ES/AM 158 Upkie Lab

November 2025

Introduction

This lab is designed for students to practice reinforcement learning (RL) on a real-robot application. You will use the Bullet-based **Spine** simulator and **Upkie** for policy optimization. Bullet is a high-performance physics engine widely used for model-based control and RL; Upkie is an open-source wheeled-biped platform with a convenient simulation and deployment tool-chain.

Submission Please submit a single compressed file containing **one short report** and **two videos** of your rollout trajectory. You may roll out multiple trajectories and include them in a single video. The submission deadline is **11:59 p.m., November 23**. Refer to the README for example videos.

Task 1 (10 pts): complete a Gym environment. Task 2 (20 pts): train a stabilizing policy on it. Task 3 (20 pts, bonus): replace the simplified dynamics with the full dynamics and retrain a stabilizing policy.

Upkie (`https://github.com/upkie`) Upkies are open-source wheeled biped robots. They use wheels for balancing and legs to negotiate uneven terrain. Upkies are designed to be built with off-the-shelf tools and components (e.g., mjbots actuators). You can develop in Python or C++ on Linux or macOS, then deploy your behaviors to the robot's Raspberry Pi.

The Upkie simulation is available through several backends (Spine, PyBullet, Genesis) and is wrapped as a Gymnasium-style environment. This means you can reuse the same structure from your problem sets to train on this environment. You are allowed to use third-party packages such as *Stable-Baselines3*.

Installation

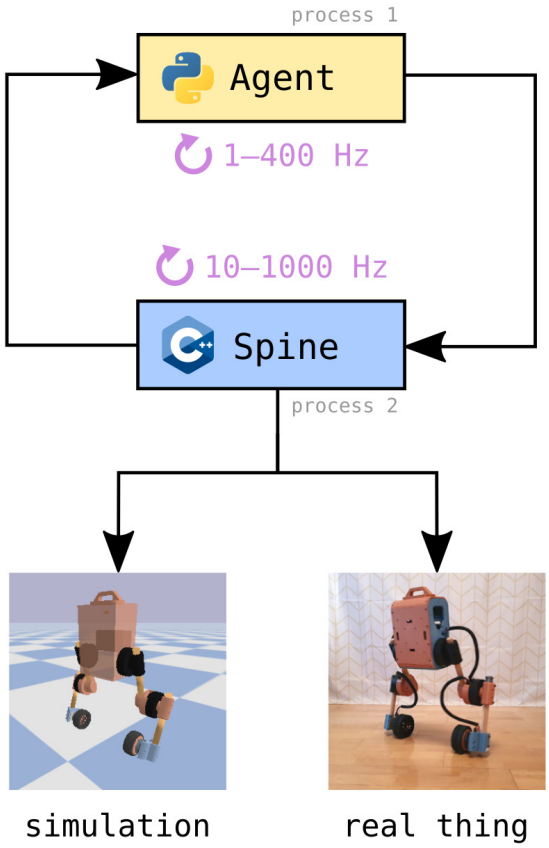
This setup is **not** supported on Google Colab. We recommend using VS Code (or your preferred editor) on a local machine.

Create environment and install Upkie

```
conda create -n upkie python=3.10
conda activate upkie
pip install upkie
```

observation dictionary


```
{
  "imu": {
    "linear_acceleration": [
      0.10456651449203491,
      0.1155887097120285,
      -0.19683143496513367
    ],
    "angular_velocity": [
      -0.018338500218555813,
      -0.0048945160966859975,
      0.013782314291377813
    ],
  },
  "servo": {
    "left_hip": {
      "voltage": 19,
      "velocity": -0.0188,
      "torque": -0.73,
      "q_current": -1.90,
      "mode": 10,
      "temperature": 22,
      "position": -0.637,
      "fault": 0,
      "d_current": -0.2
    },
    "right_hip": {
      "voltage": 18.5,
      "velocity": 0.0486,
      "torque": 0.62,
      "q_current": 1.8,
      "mode": 10,
      "temperature": 22,
      "position": 0.638,
      "fault": 0,
      ...
```

