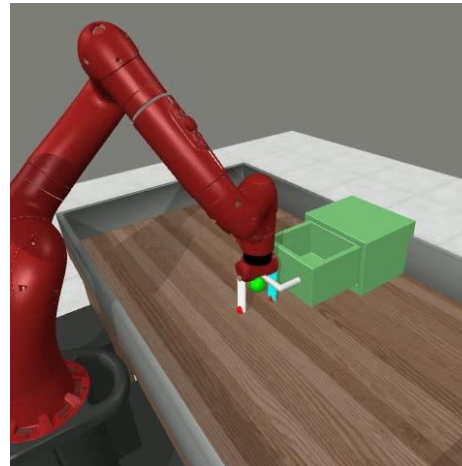


Mujoco + Metaworld Tutorial:

From Physics Simulation to Manipulation Tasks

Embodied AI: Perception, Representation and Action

Outline



- physics engine
- dynamics, contacts, friction, rendering

- task suite
- goals, rewards, success metrics, benchmarks

- controller / policy
- chooses actions from observations

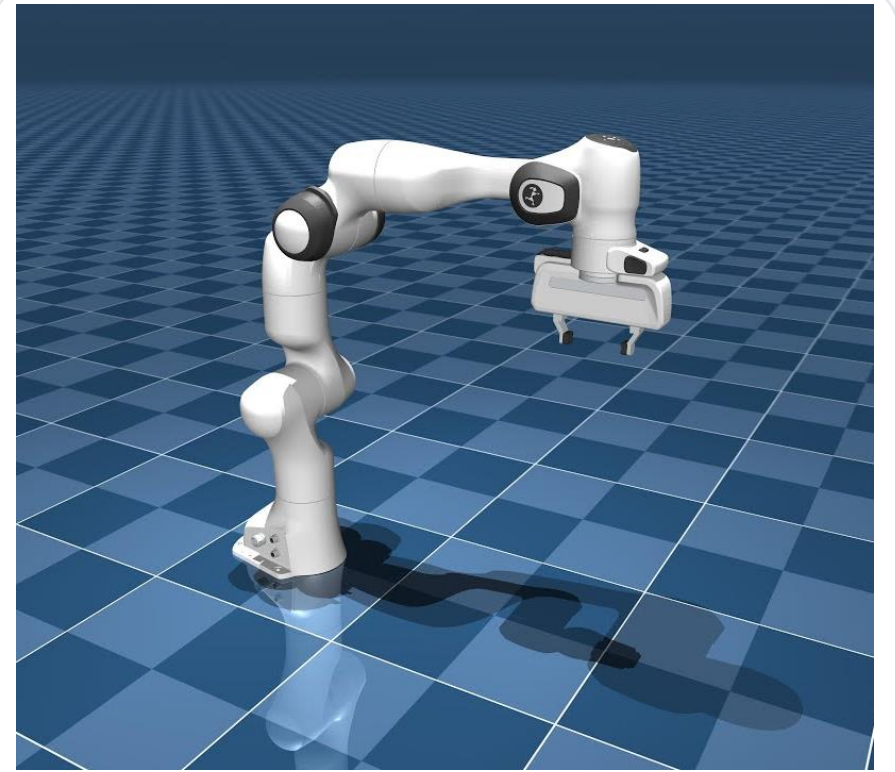
MuJoCo

What is it?

What can MuJoCo be used for?

How to use MuJoCo

- **Multi-Joint dynamics with Contact**
- Real-time simulator for rigid-body dynamics (articulated robots)
- **Strong at contacts:** collision, friction, constraints
- **Input:** model file (MJCF XML) + control signals
- **Output:** states (qpos/qvel), end-effector pose, contacts, rendered frames



Example MuJoCo scene (robot + floor + lighting)

MuJoCo

What is it?

Fast iteration

- Try ideas without risking hardware.

What can MuJoCo be used for?

Repeatable experiments

- Same seed & reset → comparable results.
- Easy ablations and baselines.

How to use MuJoCo

Visual debugging

- See contacts, failures, and trajectories.
- Tune mass/friction/damping

Important reminder:

Simulation \neq reality. Differences in contacts, sensors, and dynamics create a sim-to-real gap. For today, we use simulation as a fast, safe learning and prototyping sandbox.

MuJoCo

What is it?

What can MuJoCo be used for?

