Installation

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

Homework Part 1: PPO Advantage Estimation and Loss Computation (40%)

In this part of the assignment, you will implement key components of the Proximal Policy Optimization (PPO) algorithm using NumPy. PPO is a widely used reinforcement learning algorithm that relies on reliable estimation of advantages and stable policy updates. This task is focused on understanding and implementing different methods of **advantage estimation**, as well as the **loss functions** used in PPO.

Tasks

1. **Monte Carlo Advantage Estimation**
    - Implement the Monte Carlo (MC) method to compute advantages as the difference between the discounted return G_t and the estimated value $V(s_t)$.
2. **TD(0) Residual Advantage Estimation**
    - Implement the temporal difference (TD) advantage estimator using the one-step TD error:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

3. **Generalized Advantage Estimation (GAE)**
    - Implement GAE to interpolate between MC and TD(0) methods using a λ parameter. This balances bias and variance in the advantage estimates.
4. **Policy Loss Computation**
    - Implement the clipped surrogate objective used in PPO:

$$L^{\texttt{CLIP}} = \mathbb{E}_t\left[\texttt{min}\left(r_t A_t, \texttt{clip}(r_t, 1-\epsilon, 1+\epsilon) A_t\right)\right]$$

    - Include an entropy bonus to encourage exploration.
5. **Value Loss Computation**
    - Compute the mean squared error (MSE) between the predicted value estimates and the target returns.

Requirements

- Use only NumPy for implementation.
- Use the provided checking functions to validate each component:
    - check_monte_carlo

- check_td_residual
- check_gae
- check_policy_loss
- check_value_loss

Deliverable

Complete all TODO sections in hw_py/part1_ppo.py. Ensure all components pass their respective checks without modification to the test code.

Homework Part 2: Reward Function Design for Getup Task (30%)

In this part of the assignment, you are tasked with implementing a custom reward function that enables a quadruped robot to **get up from a fallen state** and stabilize its posture. This is a crucial component in reinforcement learning for locomotion tasks, where reward shaping directly impacts learning efficiency and final behavior quality.

Objective

Design and implement a reward function inside the step method of the MyGetupEnv class that encourages the robot to:

1. **Reach and maintain a desired body height** (e.g., standing upright).
2. **Achieve a stable orientation** (i.e., upright posture with gravity vector aligned).
3. **Minimize deviation from the default pose** (joint angles close to a reference configuration).
4. **Minimize body angular velocity** (i.e., avoid spinning or jerky motion).

The reward function should output a single scalar that appropriately balances these objectives. You may use weighted terms or other heuristics to shape the behavior.

```
# TODO: your code here.
# Hint: consider these objectives:
# 1. body height
# 2. body orientation
# 3. joint position (error to default pose)
# 4. body angular velocity (error to zero)

reward = ...
```

Deliverable

Complete all TODO sections in hw_py/part2_getup.py. Make sure to output hw_py/part2_height_error.png and hw_py/part2_video.mp4.

- The Height error on hw_py/part2_height_error.png should be less than **0.15** to get full points.
- If your implementation is generally correct, you will still get most of the points even if the error is higher than the threshold.

Homework Part 3: Walking Reward Function Design and PPO Training (30%)

In this part of the assignment, your goal is to design a reward function that enables a quadruped robot to **walk forward smoothly and stably**. You will work inside the MyWalkEnv environment and train a PPO agent to follow velocity commands accurately.

> ⚠ **Note:** Walking control is a challenging problem. Therefore, most of the reward terms have already been implemented for you. You **only need to implement** the following two terms:
>
> - tracking_lin_vel: reward for tracking the target **linear velocity**.
> - tracking_ang_vel: reward for tracking the target **angular velocity** (yaw rate).

These two components are critical to ensure the robot moves in the commanded direction.

Objective

The complete reward function encourages:

1. **Tracking linear and angular velocity commands**.
2. **Maintaining stable body height and upright orientation**.
3. **Minimizing joint and actuator-related costs**.
4. **Ensuring proper foot-ground interaction** (e.g., no slipping, appropriate clearance).
5. **Encouraging rhythmic foot lifting and air time** for natural walking.

The final reward is computed as a weighted combination of many of these factors, already provided in the starter code.

Reward Code

The final reward expression is structured as:

```
reward = (
        tracking_lin_vel
        + 0.5 * tracking_ang_vel
        + 0.2 * reward_height
        + -5.0 * reward_orientation
        + -1.0 * rew_termination
        + 0.5 * rew_pose
        + -0.0002 * rew_torques
        + -0.01 * rew_action_rate
        + -0.001 * rew_energy
        + -0.1 * rew_feet_slip
        + -2.0 * rew_feet_clearance
        + -0.2 * rew_feet_height
        + 0.1 * rew_feet_air_time
        + -1.0 * rew_dof
)
```

Deliverable

Complete all TODO sections in hw_py/part3_getup.py. Make sure to output hw_py/part3_LinVel_error.png and hw_py/part3_video.mp4.

- The Height error on hw_py/part3_LinVel_error.png should be less than **0.15** to get full points.
- If your implementation is generally correct, you will still get most of the points even if the error is higher than the threshold.