CSC415 Course Project: Simulation Environment Setup

Comprehensive Installation Guide & Verification

Instructor Version

Overview

This document provides step-by-step instructions to install and verify the five supported simulation environments for the course project.

Prerequisites:

- **OS:** Linux (Ubuntu 20.04/22.04) is highly recommended. macOS (Apple Silicon) is supported by most but may require specific workarounds. Windows users are advised to use WSL2 (Windows Subsystem for Linux) with GUI support.
- **Python Manager:** We strongly recommend using **uv** for ultra-fast environment management and dependency resolution.
- **GPU Requirement: Running code on a GPU is highly recommended.** Training RL agents, especially in visual or parallelized environments (like ManiSkill or mjlab), is computationally intensive. CPU-only training will be significantly slower and may limit the scope of experimentation.

---

1 Meta-World

Description: A benchmark for meta-reinforcement learning and multi-task learning consisting of 50 distinct robotic manipulation tasks.
Repository: https://github.com/Farama-Foundation/Metaworld

Installation

Meta-World is now maintained by the Farama Foundation. It relies on the `mujoco` python bindings.

1. Create and activate a fresh virtual environment using uv:

```
# Create environment
uv venv metaworld_env --python 3.10

# Activate environment
source metaworld_env/bin/activate
```

2. Install the package via uv pip:

```
uv pip install metaworld
```

Verification

Run the following Python script to ensure the environment loads and renders.

```
import metaworld
import random

# Initialize the benchmark (MT1 = Multi-Task 1 specific task)
ml1 = metaworld.ML1('pick-place-v2')
env = ml1.train_classes['pick-place-v2']()
task = random.choice(ml1.train_tasks)
env.set_task(task)

obs = env.reset()
for i in range(50):
    action = env.action_space.sample()  # Random action
    obs, reward, done, info = env.step(action)
    env.render()  # Opens a viewer window
print("Meta-World installed successfully.")
```

---

2 ManiSkill

Description: A large-scale GPU-parallelized physical simulation benchmark for generalizable manipulation skills.
Repository: https://github.com/haosulab/ManiSkill

Installation

ManiSkill requires **Vulkan** support. **GPU acceleration is critical here;** running parallel environments on CPU is not recommended.

1. Create and activate a uv environment:

```
uv venv maniskill_env --python 3.10
source maniskill_env/bin/activate
```

2. Install Pytorch (ensure CUDA version matches your driver):

```
uv pip install torch torchvision --index-url https://download.pytorch.org/whl/
    ↪ cu118
```

3. Install ManiSkill (latest version):

```
uv pip install mani_skill
```

Verification

ManiSkill includes a built-in demo script. Run the following command in your terminal:

```
# Runs a random agent on the PickCube environment
python -m mani_skill.examples.demo_random_action -e PickCube-v1 --render-mode human
```

Note: On first run, this command will prompt you to download necessary assets (Y/N). Select Yes.

---

3 DeepMind Control Suite (DMC)

Description: A set of physics-based continuous control tasks (cartpole, cheetah, walker) powered by the MuJoCo physics engine.
Repository: https://github.com/deepmind/dm_control

Installation

1. Create and activate a uv environment:

```
uv venv dmc_env --python 3.10
source dmc_env/bin/activate
```

2. Install via uv pip:

```
uv pip install dm_control
```

3. (Optional) Install `pygame` for easier rendering callbacks in custom scripts:

```
uv pip install pygame
```

Verification

```
from dm_control import suite
from dm_control import viewer

# Load the cartpole swingup task
env = suite.load(domain_name="cartpole", task_name="swingup")

# Define a policy (random)
def random_policy(time_step):
    del time_step  # Unused
    return [0.0]   # Action dimension for cartpole is 1

# Launch the interactive viewer
viewer.launch(env, policy=random_policy)
```

---

4 RLBench

Description: A challenging large-scale benchmark for robotic manipulation featuring 100+ completely unique tasks.
Repository: https://github.com/stepjam/RLBench

Important Note on Dependencies

RLBench relies on **CoppeliaSim** (formerly V-REP). You generally must use **CoppeliaSim Edu V4.1.0**. Newer versions often break the Python bindings (PyRep).

Installation Steps

1. **Download CoppeliaSim V4.1.0 (Ubuntu 20.04 version recommended):** Download the tarball from the official archives or mirrors.
2. **Extract and Set Environment Variables:** Add the following to your `~/.bashrc` or run in your terminal before installation.

```
export COPPELIASIM_ROOT=/path/to/CoppeliaSim_Edu_V4_1_0_Ubuntu20_04
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$COPPELIASIM_ROOT
export QT_QPA_PLATFORM_PLUGIN_PATH=$COPPELIASIM_ROOT
```

3. **Install PyRep (Python bindings):**

```
git clone https://github.com/stepjam/PyRep.git
cd PyRep
# Create environment for RLBench
uv venv rlbench_env --python 3.10
source rlbench_env/bin/activate

# Install requirements
uv pip install -r requirements.txt
uv pip install .
```

4. **Install RLBench:**

```
cd ..
git clone https://github.com/stepjam/RLBench.git
cd RLBench
uv pip install -r requirements.txt
uv pip install .
```

Verification

```
from rlbench.environment import Environment
from rlbench.action_modes import ArmActionMode, ActionMode
from rlbench.observation_config import ObservationConfig
from rlbench.tasks import ReachTarget

# Create environment configuration
action_mode = ActionMode(ArmActionMode.ABS_JOINT_VELOCITY)
env = Environment(action_mode, obs_config=ObservationConfig(), headless=False)
env.launch()

task = env.get_task(ReachTarget)
task.reset()

print("RLBench launched successfully. Check for the simulation window.")
env.shutdown()
```

5 MuJoCo Playground (mjlab)

Description: A modern framework combining Isaac Lab's API with MuJoCo physics, designed for high-performance RL and robotics research.
Repository: https://github.com/mujocolab/mjlab

Installation

`mjlab` is designed to be installed with `uv`. **A GPU is heavily recommended** as the framework leverages hardware acceleration for physics and rendering.

1. **Install uv (if not installed):**

```
pip install uv
```

2. **Clone and Install:**

```
git clone https://github.com/mujocolab/mjlab.git
cd mjlab
# Install dependencies and the package in editable mode
uv sync
```

3. **Note on Virtual Environment:** `uv sync` automatically creates and manages a virtual environment ('.venv') inside the project directory. You can activate it with:

```
source .venv/bin/activate
```

Verification

Run the provided demo script to ensure the viewer and physics engine are working.

```
# From inside the mjlab directory
uv run demo
```

This command should launch an interactive viewer with a pre-trained agent (e.g., Unitree G1) tracking a reference motion.