
Lunar Lander

Team: Core Chruncher
Aaditya Raj Anand (SC22B082)
Shreya Jhawar (SC22B115)
Ankit Raj (SC22B087)
Vineesh K Vinod (SC22B153)



INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
THIRUVANANTHAPURAM, KERALA, INDIA 695547

December 11, 2025



- 1 Introduction
- 2 Model and Problem Statement
- 3 Environment Implementation
- 4 Reinforcement Learning Agent
- 5 Extensions
- 6 Performance Analysis



Motivation & Problem Statement

- Autonomous lunar landing is a safety-critical task in planetary exploration missions.
- Real-world experimentation is costly, risky, and cannot support rapid design iterations.
- Reinforcement Learning enables data-driven optimal control for nonlinear, discontinuous systems.
- A physics-based simulator is essential for training, testing, and validating RL agents before deployment.

Project Goal

Develop a complete Python-based 2D lunar lander simulator with accurate rigid-body dynamics, terrain interaction, and reward shaping and train a Deep RL agent (DQN) to autonomously perform safe landings.



Problem Summary

Challenge: Develop an autonomous controller that can reliably achieve a safe, upright lunar landing under nonlinear dynamics and irregular terrain.

Requirements:

- Navigate toward a designated landing zone
- Maintain near-upright orientation throughout descent
- Regulate vertical and horizontal velocity for soft touchdown
- Ensure simultaneous two-leg contact for a valid landing
- Use thrusters efficiently to minimize fuel expenditure

Approach: A **model-free RL** formulation where a Python-based environment simulator generates transitions using continuous state dynamics, thruster actions, rigid-body physics, collision checks, and shaped rewards. The control problem is posed as an MDP with continuous state space and a discrete 4-action thruster set.



2D World Definition

Coordinate System

- Environment defined on a 2D Cartesian plane:

$$(X, Y) \in \mathbb{R}^2$$

- Viewport (visible simulation window):

$$P_1 = \{(X, Y) : X \in [-16, 16], Y \in [-10, 22]\}$$

- Uniform gravitational acceleration g acting downward.

Center of Mass (COM)

$$\mathbf{p}_{\text{COM}}(t) = \begin{bmatrix} X_{\text{COM}}(t) \\ Y_{\text{COM}}(t) \end{bmatrix}$$

The COM fully determines the lander's body-frame placement.



Environment Visualization

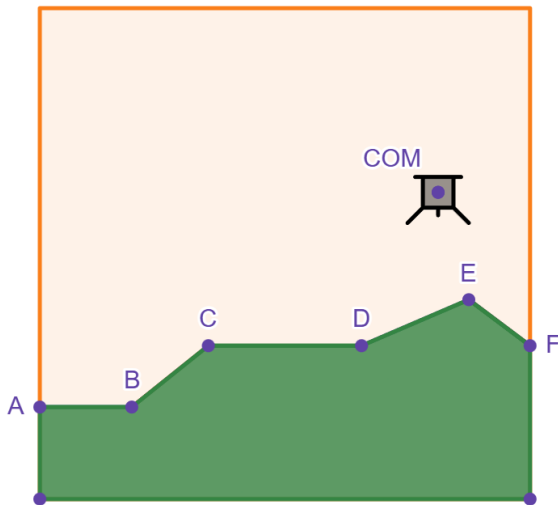


Figure: Viewport of the 2D lunar lander environment showing the lander geometry, thruster locations, legs, and the piecewise terrain.



Lander Geometry & Orientation

Rigid-Body Model

- Central chassis + two articulated legs.
- All geometric reference points are defined in the *body frame*.

Angle Convention

- Orientation θ measured from $+X$ axis.
- Upright configuration:

$$\theta = 90^\circ = \frac{\pi}{2}$$

- Positive rotation = anticlockwise.

Body-Frame Reference Points

Left thruster: $(-1, +1)$,

Right thruster: $(+1, +1)$,

Main engine: $(0, -1)$,

Left leg tip: $(-2, -2)$,

Right leg tip: $(+2, -2)$.



