Homework: PPO + Velocity Walking + Motion Tracking (Unitree G1)

This homework has three parts. Part 1 is a smaller pure-NumPy PPO exercise. Parts 2 and 3 use the Unitree G1 in mjlab (MuJoCo-Warp + rsl_rl PPO).

The robot-control tasks are intentionally lightweight: first direct velocity walking, then a single motion-tracking task with a mostly complete framework.

Installation

```
conda create -n hw python=3.10
conda activate hw
pip install -r requirements.txt
```

For Parts 2 and 3 in this workspace, use:

```
conda activate mjlab
```

A GPU is recommended for Parts 2 and 3.

Project Layout

```
hw_py/
  part1_ppo.py                # NumPy PPO components (Part 1)
  part2_walk.py               # velocity walking reward + observation design
  part3_dance_single.py       # single-trajectory tracking parameter tuning
hw_mjlab/
  walk/                       # shared G1 velocity-walking config
  dance/                      # shared single tracking reference + env config
  tracking_rewards.py         # reusable shaping utilities for walking/tracking
  train.py                    # rsl_rl PPO training helper
  evaluate.py                 # rollout, metric plot, and video helper
```

Students only edit files inside `hw_py/`.

Submission

Submit the following files:

- `hw_py/part1_ppo.py`
- `hw_py/part2_walk.py`
- `hw_py/part3_dance_single.py`

- one Part 2 checkpoint `.pt` file trained from your implementation
- one Part 3 checkpoint `.pt` file trained from your implementation
- `hw_py/part2_walk_LinVel_error.png`
- `hw_py/part2_walk_video.mp4`
- `hw_py/part3_dance_single_error.png`
- `hw_py/part3_dance_single_video.mp4`

If you submit multiple checkpoints for the same part, clearly mark which one is the final version used for grading.

We will check submitted `.pt` files for duplication/similarity. Reused, shared, or trivially renamed checkpoints will be treated as plagiarism.

Score Breakdown

- Part 1 PPO math: **20%**
- Part 2 direct velocity walking: **40%**
- Part 3 single motion tracking: **40%**

Part 1 - PPO Advantage Estimation and Loss Computation (20%)

Implement key PPO components using NumPy in hw_py/part1_ppo.py: Monte Carlo advantage, TD residual advantage, GAE, clipped policy loss, and value loss.

```
python -m hw_py.part1_ppo
```

Part 2 - Direct Velocity Walking (40%)

The Unitree G1 follows joystick-style velocity commands `(v_x, v_y, omega_z)` on flat ground. This is a direct velocity-walking task, not a motion-imitation or human-likeness objective.

Edit hw_py/part2_walk.py. The base configuration already contains the necessary proprioceptive observations and regularization rewards. Students add two compact observation terms and replace two command-tracking reward terms:

1. `observe_velocity_error` - command-tracking context, such as commanded minus measured body-frame twist.
2. `observe_stability_context` - small balance context, such as projected gravity, angular velocity, or vertical drift.
3. `track_linear_velocity` - reward matching commanded xy velocity.
4. `track_angular_velocity` - reward matching commanded yaw rate.

The final walking policy is trained with more than these two tracking rewards. The actor observation already includes base velocity, angular velocity, projected gravity, joint position/velocity, previous action, and command. The critic additionally receives foot height, air time, contact state, and contact forces. The inherited reward set keeps upright, pose, body angular velocity, angular momentum, joint-limit, action-rate, foot-clearance, foot-swing, foot-slip, soft-landing, and self-collision terms. These inherited terms provide the regularization needed for a usable walking policy; students should keep their additions compact and focused on command tracking.

Run the script; it trains a PPO policy and evaluates a fixed forward command of `1.0 m/s`, saving:

- `hw_py/part2_walk_LinVel_error.png`
- `hw_py/part2_walk_video.mp4`

```
python -m hw_py.part2_walk
```

Grading. Mean absolute forward-velocity error should be below **0.15 m/s** for full credit. Partial credit is given if the robot walks stably but does not hit the target speed exactly.

