MuJoCo Playground

streamlining simulation, training, and sim-to-real transfer onto robots

Marie Elster
UMass Amherst

06/18/2025

Marie Elster

- Computer Science and Electrical Engineering rising junior at UMass Amherst
- Alum of the Break Through Tech AI program at MIT
- Currently working at the UT PREP Summer Outreach program as an Advanced Engineering Intern here at UT
- Hobbies: Tennis, art, climbing

1) What is MuJoCo Playground?

- **MuJoCo stands for "Multi-Joint dynamics with Contact"**
  - Physics engine for facilitating research and development in robotics, biomechanics, graphics and animation

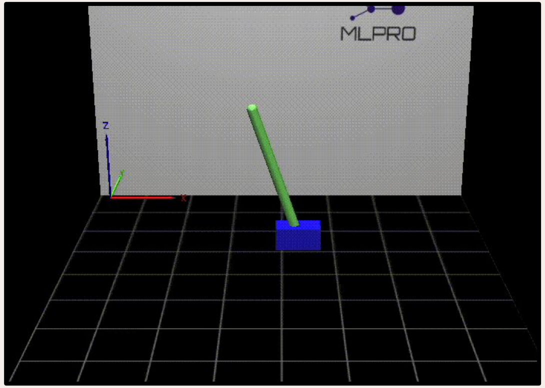
- **MuJoCo Playground** is built on MuJoCo XLA (MJX), a JAX-based branch of the MuJoCo physics engine + Brax + Madrona batch renderer
  - https://arxiv.org/abs/2502.08844 Feb 2025

- How is it used?

  1) Create a simulated environment that matches the real world
  2) Encode desired robot behavior with a reward function
  3) Train a policy in simulation
  4) Deploy to the robot

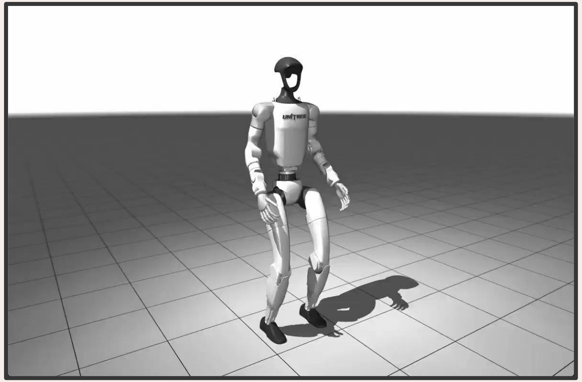
2) What is MuJoCo Playground?

Includes environments in three different categories:



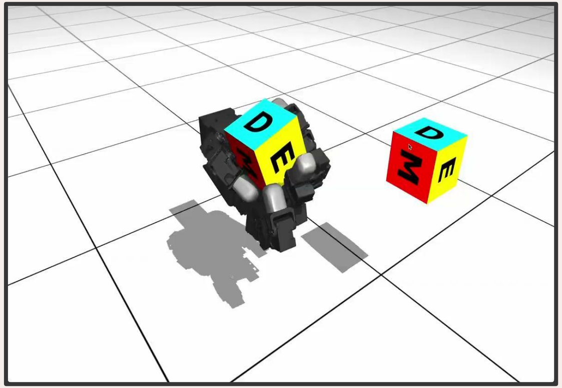
DeepMind Control Suite

Basic control tasks
(e.g. Cartpole)



Locomotion

quadrupeds (Go1), humanoids, bipedal robots: can be used to train and deploy joystick, fall recovery, handstand policies etc.



Manipulation

Hands, arms,
pick-and-place, etc.

Why do we care?

Reinforcement learning with transfer to hardware (sim-to-real) is emerging as a leading model in modern robotics, but RL is computationally intensive and time consuming.