Body-to-World Coordinate Transformation

Rotation Mapping Any body-frame point $(x_{\text{loc}}, y_{\text{loc}})$ maps to world coordinates via:

$$\begin{aligned}x_w &= X_{\text{COM}} + \sin \theta \, x_{\text{loc}} + \cos \theta \, y_{\text{loc}} \\y_w &= Y_{\text{COM}} - \cos \theta \, x_{\text{loc}} + \sin \theta \, y_{\text{loc}}\end{aligned}$$

Leg Tip World Coordinates

$$X_R = X_{\text{COM}} + 2 \sin \theta - 2 \cos \theta,$$

$$Y_R = Y_{\text{COM}} - 2 \cos \theta - 2 \sin \theta,$$

$$X_L = X_{\text{COM}} - 2 \sin \theta - 2 \cos \theta,$$

$$Y_L = Y_{\text{COM}} + 2 \cos \theta - 2 \sin \theta.$$



Terrain Model & Forbidden States

Piecewise Linear Landing Surface

$$(-16, -4) \rightarrow (-10, -4) \rightarrow (-5, 0) \rightarrow (5, 0) \rightarrow (12, 3) \rightarrow (16, 0)$$

Defines the height function:

$$Y_{\text{ground}}(X)$$

Forbidden States (Terrain Penetration)

$$Y_{\text{COM}} < Y_{\text{ground}}(X_{\text{COM}})$$

$$S_{\text{forbidden}} = \{s : Y_{\text{COM}} < Y_{\text{ground}}(X_{\text{COM}})\}$$



State and Action Spaces

8-Dimensional State Vector

$$s = (X_{\text{COM}}, Y_{\text{COM}}, v_x, v_y, \theta, \omega, \ell, r)$$

- $\ell, r \in \{0, 1\}$: leg contact flags.
- Observed by the agent at each step.

Discrete Action Space

$$a \in \{0, 1, 2, 3\}$$

- 0: No thrust (NOOP - No Operation)
- 1: Left thruster (produces a rightward force F_{right})
- 2: Main engine (produces an upward thrust F_{main})
- 3: Right thruster (produces a leftward force F_{left})



Force & Torque Model

Forces in World Frame

$$F_x = F_{\text{main}} \cos \theta + (F_{\text{left}} - F_{\text{right}}) \sin \theta$$

$$F_y = F_{\text{main}} \sin \theta + (-F_{\text{left}} + F_{\text{right}}) \cos \theta - mg$$

$$a_x = F_x/m, \quad a_y = F_y/m$$

Torque Model

$$\tau_{\text{left}} = -F_{\text{left}}, \quad \tau_{\text{right}} = F_{\text{right}}$$

$$\tau_{\text{tot}} = F_{\text{right}} - F_{\text{left}}$$

$$\alpha = \dot{\omega} = \tau_{\text{tot}}/I$$



Discrete-Time Dynamics (Semi-Implicit Euler)

Linear Updates

$$v_x^{t+1} = v_x^t + a_x \Delta t, \quad X^{t+1} = X^t + v_x^{t+1} \Delta t$$

$$v_y^{t+1} = v_y^t + a_y \Delta t, \quad Y^{t+1} = Y^t + v_y^{t+1} \Delta t$$

Angular Updates

$$\omega^{t+1} = \omega^t + \alpha \Delta t, \quad \theta^{t+1} = \theta^t + \omega^{t+1} \Delta t$$

Reason for Use

- Stable for long rollouts.
- Standard in RL simulators and physics engines.



Episode Termination Conditions

Simulation stops when any of the following occur:

- COM touches or crosses ground:

$$Y_{\text{COM}} \leq Y_{\text{ground}}(X_{\text{COM}})$$

- Either leg contacts ground:

$$\ell = 1 \quad \text{or} \quad r = 1$$

Terminal Classification

- **Safe Landing:** both legs, on-pad, upright.
- **Crash:** any other termination mode.



Reward Function Overview

Per-Step Reward

$$r_t = r_{\text{pos}} + r_{\text{vel}} + r_{\theta} + r_{\ell r} + r_{\text{ctrl}}$$

Terminal Reward

$$r_{\text{term}} = \begin{cases} +100, & \text{safe landing,} \\ -100, & \text{crash.} \end{cases}$$

Total Episode Return

$$R = \sum_t r_t + r_{\text{term}}$$



Reward Components

Position Progress

$$r_{\text{pos}} = -c_{\Delta x} \Delta d_t$$

$$d_t = |X_{\text{COM}}(t) - x_{\text{pad}}|, \quad \Delta d_t = d_t - d_{t-1}$$

- $\Delta d_t < 0$: moved closer to pad $\Rightarrow r_{\text{pos}} > 0$
- $\Delta d_t > 0$: moved away from pad $\Rightarrow r_{\text{pos}} < 0$

Velocity Reduction

$$r_{\text{vel}} = -c_{\Delta v} \Delta v_t$$

$$v_t = \sqrt{v_x(t)^2 + v_y(t)^2}, \quad \Delta v_t = v_t - v_{t-1}$$

- $\Delta v_t < 0$: slowing down $\Rightarrow r_{\text{vel}} > 0$
- $\Delta v_t > 0$: speeding up $\Rightarrow r_{\text{vel}} < 0$



Reward Components (Continued)

Tilt Penalty

$$r_{\theta} = -c_{\theta} \left| \theta - \frac{\pi}{2} \right|$$

- θ is the current lander attitude (in radians).
- $\frac{\pi}{2}$ is the desired upright orientation.
- The absolute error $\left| \theta - \frac{\pi}{2} \right|$ measures how far the lander is tilted away from upright, independent of direction (left or right).
- Larger tilt \Rightarrow larger penalty; $c_{\theta} > 0$ scales how strongly we enforce upright attitude.