How to use MuJoCo

Model files

- MuJoCo's native format is MJCF — an XML scene description language
- You describe: bodies, joints, geoms (collision/visual), actuators
- MuJoCo can also load URDF (popular in ROS), but it is more limited
- Key intuition: edit XML → the simulated world changes immediately

MJCF is human-editable (change size/mass/friction, then reload)

```
<!-- MJCF (XML) example -->
<mujoco>
  <worldbody>
    <body name="arm" pos="0 0 0">
      <joint name="j0" type="hinge" axis="0 0 1"/>
      <geom type="capsule" size="0.03 0.20"/>
    </body>
  </worldbody>
  <actuator>
    <motor joint="j0" ctrlrange="-1 1"/>
  </actuator>
</mujoco>
```

MuJoCo

What is it?

What can MuJoCo be used for?

How to use MuJoCo

Load Sawyer in MuJoCo

- Load Sawyer MJCF (XML) with MuJoCo:
`MjModel.from_xml_path` → `MjData`
- Open viewer and step physics in a loop
- Change one XML parameter → reload → see change



Sawyer robot arm (used across many MetaWorld tasks)

MuJoCo

What is it?

What can MuJoCo be used for?

How to use MuJoCo

```
demo_mujoco_load_xml.py > ...
1 import time
2 from pathlib import Path
3 import xml.etree.ElementTree as ET
4
5 import mujoco
6 import mujoco.viewer
7
8 XML_PATH = "metaworld\\assets\\sawyer_xyz\\sawyer_drawer.xml"
9
10 # --- Demo toggles ---
11 USE_MODIFIED_XML = False
12 DRAWER_POS = (0.0, 0.9, 0.0) # try (0.1, 0.7, 0.0) and re-run
13
14 # A reasonable "home" pose for the mocap-controlled hand
15 HAND_INIT_POS = (0.0, 0.6, 0.2)
16 # HAND_INIT_POS = (-0.2, 0.6, 0.4)
17 HAND_INIT_QUAT = (1.0, 0.0, 1.0, 0.0)
18 RESET_STEPS = 60
19 SETTLE_STEPS = 120
20 SETTLE_VEL_NORM = 1e-3
21
22 # Use direct joint setting, but derive a "normal" pose from the mocap home.
23 USE_HOME_QPOS = True
24
25 # If HOME_QPOS is None, it will be derived from the mocap-home pose at runtime.
26 # You can still override it with explicit values if you want.
27 HOME_QPOS = None
28
29 def _reset_mocap_welds(model: mujoco.MjModel) -> None:
```

naconda3\envs\metaworld\python.exe c:\Users\Zevin\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher 51632 -- D:\Document\CodeWorkspace\Metaworld\demo_mujoco_load_xml.py

(metaworld) D:\Document\CodeWorkspace\Metaworld>

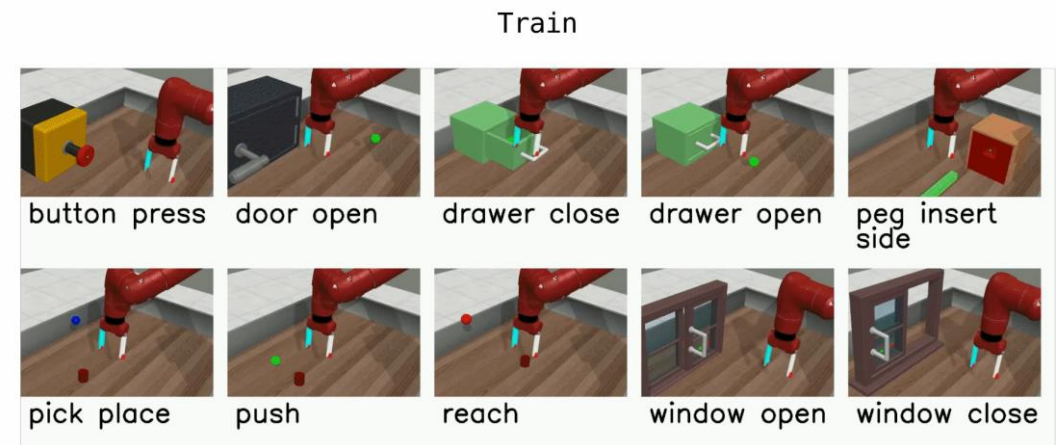
MetaWorld

What is it?

What can MuJoCo be used for?

