Autonomous Lunar Landing Using Model-Based and Deep Reinforcement Learning

Aaditya Raj Anand **Ankit Raj** **Shreya Jhawar**

Vineesh K. Vinod
Department of Avionics
Indian Institute of Space Science and Technology
Thiruvananthapuram, Kerala, India

Abstract

Autonomous lunar landing is a safety-critical control problem with nonlinear dynamics, discontinuous contact events, and stringent performance requirements, all of which make it challenging for classical methodologies and highly relevant to planetary exploration missions. Reinforcement learning provides a data-driven approach for learning control policies by interacting with simulated environments, reducing reliance on handcrafted controllers and the need for exact system modeling. This work considers the LunarLander-v3 task in the OpenAI Gymnasium environment and adopts a two-tier approach that effectively merges classical model-based reinforcement learning with deep reinforcement learning methods. First, it recasts the problem as a discrete-state, discrete-action Markov decision process and solves it using value iteration to provide insight into the state representation, reward design, and the structure of the optimal policy. Then, a two-dimensional physics-based lunar landing environment with continuous dynamics is used to train a Double Deep Q-Network agent for autonomous landing. Results show that, while the simplified MDP provides conceptual insight, the Double DQN agent has better learning stability and faster convergence in the full continuous environment, while also exposing practical challenges in reward shaping, training stability, and computational efficiency.

1 Introduction

Autonomous landing is a critical phase of spacecraft missions, where failures may result in the irretrievable loss of the mission. This problem puts forward challenges such as nonlinear dynamics, gravitational forces, limited actuation, and uncertainty in ground conditions. Classical approaches to control problems usually rely on accurate models of the system and require extensive manual tuning, which is particularly cumbersome to realize in highly nonlinear or uncertain settings. At the same time, reinforcement learning allows an agent to learn a control policy directly from interactions with the environment.

Deep reinforcement learning has recently enabled the solving of challenging continuous control problems characterized by complex state spaces and high-dimensional observations. The Lunar Lander is among the most popular benchmark problems in the reinforcement learning literature. The challenge is well known for requiring autonomous control under highly nonlinear and dynamic conditions.

Earlier studies rely on either classical control or heuristic methods, while the most recent work demonstrates the efficiency of both value-based and policy-gradient reinforcement learning approaches

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

such as Deep Q-Networks and Proximal Policy Optimization. Despite this, many works proceed straightforwardly to deep reinforcement learning methods without explicit connections to more parsimonious model-based formulations.

Herein, we structure our approach by first analyzing a simplified Markov Decision Process version of the problem through value iteration, then extend the solution to the complete continuous environment through deep reinforcement learning. This practice allows us to bridge between basic theory and practical deep reinforcement learning applications.

2 Model and Problem Statement

2.1 Environment Description

The lunar landing task is defined in a two-dimensional Cartesian plane with uniform gravitational acceleration. The lander is modeled as a rigid body, consisting of a central chassis with articulated landing legs at each end. The environment consists of rough terrain and a flat landing pad. The goal is to get the lander to touch down on the pad with low velocity, minor tilt, and both legs touching the ground.

Two environments are considered:

- A custom simplified MDP for analytical study will be designed for (Task-1).
- A comprehensive physics-based environment given in OpenAI Gymnasium (Task 2).

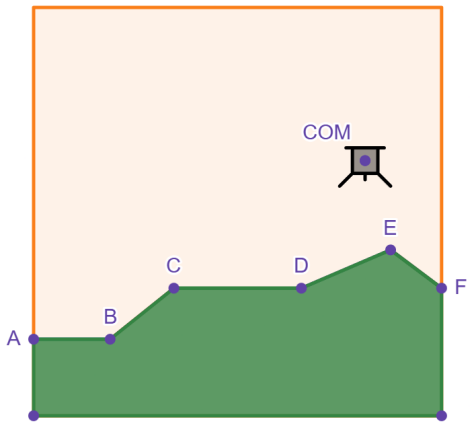

Figure 1: Two-dimensional lunar lander environment showing terrain profile, landing pad, lander body, and articulated landing legs.

2.2 Simplified MDP Model

2.2.1 State Space

We define the simplified MDP state as

$$s = (x_d, y_d, \theta_d, L, R), \tag{1}$$

where x_d and y_d represent discretized horizontal and vertical position bins, respectively, θ_d denotes the discretized orientation angle, and $L, R \in \{0, 1\}$ indicate contact states of the left and right landing legs.

