This report outlines the implementation of a lunar cargo landing environment built using PyBullet, integrated with OpenAI's Gym library. Below is a breakdown of the code's key components:

Environment Setup

- **Library Imports:** The code imports essential libraries such as gym, numpy, and pybullet to facilitate environment creation and run physics simulations.
- **Direction Dictionary:** The dirDict dictionary defines ten possible actions, each associated with specific engine activations, allowing for different combinations of thrust direction.
- **Engine Thrust:** The engine_pos and thrust_dir variables define engine positions and their corresponding thrust directions when activated.

Class Definition

The core environment is encapsulated in the CargoLanderEnv class, which inherits from gym.Env, implementing the core functionalities of the lander environment.

- **Constructor:** Initializes the action space (10 possible actions) and the observation space (including 3D position, velocity, orientation, and ground contact status), and connects to PyBullet for simulation.
- **Reset Function:** The reset() method resets the environment at the start of each episode, including reinitializing the physics simulation, loading the ground, and randomly assigning the lander's starting position and orientation.

Observation and Step Execution

- **Observation Function:** The observe() function retrieves and returns the lander's state (position, velocity, orientation, and ground contact status) as an observation vector.
- **Step Function:** The step(action) method executes a single simulation step based on the selected action. It calculates and returns the updated observation, reward, fuel consumption, and determines whether the lander has landed successfully. The reward system includes bonuses for successful landings and penalties for crashes, excessive fuel use, and high landing speeds.

Rendering and Closing

- **Render Function:** The render() function is spared to enhance the visualization of the simulation.
- **Close Function:** The close() method disconnects from the PyBullet simulation, cleaning up resources.

Main Loop (Demo Code)

In the __main__ block, the environment is instantiated, and a loop is run to simulate 3 episodes, selecting action 1 with time step frequency and outputting the reward and duration for each.

Existing Codes/Libraries

This project leverages external libraries and tools to build the lunar cargo landing environment:

- **PyBullet** handles physics simulations, including collision detection, force application, and lander control.
- **OpenAI's Gym** provides a standardized structure for defining action and observation spaces, enabling integration with reinforcement learning algorithms.
- The rocket model was custom designed to suit the simulation's needs, while the terrain model was sourced from online repositories.[1]

Reflections

Changes to the Proposal: While the initial proposal featured a simplified lander model, increasing complexity during development (such as more realistic thrust and contact detection) enhanced the simulation's realism. The action space was refined by removing lateral engines and using the four bottom engines for omnidirectional control. Additionally, the action space increased to 8 actions to improve rotational control without adding structural complexity. The reward and penalty mechanisms were optimized by eliminating factors related to the lander's distance from the ground.

Simplifications in the Model: Initially, more detailed terrain features were considered, but a simpler terrain model was chosen, with trees and rocks representing landing obstacles. The lander learns to find a flat ground for landing. Engine forces were simplified to directly represent upward thrust. The model also streamlined the landing criteria by assuming a basic linear relationship between thrust and movement, omitting complex aerodynamics for simplicity. Contact detection was reduced to a binary state (landed or not), which works effectively but could benefit from more nuanced landing stability feedback.

Future Changes: Future improvements could involve refining the reward structure to encourage smoother landings and more precise lander control. Adding features like more varied terrains, engine failures, or environmental challenges could further enhance the simulation's complexity and realism.

[1] Terrain .obj download link, reference:
https://www.turbosquid.com/FullPreview/2105855