MuJoCo makes this process more accessible:

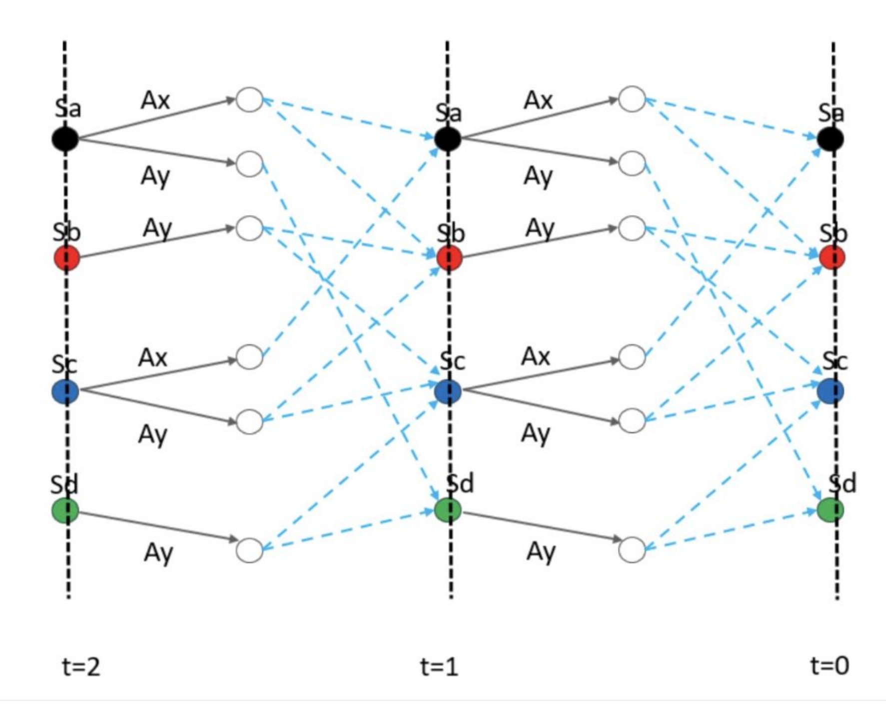
- Fast: Training tasks complete in minutes
- Cheap: Runs on a single GPU (e.g., RTX 4090)
- Simple: pip install playground
- Colab-ready

Markov Decision Process (MDP)

MDP models decision-making with these elements:
S: State space (observations)
A: Action space (motor commands)
P: Transition function: probability and next state (environment dynamics)
R: Reward function: state x action → reward
γ: Discount factor (future reward weighting)

Goal: Find optimal policy that maximizes expected cumulative reward

$$\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_t \gamma^t R(s_t, a_t)\right]$$

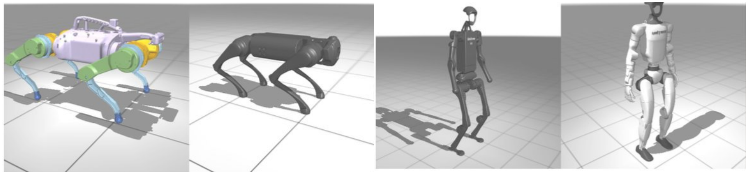


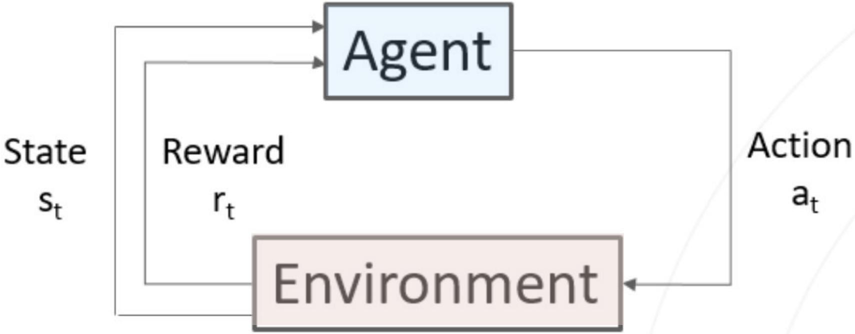
Locomotion: The RL Loop

An **agent** learns a policy to maximize rewards through trial and error in an **environment.**

Policy: Maps observations to actions

Reward: Encodes task success The better the behavior, the higher the reward.