How to use MuJoCo

- Open-source benchmark suite for robot manipulation (built on MuJoCo)
- Designed for multi-task RL and meta-RL evaluation
- Includes ~50 distinct manipulation tasks (Reach, Push, Pick-Place, Door, ...)
- Standardized success metrics (e.g., success rate) for fair comparisons



MetaWorld

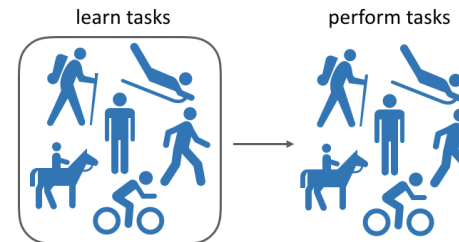
What is it?

What can MetaWorld be used for?

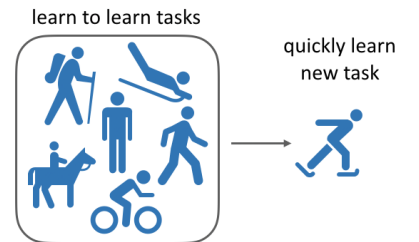
How to use MetaWorld

- **Action:** learn controllers / policies that move the end-effector to goals
- **Representation:** learn task-agnostic features from diverse interactions
- **Evaluation:** compare algorithms on standardized tasks and success rates
- **Generalization:** train on tasks/goals and test on unseen variations

multi-task reinforcement learning



meta reinforcement learning



Multi-task RL vs Meta-RL (intuition)

MetaWorld

What is it?

What can MetaWorld
be used for?

How to use
MetaWorld

API intuition

- Create environment
- Call `reset()` to get an observation
- Each step: send a 4D action → get next obs + reward + success info
- Action meaning: $[\Delta x, \Delta y, \Delta z, \text{gripper}]$

Minimal usage (from official README)

```
import gymnasium as gym
import metaworld

env = gym.make('Meta-World/MT1',
               env_name='reach-v3', seed=42)
obs, info = env.reset()

for t in range(200):
    action = [dx, dy, dz, g]
    obs, r, term, trunc, info = env.step(action)
```

Action = [Δx , Δy , Δz , gripper]

Small Cartesian moves each step → behavior becomes very intuitive to watch.

MetaWorld

What is it?

What can MetaWorld be used for?

How to use MetaWorld

The screenshot shows a code editor with the following content:

```
187 def _parse_args() -> DemoConfig:
194     parser.add_argument("--max-steps", type=int, default=20_000)
195     parser.add_argument("--no-realtime", action="store_true")
196     parser.add_argument("--noise-scale", type=float, default=0.15)
197     parser.add_argument(
198         "--action-smooth",
199         type=float,
200         default=0.20,
201         help="0..1, bigger = more responsive random walk",
202     )
203     args = parser.parse_args()
204
205     seed = None if args.seed < 0 else int(args.seed)
206     return DemoConfig(
207         env_name=str(args.env_name),
208         seed=seed,
209         print_hz=float(args.print_hz),
210         max_steps=int(args.max_steps),
211         realtime=not bool(args.no_realtime),
212         noise_scale=float(args.noise_scale),
213         action_smooth=float(args.action_smooth),
214     )
215
216
217 if __name__ == "__main__":
218     run(_parse_args())
219
```

The terminal output at the bottom shows the following execution results:

```
=0.18999999999999999 in_place=0.13328890369876714 ||qpos||=2.886039345
[1153] reward=+0.740 t=2.55 tcp=[+0.503, +0.567, +0.002] goal_site=[+0
=0.18999999999999999 in_place=0.13328890369876714 ||qpos||=2.896955466
[1166] reward=+0.732 t=2.71 tcp=[+0.506, +0.502, +0.003] goal_site=[+0
=0.18999999999999999 in_place=0.13328890369876714 ||qpos||=2.874010121
[1179] reward=+0.724 t=2.87 tcp=[+0.506, +0.440, +0.003] goal_site=[+0
t=0.18999999999999999 in_place=0.13328890369876714 ||qpos||=2.85451316
```

Takeaways

Summary

- MuJoCo: MJCF/URDF → simulate with `mj_step` → visualize with viewer
- MetaWorld: MuJoCo + tasks + rewards + success metrics (benchmarks)
- You can control Reach with simple 4D actions: $\Delta x \Delta y \Delta z$ + gripper
- Next step: replace greedy control with a learned policy (RL / imitation)

Resources

- **MuJoCo docs:** mujoco.readthedocs.io
- **MetaWorld repo:** github.com/Farama-Foundation/Metaworld
- **Meta-World paper:** arxiv.org/abs/1910.10897
- **Easy extension:** try Push / Pick-Place

Q&A prompts

- Installation / common pitfalls
- How to switch tasks
- Using images instead of state
- Sim-to-real considerations

Thanks for listening
Q & A