$$

where R_t is the empirical return. This optimization process is repeated iteratively across multiple training epochs, gradually refining the policy until it converges toward an optimal strategy.

Ashwinikumar Rathod [4] in his paper demonstrates the effectiveness of PPO in training an AI agent to land a reusable rocket in a custom Unity simulation, achieving a success rate of 98 percent. Their approach relies on addressing exploration difficulties, and their method uses carefully designed sparse rewards combined with curriculum learning.However, their environment assumes simplified dynamics (e.g., no atmospheric drag) and focuses solely on PPO.

3 Conclusion

Reinforcement learning presents considerable potential for autonomous rocket landing. Fundamental algorithms such as Q learning enable faster training but often compromise precision, whereas more advanced methods like Proximal Policy Optimization (PPO) and Deep Q Networks (DQN) achieve higher accuracy at the expense of training efficiency. The design of state representation is a critical factor using raw states can lead to unintended exploitation of the reward function, while engineered states contribute to more effective learning. Recent hybrid models that integrate Long Short Term Memory (LSTM) networks exhibit enhanced adaptability in dynamic environments. Future research should focus on reducing sample complexity without sacrificing performance and on ensuring effective transfer from simulation to real world applications. This analysis provides a foundation for the development of reliable and cost efficient spacecraft landing systems suitable for diverse operational contexts, by that means supporting progress in sustainable space exploration.

Chapter 3

Methodology

1 Introduction

This chapter investigates the use of reinforcement learning for autonomous lunar landing, aiming to enhance the precision and reliability of spacecraft control. It introduces the `LunarLander-v3` environment from the Gymnasium toolkit, which models lunar terrain and spacecraft physics under varied gravity conditions.

The chapter outlines visual data preprocessing techniques used to improve observation quality and support effective feature extraction. It then details the development of three reinforcement learning algorithms: Q Learning, Deep Q Networks (DQN), and Proximal Policy Optimization (PPO), including their architectures, training methods, and implementations.

Overall, the chapter provides a comprehensive overview of the learning strategies applied to lunar landing, setting the stage for performance evaluation in following sections.

2 Simulation Environment

Autonomous landing is simulated using the `LunarLander-v3` environment from the Gymnasium toolkit. This simplified scenario models a lander descending toward a designated landing pad between two flags, allowing interaction via discrete actions:

- **Action 0:** No thrust – free fall under gravity.
- **Action 1:** Fire left engine – rotate clockwise.
- **Action 2:** Fire main engine – apply upward thrust.
- **Action 3:** Fire right engine – rotate counterclockwise.

To increase realism, the agent receives `RGB` image frames instead of numerical state vectors, using `render_mode="rgb-array"`. This visual based observation mimics camera perception in real world autonomous systems.

3 Q Learning for Rocket Landing

Q learning is a reinforcement learning algorithm that enables an agent to make better decisions by exploring the environment and learning from the outcomes of its actions. The agent builds a Q table to estimate the expected future rewards for each action, starting from an empty table and gradually improving it through experience. It uses an epsilon-greedy policy to balance

exploration and exploitation, allowing it to try new actions while increasingly favoring the best known ones.

After each action, the agent receives a reward and updates its Q values using the learning rate and discount factor to refine future predictions. To manage the high dimensionality of visual inputs, image processing techniques like grayscale conversion, resizing to 84x84, and normalization are applied. The processed images are then converted into discrete features such as lander position, velocity, and angle, simplifying the environment into a manageable set of states. This abstraction allows the agent to focus on key patterns, with the learning process guided by the error between predicted and actual outcomes.

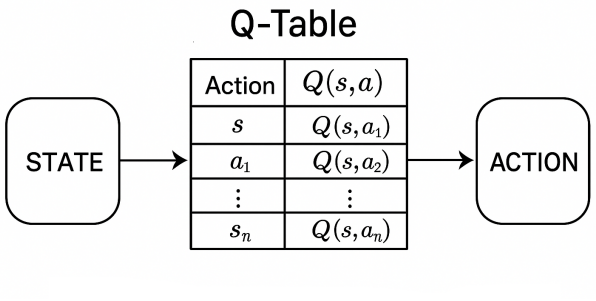

Figure 3.1: Q Learning architecture

.

3.1 Epsilon-Greedy Action Selection Strategy

Algorithm 1 Epsilon-Greedy Q Learning

function EPSILONGREEDYSELECT(Q, s, ϵ)
  Sample $p \sim \text{Uniform}(0, 1)$
  return random action if $p < \epsilon$ else $\arg\max_a Q(s, a)$
end function

function QLEARNINGSCHEDULED($\alpha, \gamma, \epsilon_{\text{start}}, \epsilon_{\text{min}}, \epsilon_{\text{decay}}, \text{maxEpisodes}$)
  Initialize $Q(s, a) \leftarrow 0$, $\epsilon \leftarrow \epsilon_{\text{start}}$
  for episode = 1 to maxEpisodes **do**
    Initialize state s
    while s not terminal **do**
      $a \leftarrow$ EPSILONGREEDYSELECT(Q, s, ϵ)
      Take a, observe r, s'
      $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
      $s \leftarrow s'$
    end while
    if episode > 1300 and episode mod $10 == 0$ **then**
      $\epsilon \leftarrow \max(\epsilon - \epsilon_{\text{decay}}, \epsilon_{\text{min}})$
    end if
  end for
  return Q
end function

The epsilon-greedy strategy manages the trade off between exploration and exploitation. Initially, the agent performs 1300 episodes of full exploration. Afterwards, epsilon starts at 1.0

and decays by 0.001 every 10 episodes until it reaches a floor of 0.01. This gradual reduction guides the agent from random action selection to informed decisions based on its Q table. A minimum exploration rate ensures continued strategy discovery throughout training.