Locomotion: The RL Loop

An **agent** learns a policy to maximize rewards through trial and error in an **environment**

Policy: Maps observations to actions

Reward: Encodes task success The better the behavior, the higher the reward.

Robot	Type	Environment
Google Barkour	Quadruped	JoystickFlatTerrain, JoystickRoughTerrain
Berkeley Humanoid	Biped	Joystick
Unitree G1	Biped	Joystick
Booster T1	Biped	Joystick
Unitree Go1	Quadruped	JoystickFlatTerrain, JoystickRoughTerrain, Getup, Handstand, Footstand
Unitree H1	Biped	InplaceGaitTracking, JoystickGaitTracking
OP3	Biped	Joystick
Boston Dynamics Spot	Quadruped	JoystickFlatTerrain, JoystickGaitTracking, Getup

The total reward is a weighted sum of reward terms (positive and negative):

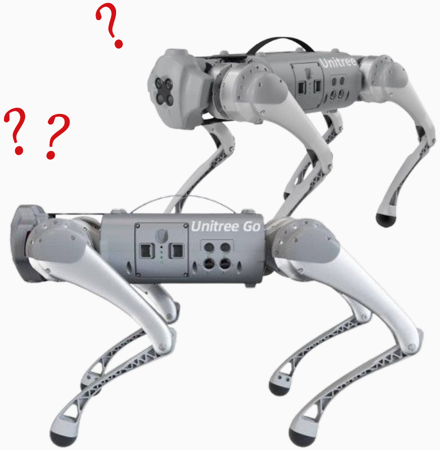
$$r_{\text{total}} = \sum_i w_i r_i,$$

Reward	Expression
Linear Velocity Tracking	$r_v = k_v \exp\left(-\|cmd_{v,xy} - v_{xy}\|^2/\sigma_v\right)$
Angular Velocity Tracking	$r_\omega = k_\omega \exp\left(-\|cmd_{\omega,z} - \omega_z\|^2/\sigma_\omega\right)$
Feet Airtime	$r_{\text{air}} = \text{clip}\left((T_{\text{air}} - T_{\text{min}}) \cdot C_{\text{contact}}, 0, T_{\text{max}} - T_{\text{min}}\right)$
Feet Clearance	$r_{\text{clear}} = k_{\text{clear}} \cdot \|p_{f,z} - p_{f,z}^{\text{des}}\|^2 \cdot \|v_{f,xy}\|^{0.5}$
Feet Phase	$r_{\text{phase}} = k_{\text{phase}} \cdot \exp\left(-\|p_{f,z} - r_z(\phi)\|^2/\sigma_{\text{phase}}\right)$
Feet Slip	$r_{\text{slip}} = k_{\text{slip}} \cdot \|C_{f,i} \cdot v_{f,xy}\|^2$
Orientation	$r_{\text{ori}} = k_{\text{ori}} \cdot \|\phi_{\text{body,xy}}\|^2$
Joint Torque	$r_\tau = k_\tau \cdot \|\tau\|^2$
Joint Position	$r_q = k_q \cdot \|q - q_{\text{nominal}}\|^2$
Action Rate	$r_{\text{rate}} = k_{\text{rate}} \cdot \|a_t - a_{t-1}\|^2$
Energy Consumption	$r_{\text{energy}} = k_{\text{energy}} \cdot \|\dot{q} \cdot \tau\|$
Pose Deviation	$r_{\text{pose}} = k_{\text{pose}} \cdot \exp\left(-\|q - q_{\text{default}}\|^2\right)$
Termination (Penalty)	$r_{\text{termination}} = k_{\text{termination}} \cdot \text{done}$
Stand Still (Penalty)	$r_{\text{standstill}} = k_{\text{standstill}} \cdot \|cmd_{v,xy}\|$
Linear Velocity in Z (Penalty)	$r_{\text{lin\_z}} = k_{\text{lin\_z}} \cdot \|v_z\|^2$
Angular Velocity in XY (Penalty)	$r_{\text{ang\_xy}} = k_{\text{ang\_xy}} \cdot \|\omega_{x,y}\|^2$

RL Locomotion with robotic dogs:
How do we make a Go1 do a handstand?