Part 3 - Single Motion Tracking (40%)

The G1 receives one fixed periodic reference trajectory. The tracking layout is adapted from BeyondMimic-style motion-command tracking while staying inside the local mjlab/rsl_rl stack: the policy observes phase-conditioned reference states, target joint position/velocity, and root-velocity error; the reward combines joint tracking, root linear/angular velocity tracking, and link pose/velocity tracking from `hw_py/motion/g1_hiphop_tracking.npz`, plus the inherited stability regularizers. The upstream BeyondMimic motion-tracking implementation is available as a larger Isaac Lab project, but it is not vendored here because its simulator, asset, and motion-file pipeline differ from this lightweight homework. The reward implementation is complete. Students only tune four interpretable parameters in hw_py/part3_dance_single.py:

- `JOINT_POS_STD`
- `LINK_POSE_STD`
- `JOINT_POS_WEIGHT`
- `LINK_POSE_WEIGHT`

Recommended search ranges:

- `JOINT_POS_STD`: `0.20` to `0.70`
- `LINK_POSE_STD`: `0.30` to `1.00`
- `JOINT_POS_WEIGHT`: `1.5` to `5.0`
- `LINK_POSE_WEIGHT`: `0.4` to `1.8`

The expected workflow is to run the default setting once, inspect the plot/video, adjust one or two parameters, and repeat two or three times. The other reward parameters are intentionally fixed so that the tuning task stays interactive without turning into a large search problem. The provided default setting is

intentionally reasonable but not fully tuned, so students should be able to improve it with a small number of runs.

Run the script; it trains a PPO policy, evaluates one rollout, and saves:

- `hw_py/part3_dance_single_error.png`
- `hw_py/part3_dance_single_video.mp4`

```
python -m hw_py.part3_dance_single
```

To check/play the single-trajectory tracking result without retraining, load a checkpoint and call `play()`. This rolls out one fixed hiphop trajectory, prints the mean joint-tracking error, and writes the plot/video to `hw_py/`.

The current `mjlab` build in this homework only supports offline rendering (`rgb_array`) for this environment. It does **not** provide a pop-up live viewer for `play()`, so the intended visualization path is the saved mp4.

Use the latest checkpoint automatically:

```
CKPT=$(find logs/rsl_rl/hw_part3_dance_single_g1 -name 'model_*.pt' | sort -V |
tail -1)
test -n "$CKPT"
conda run --no-capture-output -n mjlab python - <<PY
from hw_py.part3_dance_single import play
play("$CKPT")
PY
```

Or play a specific checkpoint:

```
conda run --no-capture-output -n mjlab python - <<'PY'
from hw_py.part3_dance_single import play
play("logs/rsl_rl/hw_part3_dance_single_g1/<timestamp>/model_1200.pt")
PY
```

To export the full motion video instead of the default short check rollout, enable `full_motion=True`. For the current hiphop reference this uses all 7426 frames from `hw_py/motion/g1_hiphop_tracking.npz`: warning(be patient, it takes a long time to export the full video)

```
conda run --no-capture-output -n mjlab python - <<'PY'
from hw_py.part3_dance_single import play
play("logs/rsl_rl/hw_part3_dance_single_g1/<timestamp>/model_1200.pt",
full_motion=True)
PY
```

After `play()` finishes, inspect the generated files:

- `hw_py/part3_dance_single_error.png`
- `hw_py/part3_dance_single_video.mp4`

Grading. Mean absolute joint tracking error should be below **0.15 rad** for full credit. Partial credit is given if the robot clearly follows the rhythm and reduces tracking error above the threshold.

Tips

- For walking, start with compact observations and simple exponential velocity rewards before adding extra shaping.
- For tracking, tune standard deviations first, then reward weights. Avoid changing many parameters in one run.
- For debugging, lower `num_envs` and `max_iterations` in each script's `main()` call.
- Logs are written under `logs/rsl_rl/<experiment>/<timestamp>/`. Inspect with `tensorboard --logdir logs/`.

Files and directories prefixed with `DELETE_ME_` are legacy copies, generated cache, or temporary smoke-test artifacts. They are intentionally marked for cleanup and are not required by the current homework.