OpenAI Gym Report

Cao Zhihan, Cheng Yuchao, Teng Yu

I. Code explanation:

1. Main Features

Observation Space:

- The car's surroundings are represented by a Field of View (FOV) centered on the car's position.
- The position of the car on the map.
- The visit count of each grid cell to track which areas have already been explored.

Action Space:

- The car can move in four directions: up, down, left, and right.
- This forms a discrete action space where each action corresponds to a movement command.

Reward Structure:

Negative rewards (penalties) for actions that hinder progress:

- Collision with obstacles results in a penalty of -1.
- Revisiting explored areas gives a penalty that scales with the number of previous visits (penalty: -0.01 per visited grid).
- Movement incurs a small penalty of -0.1 to discourage excessive wandering.

Positive rewards:

- Discovering new, unexplored areas yields a small reward (0.01 per new grid).
- Completing the entire map exploration grants a larger reward (5).

2. Key Variables and Initialization

Map Representation:

- Obstacles, unexplored areas, and the car's position are represented by different integer values in the grid: OBSTACLE = 0, UNEXPLORED = 1, CAR = 2
- The car is represented as a 2x2 grid, and it occupies a grid space starting from a given initial position (INIT_POS).

Step Size and Map Dimensions:

- The car moves in discrete steps determined by STEP_SIZE = 5.
- The environment map is a grid of size MAP_SIZE = (40, 21).
- A total of 240 obstacles are placed randomly within the map.

Field of View (FOV):

- The FOV is represented as a 10x10 area around the car's current position, which is used to simulate its perception of the environment.

3. Classes and Functions

Integration with Gym:

The environment inherits from gym.Env, defining a custom RL environment. The action space is a discrete set of 4 actions, and the observation space includes the car’s position and FOV.
MapBuilder.py (custom-built) is imported to handle map construction.

4. Code Flow

- The car begins at the initial position in an unexplored environment.
- At each step, the car selects an action (move in one of four directions) and updates the environment based on that action. Users can view rendered map, fov_map and visit_count.

- Observations (its surroundings and position) and rewards (based on its actions and new state) are calculated, and this cycle repeats until exploration is complete.

5. Conclusion

The dummy_gym.py file creates a simulation environment for reinforcement learning. It employs a grid-based map where the car, controlled by the RL agent, must explore while avoiding penalties and collecting rewards. The use of OpenAI Gym allows for easy integration into RL workflows, and the modularity of the code suggests that additional features or complexities can be added as needed for further experimentation.

II. Existing Codes/Libraries

All existing codes are included in a zip file and has been uploaded to the GitHub repository.
Currently, no external tools or robotic simulators are being used. The required environment for code execution is specified in the environment.yml file.
The libraries currently in use are:

- gym==0.26.2
- jupyter==1.0.0
- matplotlib==3.5.3
- numpy==1.21.6

III. Reflections/Lessons Learned

1. dummy_gym.py Upgrades

In the proposal, we mentioned that the car will move in multiple steps, but in the process of code implementation, we initially only judged whether the car will collide with an obstacle at the end after moving multiple steps, ignoring the possibility that the car may collide in the process of moving multiple steps, to address it we adopted the method of generating the region of the travelling path, detecting whether this region exceeds the boundary or there is an obstacle inside the region, and satisfying one of them is considered as a collision, which solves this problem.
For now, we have only implemented penalties for collisions with obstacles. We are considering introducing a penalty system based on the car's proximity to obstacles, but this will depend on the effectiveness of the current approach in the training phase.

2. MapBuilder.py Upgrades

Initially, the Map Builder in the project did not take into account the dimensions of the cars, which would have led to exploration problems with the cars. The map is technically connected, but the size of the cars (e.g., 7x7) makes exploration of certain tight areas difficult.
Considering the importance of this problem, to address it we introduce a parameter called size, which adjusts the size of obstacles in the map as well as the size of passable paths according to the size of the vehicle (in this process, we use the idea of expansion, where each cell is expanded according to the size of the vehicle to enable the construction of the map) This adjustment ensures that the car can fully explore the map while maintaining the overall connectivity of the map.

3. Potential Improvements

In the future, we aim to add more randomness to obstacle placement and further refine the reward and penalty systems to enhance the overall performance of the environment. The current reward system is simple and might benefit from additional features such as prioritizing unexplored areas that are farther from the current position or giving more nuanced penalties for obstacles.