3.2 Reward Mechanism

Algorithm 2 Q Table Learning with Shaped Reward

Initialize Q-table $Q(s, a)$, learning rate α, discount factor γ, and exploration rate ϵ
for *episode* $= 1$ to M **do**
    Reset environment and observe initial state s
    while s not terminal **do**
        Select action a using ϵ-greedy policy on Q
        Execute a, observe s', raw reward r, and environment info
        Compute shaped reward r_{shaped} based on distance, orientation, fuel use, time, velocities, and landing outcome
        Update $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r_{\text{shaped}} + \gamma \max_{a'} Q(s', a')]$
        $s \leftarrow s'$
    end while
    Update ϵ via decay schedule
end for

The reward function shapes agent behavior to promote efficient and stable landings. Rewards are given for reducing distance to the landing pad and maintaining ideal descent rates. Penalties discourage excessive tilting, high horizontal velocity, prolonged landing times, and heavy main engine use, while side engine usage is rewarded for fuel efficiency. A large bonus is granted for safe landings, and extreme negative rewards are capped to ensure training stability. Together, these factors guide the agent toward controlled, fuel efficient landings.

3.3 Agent Environment Interaction and Q Learning Updates

3.3.1 Agent Environment Interaction

Algorithm 3 Agent Environment Interaction

Initialize state s
for $t = 1, 2, \ldots, T$ **do**
    Choose action a from state s using policy derived from Q (e.g., ϵ-greedy)
    Take action a, observe reward r and next state s'
    Update $Q(s, a)$ using Q-Learning update rule
    $s \leftarrow s'$
end for

3.3.2 Update the Q Values

Algorithm 4 Q Learning Table Updates

Initialize $Q(s, a)$ arbitrarily for all state action pairs
Set learning rate $\alpha \in (0, 1]$ and discount factor $\gamma \in [0, 1]$
for each episode **do**
    Initialize state s
    for each step of episode until s is terminal **do**
        Choose action a from s using policy derived from Q (e.g., ϵ-greedy)
        Take action a, observe reward r and next state s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    end for
end for

The agent learns by interacting with the environment across multiple episodes, each representing a full cycle from start to terminal state. In each episode, the agent:

- **Observes the state:** Begins in an initial state defined by factors like position and velocity.
- **Selects an action:** Chooses an action using the ϵ-greedy policy from the Q table.
- **Acts and observes:** Executes the action, receiving a new state and a reward.
- **Updates Q table:** Adjusts the Q value based on the reward and the next state's estimated value.
- **Repeats:** Continues until the episode ends, then proceeds to the next episode to reinforce learning.

3.4 Validation of Learning Efficiency

Algorithm 5 Evaluation of Visual Based Agent

Input: Q function Q_θ, Environment $\mathcal{E}$, Evaluation episodes K
Output: Mean return $\bar{G}$, Mean length $\bar{L}$, Visualization $\mathcal{F}$
function EVALUATE_POLICY($Q_\theta, \mathcal{E}, K$)
    Initialize statistics $\mathcal{S}$ and final episode frames $\mathcal{F}$
    for *episode* $= 1$ to K **do**
        Reset environment, initialize return G and episode frames
        while episode not done **do**
            Store current frame
            Preprocess observation and extract features
            Select greedy action using Q_θ
            Step in environment, update return and observation
        end while
        Record episode return and length
        if *episode* $= K$ **then**
            Save frames for visualization
        end if
    end for
    Compute averages: $\bar{G}$ and $\bar{L}$
    SAVE_FRAMES_AS_GIF($\mathcal{F}$), PLOT_LEARNING_CURVE($\bar{G}$)
    return $\{\bar{G}, \bar{L}, \mathcal{F}\}$
end function
function PREPROCESS_OBSERVATION(*observation*)
    Convert to grayscale, resize, normalize
    return processed observation
end function
function EXTRACT_FEATURES(*processed_observation*)
    Return lower-dimensional feature vector
end function
function SAVE_FRAMES_AS_GIF($frames$)
    Convert frame sequence to animated GIF
end function

The `evaluate_policy` function measures the agent's true performance by running it without exploration, selecting actions solely based on the highest Q value. Each episode begins with an environment reset and observation, which is preprocessed and encoded into features. The agent acts based on these features until the episode ends.

Rewards are averaged across episodes to track progress and plotted over time. The final episode's frames are saved and turned into a GIF using `save_frames_as_gif`, providing a visual insight into the agent's behavior.

