Final_Project_PPO

> This project implements reinforcement learning (RL) for trajectory generation of a UR5 robot arm using the Proximal Policy Optimization (PPO) algorithm.

---

1. System Setup

1.1 macOS Setup

```
brew install glfw
conda create -n ur5 python=3.10
conda activate ur5

pip install mujoco gymnasium numpy
pip install mujoco-python-viewer
pip install "dm_control>=1.0.0"
pip install matplotlib

cd cleanrl
pip install -r requirements/requirements.txt
```

1.2 Windows Setup

```
conda create -n ur5 python=3.10
conda activate ur5

pip install glfw
pip install mujoco gymnasium numpy
pip install mujoco-python-viewer
pip install "dm_control>=1.0.0"
pip install matplotlib

cd cleanrl
pip install -r requirements/requirements.txt
```

---

2. Running the Training Code

2.1 Making the UR5 Environment Discoverable

- Before training, ensure Python can locate your custom environment folder by adding it to `PYTHONPATH`.

```
cd ..
pwd  # Example: /home/user/Desktop/UR5

export PYTHONPATH=<current path>:$PYTHONPATH
# e.g.) export PYTHONPATH=/home/user/Desktop:$PYTHONPATH
```

2.2 Logging Training Data with Weights & Biases (wandb)

We use **wandb** for experiment tracking.

- **Please sign up for W&B** (https://wandb.ai)
- Type the commend below for log in on terminal

```
wandb login
```

- Paste your API key when prompted.

2.3 Running PPO

```
python cleanrl/cleanrl/ppo_continuous_action.py
```

---

3. Reward Function & Parameter Tuning

You **must modify the following two sections in env.py** to achieve the desired behavior.

3.1 Reward Configuration

- Below is the editable reward configuration inside env.py.

```
self.reward_cfg = {
    dist_success_thresh": 0.03,
    "time_penalty": 0.01,
    "collision_penalty": 10.0,
    "success_bonus": 100.0,
    "orient_weight": 0.1,
    "orient_power": 2.0,
    "progress_scale": 5.0,
}
```

3.2 Reward Function

```
    def _compute_reward(self, dist, prev_dist, collided: bool, success: bool):
```

- The following reward function is only an example and may need modification to improve performance depending on the task.
- Typically, it can be composed of terms such as distance to the target, goal-reaching rewards, and time-delay penalties.
- Each term can be designed using L1 loss, L2 loss, exponential functions, logarithmic functions, or other formulations depending on the desired behavior.

---

4. PPO Hyperparameter Tuning

- You can modify the following inside ppo_continuous_action.py:
- Key PPO Parameters
  - total_timesteps: Total training duration
  - num_envs: Parallel environments for faster sample collection
  - num_minibatches: Batch splitting for PPO updates
  - learning_rate: Controls policy/critic update speed
  - gamma: Discount factor
- Neural Network Configuration
  - MLP hidden layers
  - Activation functions

---

5. Result

- Before running inference.py, please modify line 29 as shown below.

```
agent.load_state_dict(torch.load("runs/<your run
name>/ppo_continuous_action.cleanrl_model", map_location=device))
```

- You can visualize the output by running inference.py.

```
python cleanrl/cleanrl/inference.py
```