Velocities are intentionally omitted to reduce the dimensionality of the state space, thereby making value iteration computationally feasible.

2.2.2 Action Space

The action space is discrete and defined as

$$\mathcal{A} = \{\text{noop},\ \text{left engine},\ \text{main engine},\ \text{right engine}\} \tag{2}$$

These correspond respectively to: no action, firing the left side thruster, the main engine, or the right side thruster.

2.2.3 Transition Dynamics

State transitions are solved using the semi-implicit Euler method due to its long-term stability, which is useful for reinforcement learning experiments involving many rollouts. Given a current state and a selected action, the subsequent state is uniquely determined. Small disturbances may optionally be added to model uncertainty in state transitions.

2.2.4 Reward Function

The reward function is designed to guide the agent toward a safe landing and consists of the following components:

- Step-wise shaping rewards that encourage proximity to the landing pad
- Penalties for excessive tilt
- Positive rewards for landing leg contact
- A terminal reward of $+100$ for a safe landing
- A terminal penalty of -100 for a crash

2.2.5 Termination Conditions

An episode terminates under any of the following conditions:

- Both landing legs make stable contact with the landing surface
- The lander crashes or exits the simulation boundaries

2.2.6 Solution Method

Value iteration is employed to compute the optimal policy over the discrete MDP. This analytical solution provides insight into the reward structure and the resulting policy behavior in the simplified environment.

2.3 Gymnasium LunarLander Environment (Task-2)

2.3.1 State Space

The *Gymnasium LunarLander-v3* environment provides an 8-dimensional continuous state vector defined as

$$s = (x, y, v_x, v_y, \theta, \omega, c_L, c_R), \tag{3}$$

where x and y denote the horizontal and vertical positions of the lander, v_x and v_y represent the corresponding linear velocities, θ is the orientation angle, ω is the angular velocity, and $c_L, c_R \in \{0, 1\}$ indicate contact of the left and right landing legs with the surface.

2.3.2 Action Space

The action space is discrete and consists of four possible actions corresponding to engine activations:

- Do nothing
- Fire left orientation engine
- Fire main engine
- Fire right orientation engine

2.3.3 Reward Function

The environment employs a shaped reward function that incorporates the following components:

- Penalties for fuel consumption
- Penalties for excessive tilt and high velocities
- Positive rewards for landing leg contact
- A large terminal reward for a successful landing and a terminal penalty for a crash

2.3.4 Termination Conditions

An episode terminates upon successful landing, a crash, or if the lander exits the environment boundaries.

3 Reinforcement Learning Agent

3.1 Algorithm Description

A Double Deep Q-Network (Double DQN) is employed to mitigate the overestimation bias observed in standard DQN. The approach maintains two separate neural networks:

- An online network used for action selection
- A target network used for stable value estimation

Action selection is performed using the online network, while the corresponding action values are evaluated using the target network. This decoupling reduces overoptimistic value estimates and improves training stability.

3.2 Implementation Details

A fully connected neural network is used to approximate the action–value function. Experience replay is employed to decorrelate training samples and improve data efficiency. During training, an ϵ-greedy exploration strategy with a decay schedule is applied to balance exploration and exploitation.

The interaction with the environment, including its physics and state transitions, is handled by the Gymnasium framework, while the learning algorithm and training pipeline are implemented by the project team. The complete set of hyperparameters used in training the Double DQN agent is listed in Table 1.

4 Extensions

We have observed that while using Double DQN network, the problems that we face are slow convergence and it may have poor performance in noisy environments. So to mitigate that several extensions to the current methodology are possible, such as dueling architecture, prioritized replay and multi-step returns. For major gains we could look into other RL agents such as Rainbow DQN, PPO, A2C, or SAC which also yield perhaps more natural ways of handling continuous action spaces. Efficient implementations of these algorithms can be found in libraries like Stable-Baselines3, CleanRL, and Tianshou, all of which can be interfaced with the current environment.

Beyond RL, other non-RL approaches include evolutionary algorithms or model predictive control, which might be considered for comparison. Further extensions to the environment, for instance, by including stochastic disturbances or more realistic terrain models, can improve the problem complexity. Any assumptions concerning additional state information would need careful justification for fairness.