4 Deep Q Networks (DQN) for Rocket Landing

Deep Q Learning (DQL) extends traditional Q learning to handle complex environments with high dimensional state spaces by using a deep neural network instead of tabular representations, allowing it to learn directly from complex inputs like images or continuous data. In this Lunar Lander project, a Convolutional Neural Network (CNN) processes visual inputs to extract

important features, while a Double DQL implementation uses two neural networks to improve stability one network chooses actions and a second network estimates their value. A Replay Buffer stores past experiences, allowing random sampling to avoid repeating mistakes and break correlation between consecutive experiences. The agent's learning is guided by an ϵ-greedy policy that balances exploration and exploitation, helping it make effective decisions about safely landing in the Lunar Lander environment.

4.1 DQN Architecture and State Representation

DQN uses a convolutional neural network that processes four stacked grayscale frames (4 × 84 × 84) to capture temporal context. The network consists of three convolutional layers:

- First layer: 32 filters with an 8×8 kernel and stride 4, extracting low level features such as edges, followed by batch normalization.
- Second layer: 64 filters with a 4×4 kernel and stride 2, capturing more complex features, with batch normalization and ReLU activation.
- Third layer: 128 filters with a 3×3 kernel and stride 1, recognizing detailed patterns relevant to spacecraft landing.

An adaptive average pooling layer resizes the feature map to 6×6, resulting in a tensor of shape 128 × 6 × 6 that is flattened and passed to a fully connected layer with 512 units. The output layer predicts Q-values for four actions: firing left thruster, right thruster, main engine, or taking no action.

This architecture enables the agent to estimate the best action and predict future rewards from image data.

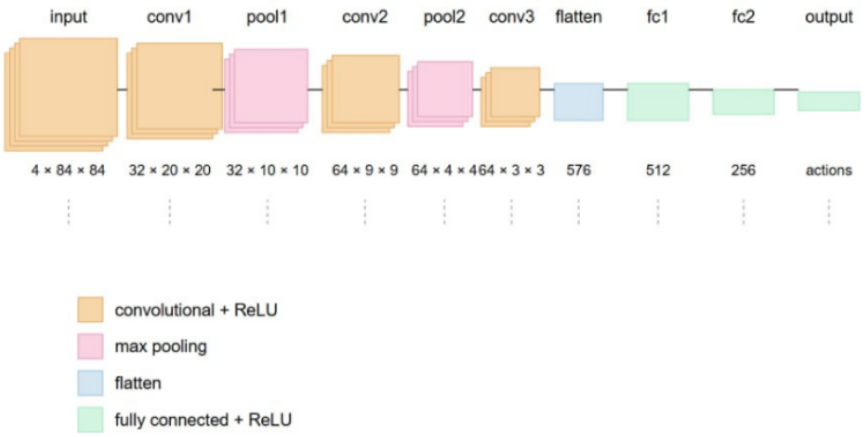

Figure 3.2: DQN architecture

4.2 Training Methodology and Learning Mechanisms

4.2.1 Initialization

Algorithm 6 DQL Agent Initialization

Initialize replay memory $\mathcal{D}$ with capacity N
Initialize action value function Q with random weights θ
Initialize target action value function $\hat{Q}$ with weights $\theta^- = \theta$
Initialize exploration rate ϵ to initial value ϵ_{start}
Initialize minimum exploration rate ϵ_{min}
Initialize exploration decay rate ϵ_{decay}
Initialize discount factor γ
Initialize learning rate α
Initialize batch size B
Initialize target network update frequency C
Initialize step counter $t = 0$
Initialize episode counter $episode = 0$
Initialize maximum number of episodes $max\_episodes$
Initialize maximum steps per episode max_steps
Initialize environment env

4.2.2 Main Training Loop

Algorithm 7 Training Loop

Initialize replay memory D and Q-network weights θ
Initialize target network weights $\theta^- = \theta$
for $episode = 1, 2, \ldots, M$ **do**
    Initialize state s_1
    for $t = 1, 2, \ldots, T$ **do**
        Select action a_t using ϵ-greedy policy based on $Q(s_t, a; \theta)$
        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 replay memory D
        Sample minibatch from D and compute target y_j
        Update Q-network by minimizing loss $(y_j - Q(s_j, a_j; \theta))^2$
        if update interval reached **then**
            Synchronize target network: $\theta^- = \theta$
        end if
    end for
    Decay exploration rate ϵ
end for

For episode = 1 to M (1500): Reset environment, get initial observation o1 ,Preprocess o1 to get first frame f1 ,Reset frame stack S to contain 4 copies of f1 ,Initialize episode reward R = 0.

