Task 2 Project Report

AVD894 - Reinforcement Learning in Practice

Team: The Reward Maximizers

Team Members:
Sahil Sachin Shete (SC22B113)
Saurabh Kumar (SC22B146)
Satyajeet Musmade (SC22B182)

1 Introduction

This task focuses on the implementation of a complete simulator for a system controlled using reinforcement learning. The objective is to set up an environment capable of generating sample trajectories under a simple control policy. No learning is performed in this task. The implemented simulator serves as the foundation for applying reinforcement learning algorithms in subsequent tasks.

2 System Description

The system under consideration is the Lunar Lander problem, implemented using the `LunarLander-v3` environment from the Gymnasium library. The environment simulates the physical dynamics of a lunar lander operating under gravity and controlled by discrete thruster actions. The lander's state is represented by an eight-dimensional continuous vector consisting of position, velocity, orientation, angular velocity, and ground contact indicators. The action space is discrete and consists of four possible engine commands.

3 Simulator Implementation

The simulator is set up by initializing the Gymnasium environment, which internally models the system dynamics, reward structure, and termination conditions. At each time step, the simulator accepts an action and updates the system state accordingly, returning the next state, reward, and a termination signal. An episode terminates when the lander successfully lands, crashes, or reaches a predefined time limit.

The simulator enables interaction through a standard agent–environment loop involving state observation, action selection, environment update, and reward observation. This interaction loop allows the generation of sample paths without requiring an explicit mathematical model of the system dynamics.

4 Control Policy

A simple control policy was implemented to demonstrate the functionality of the simulator. The policy selects actions based on the current state of the lander, such as its vertical velocity and orientation. For example, the main engine is activated when the lander is descending too rapidly, while side engines are used to correct excessive tilt. This policy does not involve any learning and is designed solely to produce meaningful sample trajectories.

5 Sample Trajectories

Using the implemented simulator and heuristic policy, sample trajectories were generated over complete episodes. Each trajectory consists of a sequence of state, action, and reward tuples, representing a sample path of the system evolution. The observed behavior confirms that the simulator correctly models the system dynamics and responds appropriately to control actions.

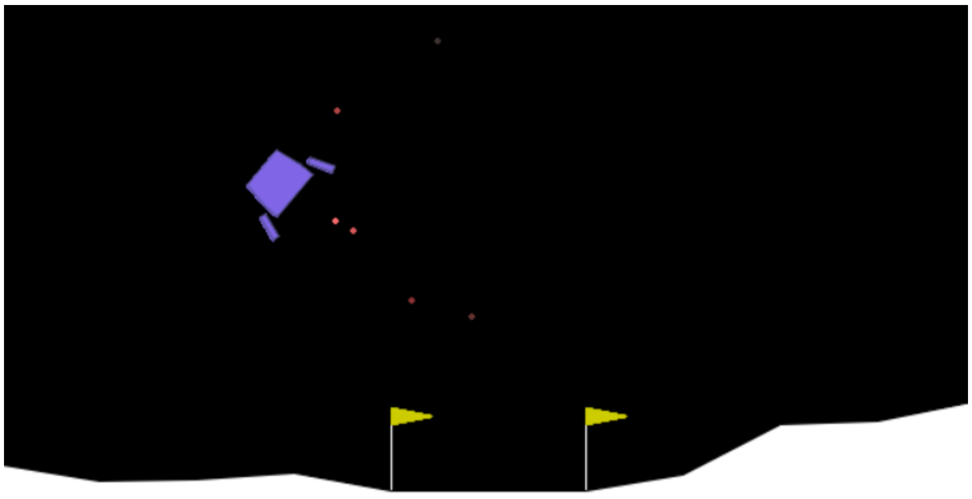

Figure 1: Visualization of the Lunar Lander simulation environment

6 Conclusion

In this task, a complete simulator for the Lunar Lander system was implemented using the Gymnasium framework. The simulator successfully generates sample paths under a heuristic control policy, fulfilling the requirements of Task 2. This implementation provides a reliable platform for applying and evaluating reinforcement learning algorithms in later stages of the project.

A Python Code for Task 2

Listing 1: Simulator and Policy Implementation

```python
import gymnasium as gym

# Implementation of a simple policy
def policy(state):
    x, y, vx, vy, angle, ang_vel, left, right = state

    # If falling fast, fire main engine
    if vy < -0.5:
        return 2

    # If tilted right, fire left engine
    if angle > 0.1:
        return 1

    # If tilted left, fire right engine
    if angle < -0.1:
        return 3

    # Do nothing
    return 0

def run_episode(env, policy_fn, max_steps=1000):
    state, _ = env.reset()

    trajectory = []
    total_reward = 0

    for t in range(max_steps):
        action = policy_fn(state)
        next_state, reward, terminated, truncated, _ = env.step(
           action)
        done = terminated or truncated

        trajectory.append((state, action, reward, next_state))
        total_reward += reward

        state = next_state
        if done:
            break

    return trajectory, total_reward

env = gym.make("LunarLander-v3", render_mode="human")

trajectory, total_reward = run_episode(env, lambda s: policy(s))

env.close()

print("Total␣reward:", total_reward)
print("Trajectory␣length:", len(trajectory))
```