The Unitree Go1 is a quadruped robot modeled in MuJoCo Playground

How do we make it do a handstand (balance on the front legs)?



MDP for robotic dog handstand

State Space

Continuous

(a) Gravity projected in the body frame
(b) Base linear and angular velocity
(c) Joint positions and velocities
(d) Previous action

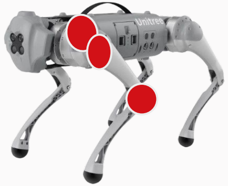
(e) (Optional) User command for joystick-based tasks

Action Space

Continuous

Torques applied to each motor
(12 total, 3 per each leg)

A relative joint position is used

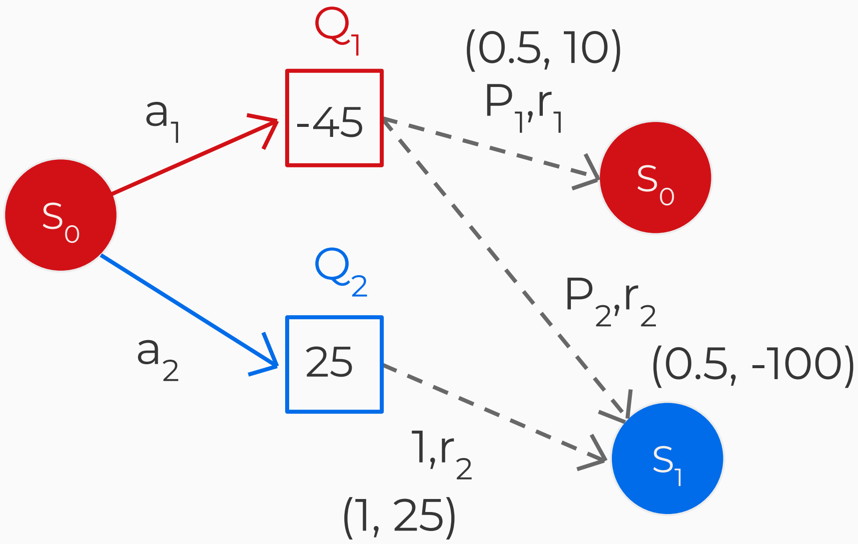


Reward function

Weighted sum of reward terms:

- **Uprightness:** Reward vertical positioning
- **Torque minimization:** Reward low control effort
- **Stability:** Reward low angular velocity
- **Slip:** Penalize foot sliding
- **Termination penalty:** Penalize falling

Back to our MDP:



$Q_1 = 5 + (-50) = -45$
$Q_2 = 25$

$\pi(s)$

s_0	a_2
s_1	
⋮	⋮
s_n	a_m

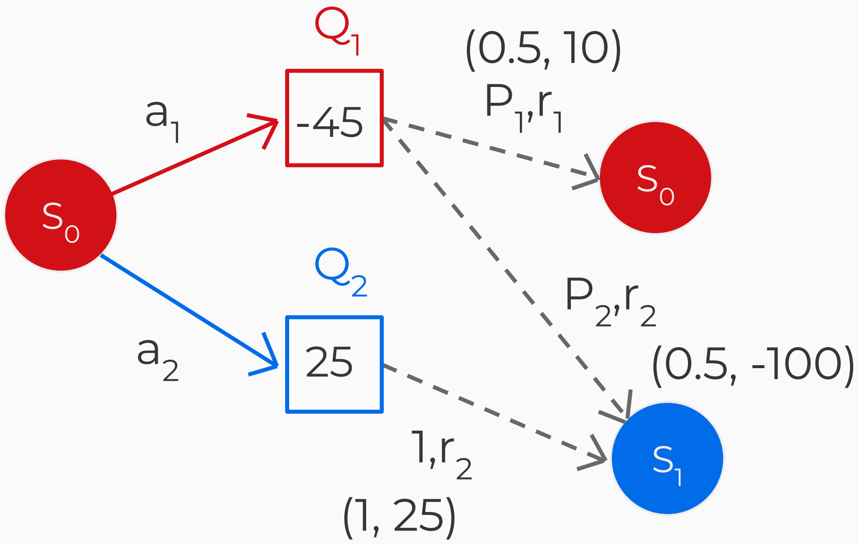
Policy table

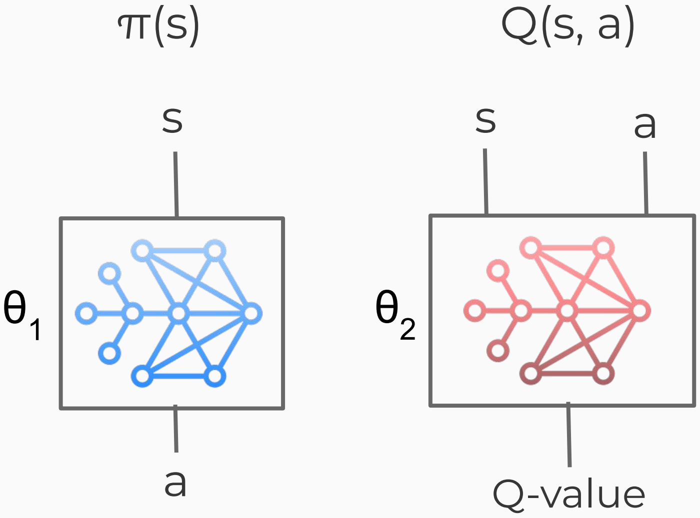
$Q(s, a)$

	a_1	a_2
s_0	-45	25
⋮	⋮	⋮
s_n	-	-

Q-value table

Back to our MDP:





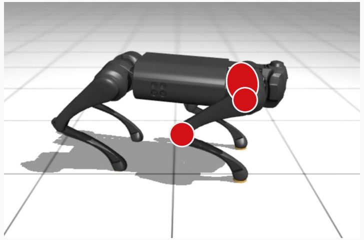
Policy Network
Actor

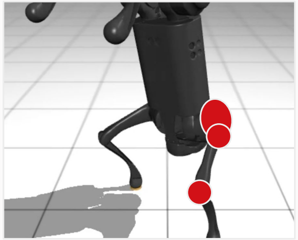
Q-Network
Critic

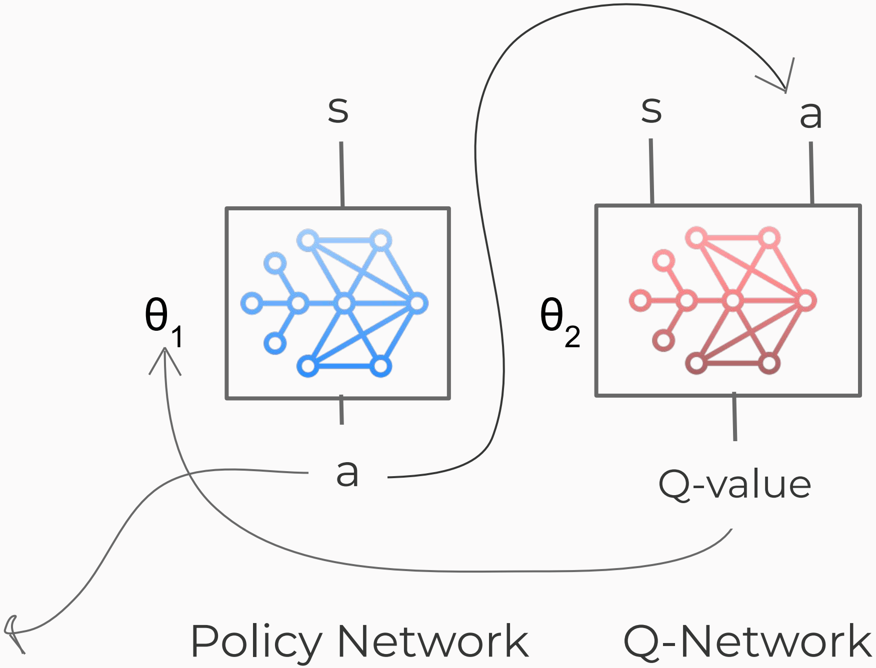
$Q_1 = 5 + (-50) = -45$
$Q_2 = 25$

Network architecture

- Asymmetric actor–critic setup
    - Q value network receives additional sensor readings
    - Improves training
- Both networks use a three-layer multilayer perceptron (MLP) with hidden sizes of 512, 256, and 128.
- Optimized using PPO
    - Prevents drastic policy changes

