Technical Assessment

Reinforcement Learning

Objective:

Train a **Proximal Policy Optimization (PPO)** agent using **PyTorch** in a custom **PyBullet environment** to achieve efficient and collision-free parking of a simulated car.

Your primary goal is to **implement and fine-tune the PPO algorithm** while designing a reward function that encourages the car to park successfully without colliding with obstacles. The focus is on reward shaping and demonstrating your understanding of PPO and reinforcement learning principles.

Environment Description:

You will be working in a **custom PyBullet environment** named CarParking-v0. The environment simulates a **self-driving car in a parking lot**:

- The car must **navigate into a designated parking spot** between obstacles without colliding.
- The environment provides a **continuous action space**:
  - Throttle: Move forward or backward.
  - Steering: Turn left or right.
- The **observation space** includes:
  - Car position and orientation.
  - Car velocity.
  - Distance to obstacles and the target parking spot.

Task Requirements:

1. **Implement PPO:**
   - Use **PyTorch** for the neural network model and policy optimization.
   - Either implement PPO from scratch or use a library like **Stable-Baselines3** with custom modifications.
2. **Reward Function Design:**
   - The default environment has sparse rewards:
     - **+100** for successful parking.
     - **-100** for collisions or going out of bounds.

        - **-1 per timestep** to encourage efficiency.
    - Your task is to **modify and improve the reward function** to make learning efficient and effective. Consider adding:
        - **Distance-based rewards:** Positive feedback for moving closer to the goal.
        - **Collision penalties:** Significant negative rewards for hitting obstacles.
        - **Orientation rewards:** Bonus for proper alignment during parking.
        - **Smooth control rewards:** Penalize abrupt steering or acceleration changes (optional).

3. **Training and Fine-Tuning:**
    - Train the PPO agent using your modified reward function.
    - Tune hyperparameters like learning rate, batch size, and discount factor to achieve stable training and reliable performance.
    - Monitor and log the **average episode reward** to evaluate training progress.

    With effective reward shaping, the agent should begin learning a meaningful policy within 50,000 to 200,000 timesteps. Final performance can be evaluated after ~500,000 timesteps depending on compute and tuning.

Success Criteria:

The agent is considered successful if it:

- **Parks correctly without collision** in at least **90% of evaluation episodes** (out of 100).
- Achieves a consistent **average reward ≥ 90** over 100 evaluation episodes.
- Demonstrates **smooth and efficient manoeuvring** during parking.

Evaluation Instructions:
To evaluate your trained agent:

- Run the model for 100 episodes using predict(obs, deterministic=True).
- Log:
    - Success rate (% of episodes where car parks without collision).
    - Average episode reward.

- Number of collisions (if any).

Deliverables:

1. **Code:**
   - Python scripts or notebooks for the PPO agent and environment.
   - Clearly organized and well-commented code.
   - Configuration and hyperparameter settings used during training.
2. **Report:**
   - Explain your **reward function design** and how it improves performance.
   - Discuss the **training strategy and hyperparameter choices**.
   - Include **performance metrics and training curves** (e.g., average reward over episodes).
   - Briefly analyze the behavior of the trained agent.

Setup Instructions:

1. **Install PyBullet and Gym:**

   pip install pybullet gym stable-baselines3 torch

2. **Download the Environment Code:**

   Place the provided car_parking_env.py file in your working directory.

   This file defines a Gym-compatible custom environment named CarParkingEnv and registers it with the ID CarParking-v0. It should implement standard Gym functions like:

   - reset(): Initialize the environment.
   - step(action): Apply an action and return observation, reward, done, and info.
   - render(): Visualize the car in the environment.
   - close(): Gracefully shut down the environment.

3. **Verify Environment Installation:**

   import gym

   import car_parking_env # Registers the custom environment

```
env = gym.make("CarParking-v0")
obs = env.reset()
done = False
while not done:
  action = env.action_space.sample()
  obs, reward, done, info = env.step(action)
  env.render()
env.close()
```

4. **Implement PPO:**
   - Use a multi-layer perceptron (MLP) policy architecture.
   - Ensure proper handling of continuous actions (throttle, steering).
   - Train the agent with appropriate hyperparameters and monitor performance.

Submission:

- Upload your **code and report** to a GitHub repository or share a zip file.
- Include a **README.md** with instructions to run the code and any dependencies.

Please contact Karanjot Singh (karanjot.singh@razer.com) for any issues regarding the assignment.

All the best!! Let us see how your agent navigates the parking lot!