4.2.3 State Preprocessing Function (for each Frame)

Algorithm 8 State Processing Function

function StackFrames($stacked\_frames, frame, is\_new\_episode$)
  $gray\_frame \leftarrow$ ConvertToGrayscale($frame$)
  $resized\_frame \leftarrow$ ResizeTo84x84($gray\_frame$)
  $normalized\_frame \leftarrow resized\_frame/255.0$
  if $is\_new\_episode$ **then**
    $stacked\_frames \leftarrow$ CreateDeque($max\_length = 4$)
    for $i = 1$ to 4 **do**
      $stacked\_frames$.append($normalized\_frame$)
    end for
  else
    $stacked\_frames$.append($normalized\_frame$) ▷ Oldest frame automatically removed
  end if
  $stacked\_state \leftarrow$ Stack($stacked\_frames, axis = 0$) ▷ Shape: $(4, 84, 84)$
  return $stacked\_state, stacked\_frames$
end function

This function preprocesses a visual frame (grayscale, resize, normalize) and stacks it with the previous frames to form a 4 frame state tensor. If it's a new episode, it resets the stack using the current frame.

4.2.4 Frame Stacking

Algorithm 9 Frame Stacking

Input: Frame stack S, current frame f, episode flag is_new_episode
Output: Stacked state tensor stacked_state updated frame stack S State Preprocess the current frame f
if is_new_episode is **True then** State Initialize the frame stack with four copies of the processed frame
else
  Append the processed frame to the existing stack, removing the oldest frame
end if
Stack the frames along a new dimension to form the state tensor
return the stacked state tensor and updated frame stack

This function preprocesses a frame and stacks it with the last three to form a 4 channel state used for deep RL. On new episodes, the stack is reset with four copies of the first processed frame.

4.3 Implementation Details and Performance Analysis

4.3.1 Loss Function

This function calculates the loss for Double DQN by comparing predicted Q values from the main network to target Q values computed using the target network. The result is the mean squared error used to train the main network.

Algorithm 10 DQN Loss

Input: $main\_network$, $target\_network$, batch (s, a, r, s', d), discount γ
Output: $loss$
$a' \leftarrow \arg\max(main\_network(s'))$
$q' \leftarrow target\_network(s').\text{gather}(1, a'.unsqueeze(1)).squeeze(1)$
$target \leftarrow r + \gamma \cdot q' \cdot (1 - d)$
$q \leftarrow main\_network(s).\text{gather}(1, a.unsqueeze(1)).squeeze(1)$
$loss \leftarrow \text{mean}((q - target)^2)$
return $loss$

4.3.2 Evaluation Performance

Algorithm 11 Evaluate DQL Agent Perfemance

for each episode $e = 1$ to E **do**
    Initialize state s
    $done \leftarrow False$, $reward \leftarrow 0$
    while not $done$ **do**
        $a \leftarrow \arg\max_a Q(s, a; \theta)$
        Take a, observe s', r, $done$
        $reward \mathrel{+}= r$, $s \leftarrow s'$
    end while
    Log $reward$
end for
Compute and report average reward

This function evaluates the agent's learning performance by computing key metrics like average reward, convergence speed, and reward stability. It also visualizes reward trends and exploration decay to assess training quality.

5 Proximal Policy Optimization (PPO) for Rocket Landing

Proximal Policy Optimization (PPO) is a reinforcement learning method that enables stable and efficient training, especially for challenging tasks like the Lunar Lander.

In this project, PPO trains an agent using a neural network that turns environment states into action probabilities, letting the agent adapt its landing approach on the fly.

It keeps learning steady by clipping policy updates, preventing sudden, big changes that could disrupt training.

The agent uses an Advantage Function to focus on actions that lead to better results than expected, guiding smarter decisions.

By gathering experiences in batches and updating the policy regularly, PPO helps the agent progressively improve its landing skills through continuous practice and feedback.