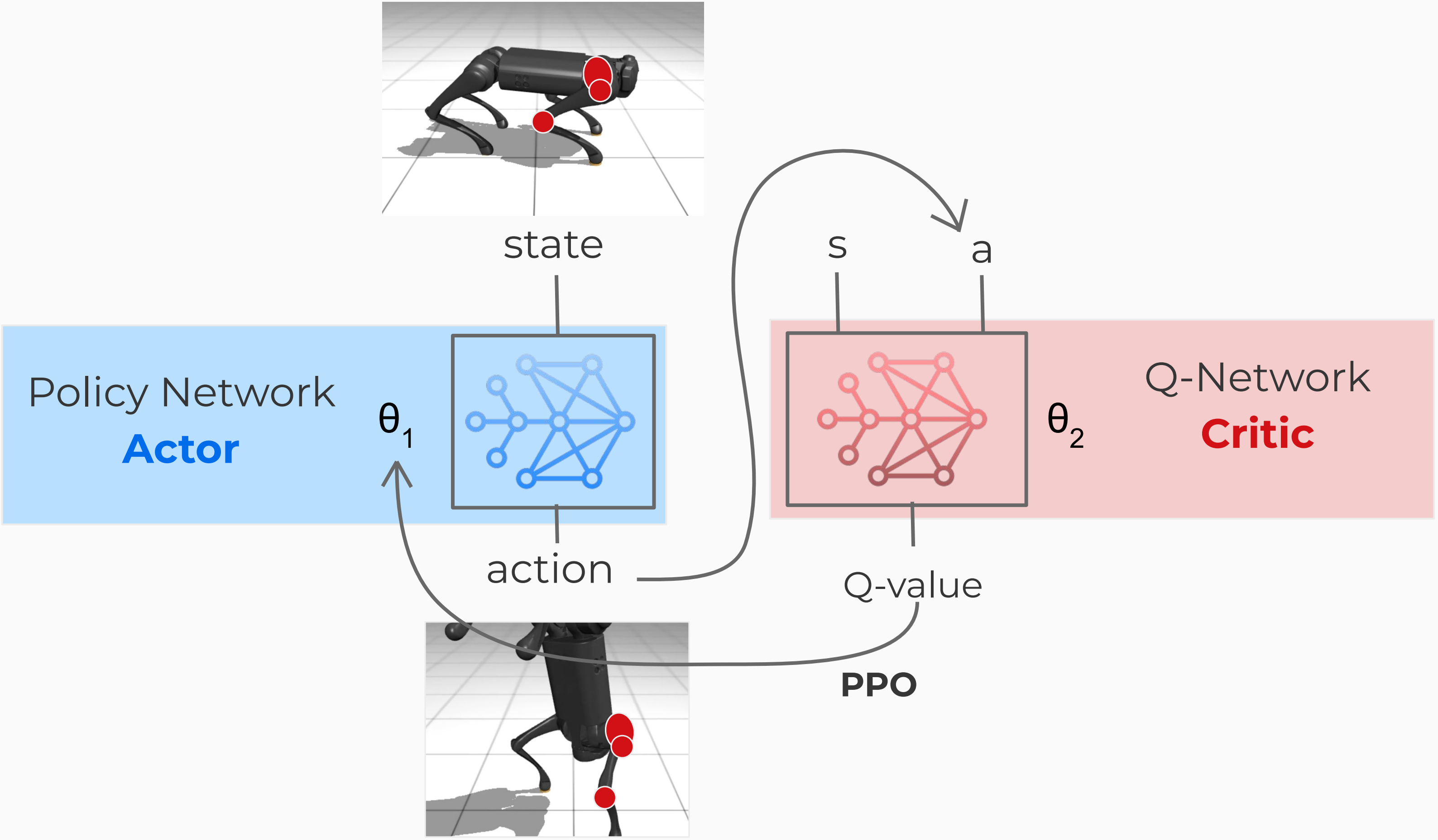






Policy Network
Actor

Q-Network
Critic

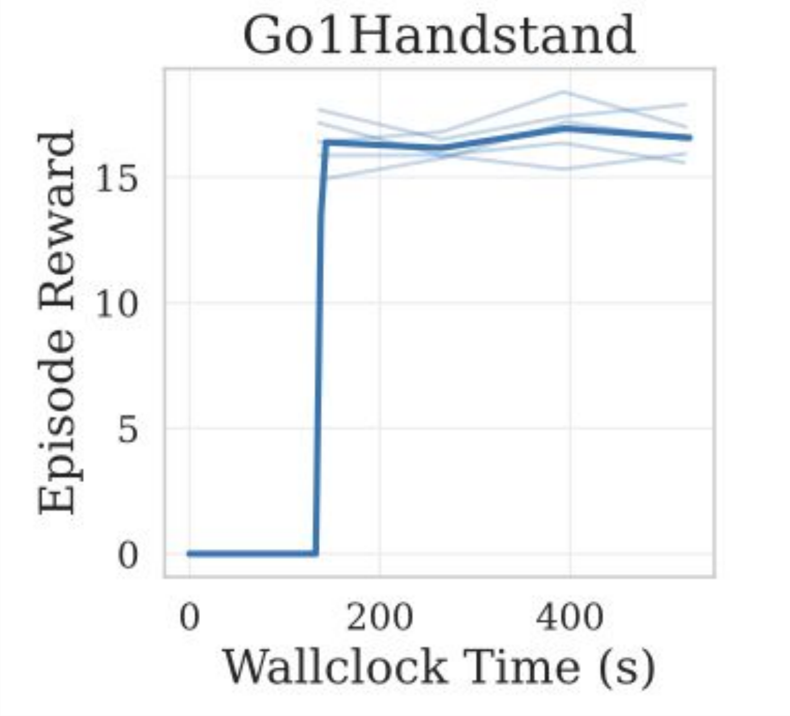

state
s
a
Policy Network
Actor
θ_1
θ_2
Q-Network
Critic
action
Q-value
PPO

Training (demo)

Zero fine-tuning

Fine tuned to minimize energy consumption

RL Training Results



Trained with PPO

Environments are run across 5 seeds on a single A100 GPU

Env	PPO Steps per Second
BarkourJoystick	385920 ± 2162
BerkeleyHumanoidJoystickFlatTerrain	120145 ± 484
BerkeleyHumanoidJoystickRoughTerrain	30393 ± 44
G1Joystick	106093 ± 131
Go1Footstand	204578 ± 906
Go1Getup	96173 ± 230
Go1Handstand	204416 ± 738
Go1JoystickFlatTerrain	417451 ± 2955
Go1JoystickRoughTerrain	291060 ± 727
H1InplaceGaitTracking	289372 ± 1498
H1JoystickGaitTracking	291018 ± 1111
Op3Joystick	198910 ± 406
SpotFlatTerrainJoystick	404931 ± 2710
SpotGetup	266792 ± 1038
SpotJoystickGaitTracking	407572 ± 4091

Sim-to-Real Transfer: How do we make it work in real life?

How do we account for instability?
Differences in the environment, robot, terrain?

Domain randomization

To reduce the sim-to-real gap, several parameters are randomized during training:

- **Sensor noise**: All sensor readings are corrupted with noise.
- **Dynamic properties:** Physical parameters that are difficult to measure precisely (e.g., link center-of-mass, reflected inertia, joint calibration offsets).
- **Task uncertainties:** Ground friction and payload mass.

Forces policy to be robust to a range of conditions