ME5418 - Gym Coding Report

Group 42: HUANG Yiming, LI Yuwei, MA Yuxuan

October 18, 2024

1 Introduction

File $robotic_arm_gym_v1.py$ defines the OpenAI Gym environment of our project. The environment models a 3-DOF robotic arm in a space with a switch button, where the arm's objective is to efficiently move its end-effector to press the button.

- The robot arm uses discrete torque actions to control its joints and the option to press the button.
- The observation space includes information about joint angles, end-effector and target positions, velocities, and the push state.
- The reward system encourages efficient movement toward the target and penalizes unnecessary actions, promoting a learning process that prioritizes precision and speed in task completion.

2 Code Explanation

$class RoboticArmEnv(MujocoEnv, utils.EzPickle)$:
The class defines the entire environment, including the observation space, action space, the simulation process, and how the robot interacts with the environment. It extends from MujocoEnv and EzPickle and simulates a robotic arm with 3 joints and the ability to push a button.

2.1 Key Components

- *Action space*: Defined as $Discrete(54)$, where 54 possible discrete actions are divided into two main parts: **Torque Control** and **Button Press**. The first 27 (3^3) actions control the torques applied to the three joints (either increasing 1.0, decreasing 1.0, or keeping them constant) which is represented by $action\_map = []$ inside function $step(self, action)$.
  The next 27 actions include a combination of torque control along with an attempt to press a button. If $action < 27$, it means it's a torque action without pressing action, while $action27$ 53 represents the torque + button operation.
- *Observation Space*: Consists of 19 elements, capturing various state information about the robot:
  1. Cosine and sine of joint angles for all 3 joints (6 values total).
  2. The X, Y, Z coordinates of both the end-effector and the target (6 values total).
  3. The angular velocities of the joints (3 values total).
  4. The relative position of the end-effector to the target in X, Y, Z directions (3 values).
  5. The push state of the end-effector, indicating whether it pushed the button (0 or 1).
- *_calculate_reward Function*: The reward system is defined as follows:
  1. **Dense negative reward**: The agent gets a small penalty proportional to the distance between the end-effector and the target.
  2. **Small reward**: If the end-effector is near the target (distance < 0.05), the agent receives a small positive reward.
  3. **Penalty for invalid button presses**: If the button is pressed but the end-effector is far from the target, the agent is penalized.
  4. **Large reward**: A reward of 100 is given when the button is successfully pressed, and the episode ends.

2.2 Code Flow and Execution

step Function defines how the environment responds to an action taken by the agent. It applies the selected action to the joints by updating the torques by following steps of: *Reset push state→ Action processing→ Simulation→ Observation Update.*
The class starts by defining the observation and action spaces during initialization. When the step function is called, the environment processes the action taken by the agent, updates the torques, and simulates the movement of the robot. The robot's observations, rewards, and terminal conditions are then updated accordingly. The agent learns to control the robot by observing the reward signals and adjusting its actions.

2.3 Demo Code for Testing

File *test.py* conduct a test for the gym. It will simulate and render for 10000 time step in MuJoCo simulator. For each time step, it will pick a random action in action space to control the robotic arm, and get observation, reward and done signal via step() function accordingly. The action and observation will be output to the terminal.
Every 1000 time step or when the task is done, the environment will be reset, which the position of the target and the state of the robotic arm will be randomized.
If the test works, you will see a robotic arm moving erratically in the Mujoco simulator, surrounded by a small red ball as the target.

3 Existing Codes/Libraries

3.1 MuJoCo(Multi-Joint dynamics with Contact)

In this project, we use MuJoCo as our simulator. MuJoCo is a physics engine widely used in robotics, control, and reinforcement learning. Developed by Emo Todorov, it allows for efficient simulation of multi-body systems with contact dynamics, making it ideal for applications such as robotic manipulation and biomechanics.

3.2 Mujoco-py

mujoco-py is a Python wrapper that allows users to interface with MuJoCo using Python. This wrapper is commonly used in reinforcement learning research because it connects MuJoCo's powerful physics simulation with Python's broad ecosystem for machine learning (e.g., TensorFlow, PyTorch).

3.3 Gym

Gym was used as the foundation for defining our reinforcement learning environment. We extended the Env class to define the custom action and observation spaces for our 3-DOF robotic arm, as well as to implement the reward function and environment reset logic.

4 Reflections/Lessons Learned

Several suggestions were applied to the project after we discuss our proposal with Prof and TAs, mainly includes changes on the **robot model** used for the task and the definition of **action_space**:

- A 3-DOF robotic arm is applied rather than a 6-DOF one mentioned in the proposal. Because our goal is simplified to only a push execution without any rotation, there's no need to use a 6-DOF arm which is unnecessary and will cause the task to become complex. As a result, the observation space is rewritten accordingly which is shown above.
- The action is changed into torque control of the 3 joints rather than directly controlling the angle of joints which is designed in our proposal. This makes the state space continuous at the same time keeps action space discrete. The task becomes more interesting when we design in this way.
- **Assumption/Simplification** is applied to the goal: The simulation goal is only to reach the target position (close enough) and execute press button action. And the goal is assumed to be at a fixed position within the robot arm's workspace.
- **Potential Changes**: There may exist adjustments on the values of rewards (only the values) based on training performance.