Reward Components (Continued)

Leg Contact Bonus

$$r_{lr} = c_\ell \ell + c_r r$$

- $\ell, r \in \{0, 1\}$ are binary indicators:
 - $\ell = 1$ if the left leg is in contact with the ground, else 0.
 - $r = 1$ if the right leg is in contact with the ground, else 0.
- Each leg contact yields a positive reward:
 - c_ℓ = bonus for left leg,
 - c_r = bonus for right leg (typically $c_\ell = c_r = 10$).
- Encourages the agent to achieve a stable, leg-based touchdown rather than hitting with the body.



Reward Components (Continued)

Control Penalty (Fuel Cost)

$$r_{\text{ctrl}} = -c_s u_{\text{side}} - c_m u_{\text{main}}$$

- $u_{\text{side}} \in \{0, 1\}$: 1 if any side thruster fires in this step, else 0.
- $u_{\text{main}} \in \{0, 1\}$: 1 if the main engine fires in this step, else 0.
- $c_s, c_m > 0$ are small per-step fuel costs (e.g., $c_s = 0.03$, $c_m = 0.3$).
- Penalizes excessive thrusting \Rightarrow encourages fuel-efficient and smoother trajectories.



Complete Per-Step Reward (Compact Form)

$$\begin{aligned} r_t = & -c_{\Delta x} \Delta d_t - c_{\Delta v} \Delta v_t - c_{\theta} \left| \theta(t) - \frac{\pi}{2} \right| \\ & + c_{\ell} \ell(t) + c_r r(t) \\ & - c_s u_{\text{side}}(t) - c_m u_{\text{main}}(t) \end{aligned}$$



Environment Workflow: Step Execution

- Read current state and convert agent action into corresponding thruster commands.
- Compute resulting forces and torques, then update position, velocity, angle, and angular rate using the selected integration method.
- Determine leg-tip world coordinates and check for interaction with the terrain profile.
- Handle ground contact:
 - latch contact flags,
 - stop downward motion,
 - correct any penetration into the surface.
- Form the next state and update step counter.
- Evaluate reward components using differential shaping and current posture/contacts.



Environment Workflow: Episode Control

- After each state update, the environment checks:
 - 1 **Viewport bounds** — terminate immediately if the lander leaves the visible area.
 - 2 **Ground / Leg Contact** — classify as safe landing or crash based on posture and landing zone.
 - 3 **Maximum Steps** — truncate if the episode exceeds the step limit.
- Based on the condition triggered, the environment:
 - applies appropriate terminal reward (if applicable),
 - sets `done` / `truncated` flags,
 - returns the next state, reward, and diagnostic information.



Diagnostic Information Returned

The environment returns a dictionary containing the following fields:

■ Step Tracking

- `step_count` — current timestep index
- `terminated_reason` — why the episode ended (if it did)

■ Leg-Tip World Coordinates

- `xr, yr` — right leg tip position
- `xl, yl` — left leg tip position
- `ground_r, ground_l` — terrain height under each leg

■ Applied Thruster Forces

- `F_main, F_left, F_right`
- `action` — discrete action chosen

■ Reward Diagnostics

- `r_pos, r_vel` — differential reward components
- `delta_d, delta_v` — change in distance and speed



Deep Q-Learning Overview

Objective: Learn $Q(s, a)$ that estimates

$$Q^\pi(s, a) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right]$$

DQN replaces a Q-table with a neural network:

$$Q_\theta(s, a) \approx Q^*(s, a)$$

Neural Network Architecture:

$$s \in \mathbb{R}^8 \rightarrow \text{FC}(128) \rightarrow \text{ReLU} \rightarrow \text{FC}(128) \rightarrow \text{ReLU} \rightarrow \text{FC}(4)$$

Replay buffer \mathcal{D} stores transitions (max 100,000):

$$(s_t, a_t, r_t, s_{t+1}, d_t)$$



Double DQN Logic

Standard DQN suffers from overestimation. Double DQN splits:

$$a^* = \arg \max_a Q_{\theta}(s_{t+1}, a) \quad (\text{online net})$$

$$y = r + \gamma Q_{\theta^-}(s_{t+1}, a^*) \quad (\text{target net})$$

Why target network? Freeze a copy of the Q-network to stabilize learning:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s_{t+1}, a')$$

- Maintains a separate target network θ^-
- Updated via **soft updates**:

$$\theta^- \leftarrow \tau \theta + (1 - \tau) \theta^-, \quad \tau = 0.005$$

This reduces instability caused by rapidly changing Q-targets.