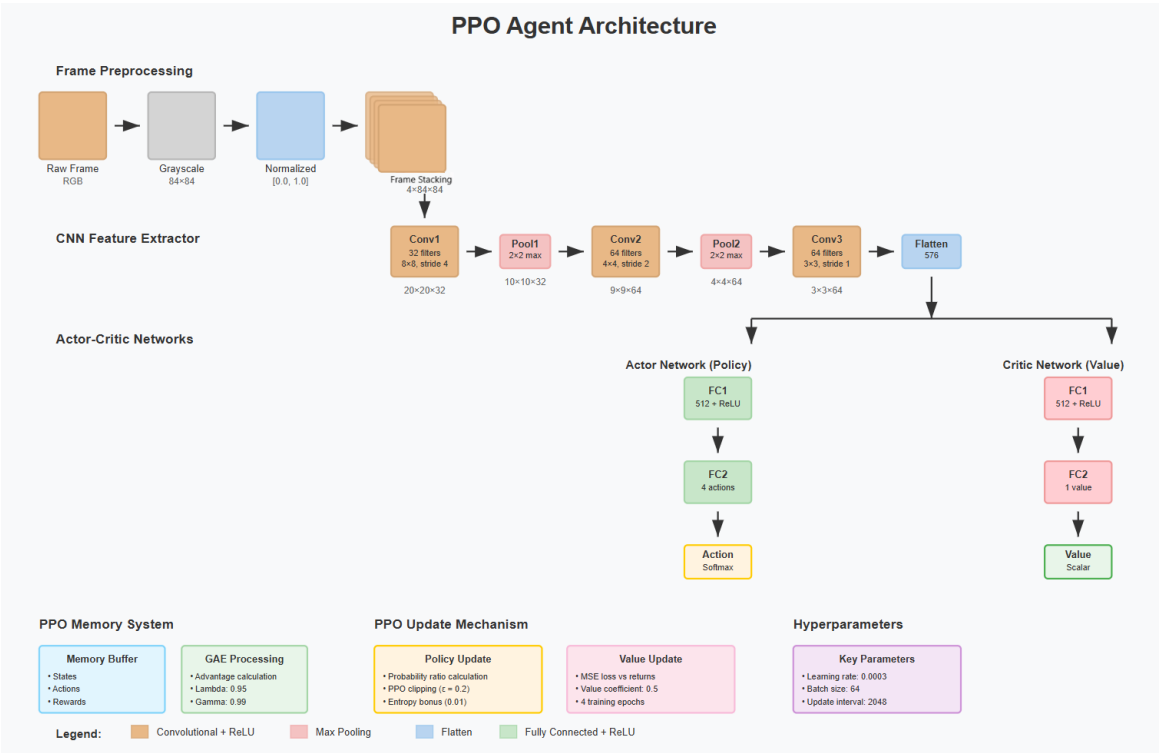

Figure 3.3: PPO architecture

5.1 Clipped Surrogate Objective

Algorithm 12 Clipped Objective

function COMPUTE_CLIPPED_OBJECTIVE($states, actions, old\_log\_probs, advantages$)
    Compute new log-probs and value estimates
    $ratio \leftarrow \exp(\text{new\_log\_probs} - \text{old\_log\_probs})$
    $surr1 \leftarrow ratio \cdot advantages$
    $surr2 \leftarrow \text{clip}(ratio, 1-\epsilon, 1+\epsilon) \cdot advantages$
    $loss \leftarrow -\text{mean}(\min(surr1, surr2))$
    return $loss$, value estimates, entropy
end function

This function computes the PPO loss by comparing new and old action probabilities, using a clipped ratio to ensure stable policy updates. It returns the actor loss along with value estimates and entropy to guide learning.

5.2 Actor-Critic Architecture

5.2.1 Actor-Critic Network Forward Pass

Algorithm 13 Actor Critic Forward

function ACTOR_CRITIC_FORWARD($state\_batch$)
    $features \leftarrow$ flatten(cnn($state\_batch$))
    $action\_logits \leftarrow$ linear_actor($features$)
    $state\_value \leftarrow$ linear_critic($features$)
    return $action\_logits, state\_value$
end function

5.2.2 Action Selection Using Actor-Critic Policy

Algorithm 14 Select Action

function SELECT_ACTION($state$)
    Get $action\_logits, value \leftarrow actor\_critic\_forward(state)$
    $probs \leftarrow softmax(action\_logits)$
    $action \leftarrow$ Categorical($probs$).$sample()$
    $log\_prob \leftarrow \log(probs[action])$
    return $action, log\_prob, value$
end function

The `actor_critic_forward` function processes input through shared CNN features to generate both action logits (actor) and state values (critic). The `select_action` function samples an action from the policy distribution and returns the action, its log-probability, and the state value.

5.3 Generalized Advantage Estimation

Algorithm 15 Calculate GAE

function CALCULATE_GAE_ADVANTAGES($rewards, values, dones, \gamma, \lambda$)
    Initialize $advantages \leftarrow zeros(T)$, $last \leftarrow 0$
    for $t = T-1$ to 0 **do**
        $next \leftarrow 0$ if $dones[t]$ else $values[t+1]$
        $mask \leftarrow 0$ if $dones[t]$ else 1
        $delta \leftarrow rewards[t] + \gamma \cdot next \cdot mask - values[t]$
        $advantages[t] \leftarrow delta + \gamma \cdot \lambda \cdot mask \cdot last$
        $last \leftarrow advantages[t]$
    end for
    $returns \leftarrow advantages + values$
    $advantages \leftarrow \dfrac{advantages - mean}{std + 1e{-}8}$
    return $returns, advantages$
end function

This function computes the Generalized Advantage Estimation (GAE) by iterating backward through the episode to calculate temporally smoothed advantages. It returns both the normalized advantages for the actor and the value targets (returns) for the critic.

6 Conclusion

This chapter introduced the implementation of three reinforcement learning methods Q Learning, Deep Q Networks (DQN), and Proximal Policy Optimization (PPO) for autonomous lunar landing. Each method was adapted to address specific challenges in control, observation processing, and training stability within a simulated environment.

These developments establish the technical foundation for the experimental phase. In the next chapter, we will present our experiments, results, and analysis, providing a detailed evaluation of the applied models.

Chapter 4

Implementation and Results

1 Introduction

This chapter outlines the implementation of the proposed reinforcement learning approaches and presents the results of the conducted experiments. We begin with Q Learning under three different scenarios, incorporating techniques such as feature extraction and reward shaping. This is followed by experiments using Deep Q Learning and Proximal Policy Optimization (PPO). Each method is evaluated based on performance, and the results are analyzed to assess their effectiveness in solving the rocket landing task using image-based state representations.

