Faiza Abdullah

Final Project - NASA Space Mission AI Project

ITAI-2372-Artificial Intel Applications

Professor: Anna Devarakonda

KLUNAR NAVIGATOR CONCEPTUAL TRACK

Implementation Track: **Git Link:** https://github.com/AbdullahFaiza/KLunar-Navigator

Project Proposal

The KLunar - Navigator project proposes an AI-driven rover navigation system for lunar exploration, utilizing Q-learning to autonomously navigate a 10x10 grid from a start point (0,0) to a target (9,9) while avoiding obstacles. This conceptual solution aims to simulate lunar terrain challenges, offering a foundation for future real-world robotic missions.

Objectives

- Develop a functional Q-learning agent to optimize rover pathfinding.
- Create a Pygame-based visualization to demonstrate real-time movement.

- Ensure robustness against obstacles and grid boundaries.
- Prepare the system for scalability to larger grids or dynamic environments.

Scope: The project encompasses environment simulation, rover movement logic, Q-learning implementation, and visual feedback. It targets educational and research applications, with potential extension to space exploration planning.

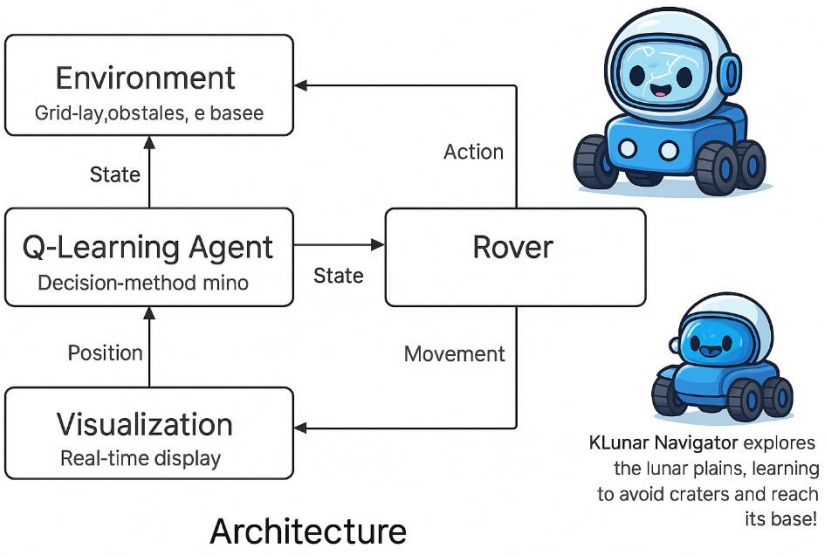
Benefits

- Enhances understanding of reinforcement learning in robotics.
- Provides a testable prototype for lunar navigation strategies.
- Supports educational outreach with an interactive visualization.

Resources

- Python 3.12 with Pygame, NumPy, and asyncio libraries.
- Computational resources for simulation (standard PC).

Architecture



Architecture

Detailed Solution Plan

Problem Statement

Design a system where a rover autonomously navigates a 10x10 lunar grid, avoiding obstacles and reaching a target, using Q-learning and real-time visualization.

Solution Approach

1. Environment Setup

- Class: LunarEnvironment
- Features: 10x10 grid, start (0,0), target (9,9), random obstacles (max 3).
- Implementation: Tracks rover_pos, handles resets, and defines step logic.

2. Rover Movement

- Class: Rover
- Features: Four actions (right, left, down, up), energy tracking, boundary, and obstacle checks.
- Implementation: Updates position based on valid moves, returns new state.

3. Q-Learning Agent

- Class: QLearningAgent
- Features: Q-table for 10x10x4 states, learning_rate (0.4), discount_factor (0.98), epsilon (0.15).
- Implementation: Chooses actions, learns from rewards (-1 base, 100 target, -50 obstacle), saves q_table.npy.

4. Visualization

- Class: Visualizer
- Features: Pygame window (600x600), renders rover (blue), target (green), obstacles (red).
- Implementation: Updates display per move, caps at 60 FPS.

5. Main Loop

- File: main.py
- Features: 100 episodes, 200 max steps per episode, async updates.
- Implementation: Manages setup, loop, and Q-table persistence.

Technical Details

- Sync: env.rover_pos forced to next_state in update_loop.
- Optimization: Reduced obstacles, adjusted learning parameters for convergence.
- Termination: max_steps ensures loop completion.

Future Enhancements

- Dynamic obstacle generation.
- Multi-rover coordination.
- 3D terrain simulation.

Testing Plan

Objective: Validate the rover's ability to navigate from (0,0) to (9,9), avoid obstacles, and complete 100 episodes within 200 steps each.

Test Cases

1. **Initial Movement**
    - **Input**: Start at (0,0), action 0 (right).
    - **Expected**: Moves to (0,1), reward = -1, done = False.
    - **Method**: Run main.py, observe Pygame and terminal.
2. **Obstacle Avoidance**
    - **Input**: Move toward obstacle (e.g., (1,0)).
    - **Expected**: Stays at current position, reward = -50, done = True.
    - **Method**: Check debug output, verify position.

3. **Target Reach**
    - **Input**: Move to (9,9).
    - **Expected**: reward = 100, done = True, episode ends.
    - **Method**: Track 100 episodes, confirm q_table.npy.
4. **Boundary Check**
    - **Input**: Action 3 (up) at (0,0).
    - **Expected**: Stays at (0,0), reward = -1, done = False.
    - **Method**: Test edge cases, monitor logs.
5. **Episode Completion**
    - **Input**: Run 100 episodes.
    - **Expected**: Logs Episode 100/100, creates q_table.npy.
    - **Method**: Execute full simulation, verify output.

Testing Environment

- Python 3.12, Pygame, NumPy, asyncio.
- Standard PC, VS Code for execution.

Success Criteria

- Rover moves visibly toward (9,9) in Pygame.
- 100 episodes complete within 20,000 total steps.
- q_table.npy generated with non-zero values.

Snapshots of Rover Movement