Epsilon-Greedy Exploration and Loss Function

Exploration policy:

$$a = \begin{cases} \text{random action} & \text{with probability } \epsilon, \\ \arg \max_a Q(s, a) & \text{with probability } 1 - \epsilon \end{cases}$$

ϵ starts at 1.0 (pure exploration) and decays linearly to 0.1 over 150k frames.

Huber loss (smooth L1): $\mathcal{L}(\theta) = \text{Huber}(Q_\theta(s, a) - y)$

Standard for DQN-style algorithms and gives a smooth gradient near zero

Gradient clipping used: $\|\nabla_\theta\|_2 \leq 10$

Learning Rate: 3×10^{-4}



Training and Evaluation

Each step:

- 1 Compute ϵ and select action
- 2 Step environment
- 3 Store transition in replay buffer
- 4 Sample minibatch to update Q_θ
- 5 Compute Double DQN target
- 6 Update online network using Adam
- 7 Soft-update target network

Training stops when: $\text{AvgReward}_{100} > 250$

During evaluation:

- No exploration ($\epsilon = 0$)
- Pure greedy action: $a = \arg \max_a Q_\theta(s, a)$



Why Extend DQN?

Despite strong performance in environments like LunarLander, basic DQN still suffers from:

- Overestimation bias
- Inefficient sample usage
- Slow convergence
- Poor performance in noisy/complex environments
- Instability due to bootstrapping + function approximation

Extensions improve:

- Stability
- Sample efficiency
- Convergence speed
- Final performance



Possible Extensions

Several major extensions can significantly enhance DQN

- Our current DQN already uses:
 - Double DQN
 - Soft target updates
 - Huber loss + grad clipping
- Next easy improvements:
 - Dueling architecture
 - Prioritized replay
 - Multi-step returns
- For major gains:
 - Rainbow DQN
 - Distributional RL
 - Noisy layers



Training Metrics

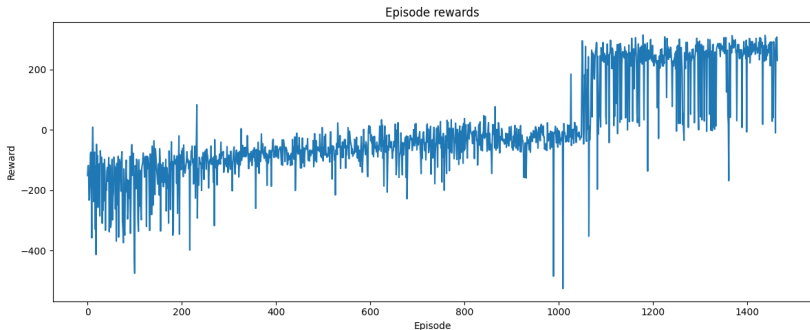


Figure 1: Reward vs Episodes - Training

- Up to 1100 episodes, the rewards are mostly negative due to larger value of epsilon
- Once algorithm becomes greedy, its reward start increasing



Evaluation Metrics

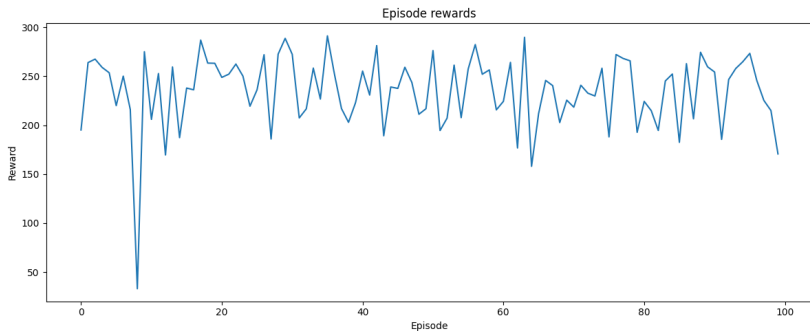


Figure 2: Reward vs Episodes - Evaluation



Questions?

Core Crunchers



Thank you.