2 Experiments Conducted and Results Obtained

As a reminder, our experiments were tested at various algorithms, and in each scenario, we kept the best accuracy along with its corresponding resolution. Reinforcement learning (RL) algorithms differ significantly in their architecture, learning capabilities, and suitability for complex environments. In this study, we compare three prominent RL approaches Q learning, Deep Q Learning (DQN), and Proximal Policy Optimization (PPO) in the context of the LunarLander-v3 environment, utilizing image based state representations.

Q learning, a classical value based method, employs a tabular representation (Q table) to estimate the optimal action value function. While effective in discrete, low dimensional environments, its performance deteriorates rapidly in high dimensional or continuous state spaces. In the LunarLander-v3 environment, where state observations are represented as images, Q table methods become impractical due to the curse of dimensionality and the inability to generalize across similar states. As a result, agents trained with Q learning fail to develop stable landing policies and exhibit poor convergence.

Deep Q Learning (DQN) extends Q learning by integrating deep neural networks to approximate the Q function, enabling learning from raw visual input. This function approximation significantly enhances the agent's ability to generalize across states and improves learning efficiency in high dimensional environments. In our experiments, DQN demonstrated steady performance gains over time, with the agent achieving increasingly higher rewards, indicating the feasibility of image based learning in complex control tasks.

PPO outperformed Q learning and DQN with better stability and results. Using a clipped objective function for controlled policy updates, the PPO agent showed consistent performance with minimal late stage variance, indicating task mastery. The study confirms Q table methods struggle with complex environments, while PPO excels in visual based reinforcement learning applications like LunarLander-v3.

Aspect	**PPO**	**Q-Table**	**Deep Q-Learning**
Algorithm Type	Policy-based	Value-based	Value-based (neural network)
Episodes	∼1500	∼10,000	∼1300
Initial Rewards	Poor (∼ -200 to -600)	Poor (∼ -200 to -600)	Moderate (∼ -50 to -100)
Final Rewards	Positive (0 to 200)	Declining (∼ -200 to -600)	Stable (∼ -50 to -100)
Learning Progression	Clear improvement over time	No improvement/deterioration	Relatively stable
Reward Volatility	High (large oscillations)	Very high	Moderate
Maximum Reward	∼250 (around episode 900)	∼50 (rare peaks)	∼0 (occasional peaks)
Overall Trend	Strong positive learning curve	Negative trend	Flat, consistent performance
Convergence	Converges to positive rewards	Fails to converge	Converges to suboptimal solution
Stability	Initially unstable, stabilizes with training	Consistently unstable	Relatively stable throughout
Sample Efficiency	Good (learns in ∼1000 episodes)	Poor (no improvement in 10,000 episodes)	Moderate (quickly reaches stable performance)

Table 4.1: Detailed Comparison of PPO, Q-Table, and Deep Q-Learning Algorithms

2.1 Experiment 1: Training a Model with Q-Learning

In this experiment, we develop the Q learning model according to three different scenarios.

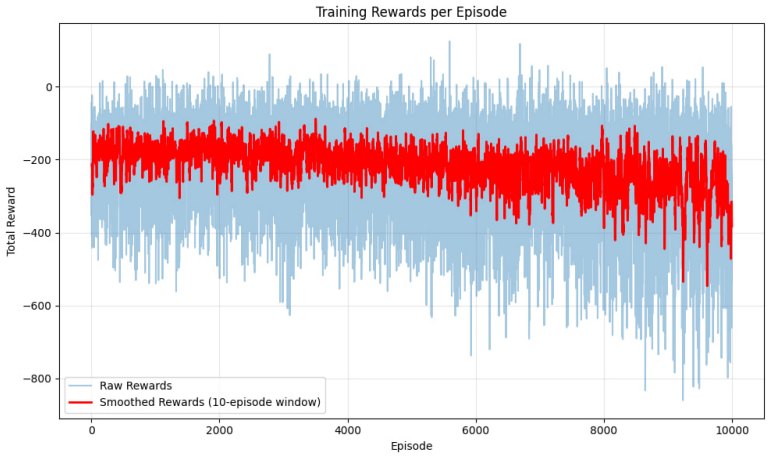

Figure 4.1: Q learning Graph

2.1.1 Scenario 1 :Q Learning with Image Based State Representation

In this scenario, A Q learning agent learns to control the `LunarLander-v3` using raw visual input. Frames are grayscaled, resized to 16×16, and binarized, then flattened into tuples used as keys in a dictionary based Q table. The agent uses an ϵ-greedy policy and updates Q values via the Bellman equation. While this setup demonstrates that image based RL is possible without feature extraction, performance is limited due to the high dimensional state space and lack of generalization.

2.1.2 Scenario 2 :Q Learning with Image Based State Representation (Feature Extraction)

In Scenario 3, the Q learning agent uses enhanced image based state representation by extracting key features from grayscale, resized images using edge detection. This highlights important elements like the lander and pad contours, allowing the agent to focus on relevant environmental details. The extracted features are converted into discrete grids, reducing state complexity. Additionally, a refined, scaled reward shaping system is implemented to encourage horizontal and vertical alignment, stability, and controlled descent, while applying proportional penalties for poor performance to promote more precise learning.