Table 1: Hyperparameters used for Double DQN training

Parameter	Value
State dimension	8 (position, velocity, angle, leg contacts)
Action space	4 discrete actions
Hidden layer 1	Fully connected, 128 neurons, ReLU
Hidden layer 2	Fully connected, 128 neurons, ReLU
Output layer	Fully connected, 4 neurons (Q-values)
Optimizer	Adam
Learning rate	3×10^{-4}
Loss function	Huber loss (Smooth L1)
Replay buffer size	100,000 frames
Batch size	64
Discount factor (γ)	0.99
Exploration strategy	ϵ-greedy
Initial ϵ	1.0
Final ϵ	0.1
ϵ decay	Linear decay over 150,000 frames
Target network update	Soft update
Target update rate (τ)	0.005
Training frequency	Every 4 environment steps

5 Performance Analysis and Comparison

While the value iteration-based policy performs well in the simplified MDP, it does not generalize to the full continuous environment. By contrast, the Double DQN agent successfully learns stable landing trajectories in the Gymnasium environment. Compared to standard DQN, Double DQN exhibits smoother learning curves, lower variance in episode rewards, and faster convergence.

Due to the time constraints we haven't run the DQN agent in the environment. First we have trained the Double DQN model till the average reward over 100 episodes is more than 250, the threshold is set higher to improve the system stability. Then we have evaluated the trained model for 100 episodes. Both are plotted and we can observe that the model is trained after 1400 episodes, and the system is stable.

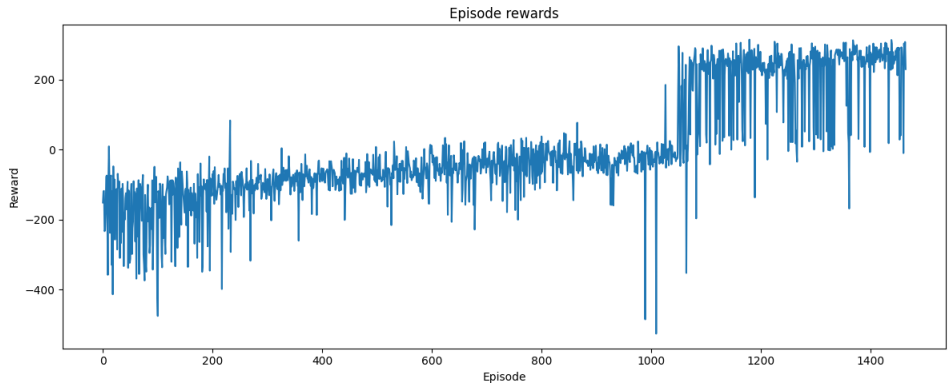

Figure 2: Training performance of the Double DQN agent showing episodic reward as a function of training episodes.

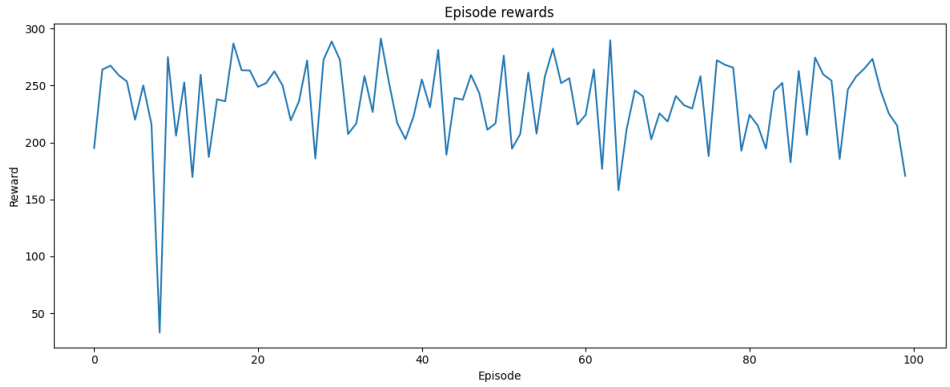

Figure 3: Evaluation performance of the trained Double DQN agent with exploration disabled, showing episodic reward over evaluation episodes.

6 Challenges

Key challenges include sensitivity to reward shaping, instability in early training, and high computation requirements. Much attention is paid to the work of hyperparameter tuning for stable learning mainly the learning rate and the epsilon decay rate . Future work may involve improving the system model, using policy-gradient methods, curriculum learning, and sim-to-real transfer for practical space applications.

References

[1] Farama Foundation. Gymnasium: A standard interface for reinforcement learning environments. `https://gymnasium.farama.org/`, 2023.

[2] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[4] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.