2.2 Experiment 2: Training a Model with Deep Q Learning

In this experiment, Deep Q Learning (DQL) is applied to the Lunar Lander environment using convolutional neural networks to process raw visual input. RGB frames are preprocessed by converting to grayscale, resizing to 84×84 pixels, and normalizing values. Four consecutive frames are stacked to provide temporal context for motion inference. The CNN architecture includes three convolutional layers (with increasing filters and decreasing kernel sizes), batch normalization, adaptive pooling, and fully connected layers to output Q values.

Reward clipping is used to stabilize learning, and an ϵ-greedy strategy manages exploration, with ϵ decaying from 1.0 to 0.01. Double DQN is employed with separate main and target networks, updated every 5 episodes, to reduce Q value overestimation and ensure training stability over 1500 episodes.

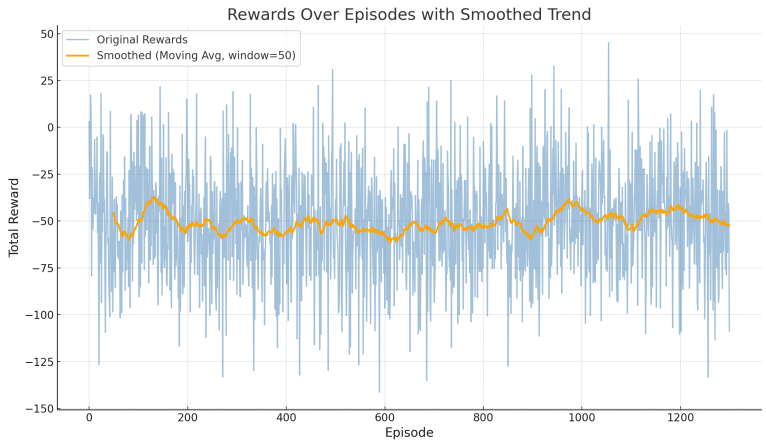

Figure 4.2: DQL Graph

2.3 Experiment 3: Applying Proximal Policy Optimization (PPO)

In this experiment, we applied Proximal Policy Optimization (PPO) with a convolutional network to learn landing directly from raw visuals. RGB frames are converted to 84×84 grayscale, normalized, and four frame stacked to capture motion. A shared CNN backbone (three conv layers: 32–64–64 filters) feeds into separate 512 unit actor and critic heads: the actor produces action logits for a categorical policy, and the critic outputs state values. PPO's clipped surrogate objective constrains policy updates, while GAE ($\lambda = 0.95$) delivers smooth advantage estimates. The combined loss (actor loss + 0.5 × critic loss − 0.01 × entropy) is optimized over 1500 episodes, with updates every 2048 steps, 4 epochs per update (batch size 64), and gradient clipping (max norm 0.5) to ensure stable training.

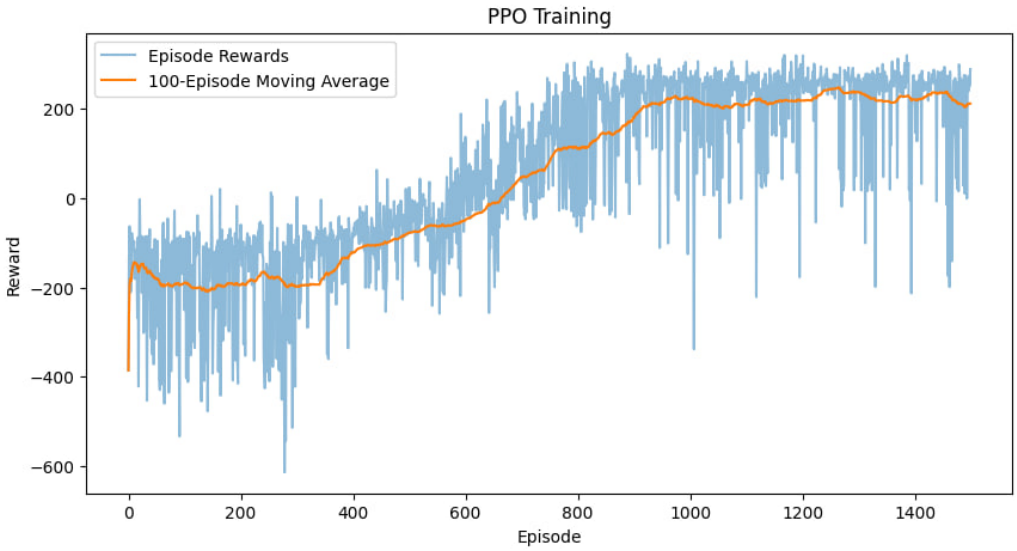

Figure 4.3: PPO Graph

3 Conclusion

In this chapter, we implemented and evaluated three reinforcement learning approaches Q Learning, Deep Q Learning, and Proximal Policy Optimization (PPO) within the context of a rocket landing task using image based state representations. Through a series of structured experiments, we explored the impact of feature extraction and reward shaping on learning

performance. The results demonstrated the relative strengths and limitations of each method, with PPO showing superior stability and convergence compared to the other approaches. These findings confirm the feasibility and effectiveness of using reinforcement learning techniques for complex control tasks and lay the groundwork for potential future enhancements.

General Conclusion

This work has demonstrated the significant potential of reinforcement learning techniques to address the complex challenges of spacecraft landing. Through our comprehensive investigation comparing Q Learning, Deep Q Networks (DQN), and Proximal Policy Optimization (PPO) algorithms within the OpenAI Gym Lunar Lander environment, we have established that machine learning approaches can effectively navigate the intricate dynamics of landing scenarios while adapting to unpredictable conditions.

Our experimental results reveal that reinforcement learning agents can successfully master the delicate balance between exploration and exploitation, eventually developing robust landing strategies that overcome the challenges of low gravity environments. The progression from traditional Q Learning to more sophisticated approaches that include image based state representation and feature extraction has led to increasingly reliable landing performance, with PPO demonstrating particular promise for deployment in real world space missions.

This research contributes to the growing evidence that smart self landing spacecraft powered by reinforcement learning represent the next frontier in space exploration technology. By eliminating the need for human intervention and enabling spacecraft to adapt to varied landing conditions in real time, these systems hold transformative potential not only for future lunar and planetary missions but also for terrestrial applications requiring precise control under uncertain conditions.

As space agencies and private companies expand their ambitions for interplanetary exploration, the integration of reinforcement learning into landing systems will likely become standard practice, opening new possibilities for missions to previously inaccessible or hazardous landing sites. Future work should focus on transferring these techniques to higher fidelity simulation environments and eventually to hardware implementations, bridging the reality gap between simulation and actual spacecraft deployment.

The journey from primitive airbag landing systems to fully autonomous, AI powered landing technology represents one of the most significant advancements in aerospace engineering. Through continued refinement of reinforcement learning approaches, we are at the edge of an evolution where spacecraft can land methodically and effortlessly on any celestial body, irrespective of the gravitational, atmospheric, or terrain challenges, furthering the scope of human existence throughout the solar system.

Bibliography

[1] Mehdi Benallegue, Yuki Kojio, and Atsuo Takanishi. A survey of reinforcement learning algorithms for dynamically stable humanoid robot locomotion. *Robotics and Autonomous Systems*, 139:103713, 2021. Accessed: May 18, 2025.

[2] Joey De Vries. Learn opengl. *Licensed under CC BY*, 4, 2015.

[3] Harrison. Pythonprogramming.net. `https://pythonprogramming.net/`. Consulté le 19 juin 2023.

[4] Pranav K. M., Ayushi Singh, and Sushil V. Chavan. Autonomous rocket landing using reinforcement learning. *International Journal of Creative Research Thoughts (IJCRT)*, 11(5):e503–e510, 2023. Accessed: May 18, 2025.

[5] Macromoltek. Ai & neural networks, 2020. Accessed: 2025-05-18.

[6] NASA. What is microgravity? `https://www.nasa.gov/centers-and-facilities/glenn/what-is-microgravity/`, 2024. Accessed: 2025-05-20.

[7] OpenAI. Openai baselines: Ppo, 2024. Accessed: March 14, 2025.

[8] Zhengkai Qi, Junkang Zhang, Faming Fang, Tingting Wang, and Guixu Zhang. Ugnet: Uncertainty aware geometry enhanced networks for stereo matching. *Pattern Recognition*, page 110410, 2024.

[9] Eli Stevens and Luca Antiga. *Deep Learning with PyTorch: Essential Excerpts*. Manning Publications, 2020. Available via Manning Publications.

[10] Suraj Subramanian. Diving into deep reinforcement learning with deep q-learning. `https://towardsdatascience.com/diving-into-deep-reinforcement-learning-with-deep-q-learning-376e588bb803/`, 2019. Accessed: May 18, 2025.

[11] Ivan E Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764, 1968.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetts, London, England, 2nd edition, 2018.

[14] Achraf Toumi and Elias Rius. AI-Guided Rocket Landing: Navigating Precision Descent Strategies. `https://www.researchgate.net/publication/383557912`, 2024. Accessed: May 18, 2025.

[15] Wikipedia contributors. Gimbaled thrust diagram — wikipedia, the free encyclopedia. https://fr.m.wikipedia.org/wiki/Fichier:Gimbaled_thrust_diagram.gif, 2023. Accessed: May 18, 2025.

[16] Wikipedia contributors. Reaction control system — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Reaction_control_system, 2024. Accessed: 2025-05-20.

[17] Wikipedia contributors. Thrust vectoring — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Thrust_vectoring, 2024. Accessed: 2025-05-20.

[18] Callum Wilson and Annalisa Riccardi. Improving the efficiency of reinforcement learning for a spacecraft powered descent with q-learning. *Optimization and Engineering*, 24:223–255, 2023. Accessed: May 18, 2025.