action dictionary


```
{
  "servo": {
    "left_hip": {
      "position": -0.6597,
      "velocity": 6.0939e-06,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    "left_knee": {
      "position": 1.0698,
      "velocity": -2.5212e-05,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    "left_wheel": {
      "position": null,
      "velocity": -2.6646
    },
    "right_wheel": {
      "position": null,
      "velocity": 2.66464
    }
    "right_hip": {
      "position": 0.658,
      "velocity": -5.149,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    "right_knee": {
      "position": -1.070,
      "velocity": 2.1e-05,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    ...
```


Figure 1: The Upkie repository uses the Spine simulator as its primary backend and communicates with it at 200 Hz. The servo environment applies actions to each joint and wheel by specifying position and velocity commands as well as PID parameters.

Launch the simulator

From the Upkie workspace, run:

```
./start_simulation.sh
```

You should see the Upkie simulator window. You can interact with the robot using the mouse (e.g., apply external forces).

Minimal roll-out

Open another terminal (with the same Conda environment activated) and run:

```
python rollout_policy.py
```

This executes a minimal roll-out against the simulated environment. Replace the model name with your own policy file as needed.

1 Task1: Finish environment (10 pt)

All simulation is handled by the Spine backend, so you will not find physics code in this repository. However, the Upkie repository provides a wrapper environment around the simulator. The underlying simulator accepts position/velocity commands and PID parameters for every joint and wheel. To simplify, Upkie also provides a CartPole-like model:

The Upkie ***pendulum*** wrapper makes the robot behave as a wheeled inverted pendulum: the legs are kept straight and only the wheel velocities are actuated. The action is the commanded ground velocity $a = [\dot{p}^*]$ (m/s), internally mapped to wheel speed commands (a practical range is $[-1, 1]$ m/s). The observation is $o = [\,\theta,\ p,\ \dot{\theta},\ \dot{p}\,]$, where θ is the base pitch (rad), p the average wheel contact position (m), and the dotted terms are the corresponding angular and linear velocities (rad/s and m/s). Observation and action spaces are not normalized; full Spine observations remain available in the `info` dictionary returned by `reset` and `step`.

Currently, the environment returns a constant reward and truncates after 300 steps. Your first task is to design termination conditions and/or modify the reward so that a policy can learn to stabilize Upkie. A reference script, ***rollout_policy.py***, is provided; after you implement the environment, roll-out to visualize the expected termination. In this task you only need to modify the environment—training code will be written in the next task.

TODO: Finish upkie/envs/upkie_pendulum.py Write a description of your environment in report.

2 Task 2: Train a Stabilizing Policy (20 pt)

You now have a complete Gymnasium environment. Your goal in Task 2 is to train a policy—using any method you prefer—to stabilize Upkie in the upright position. An interface for constructing the Upkie environment is provided in ***rollout_policy.py***.

TODO: Train a policy and test it using *rollout_policy.py* (or your own code). Write a short report describing your approach and results.

3 Task 3: Train a Stabilizing Policy on the Full Model (20 pt, BONUS)

We now move on to a more challenging model. Upkie is controlled by six motors, and you can command position, velocity, and PID gain scales for each joint.

Upkie–Servos. This environment exposes direct moteus-style control of six servos {`left_hip, left_knee, left_wheel, right_hip, right_knee, right_wheel`}. The *action* is a dictionary keyed by servo, with fields `position` θ^* (rad; set `NaN` to disable the position loop), `velocity` $\dot{\theta}^*$ (rad/s), `feedforward_torque` τ_{ff} (N·m), `kp_scale`, `kd_scale` (both $\in [0, 1]$), and `maximum_torque` $\tau_{\max}$ (N·m). The servo applies torque

$$\tau = \text{clamp}_{[-\tau_{\max}, \tau_{\max}]}\big(\tau_{\text{ff}} + k_p\, k_p^{\text{scale}}(\theta^* - \theta) + k_d\, k_d^{\text{scale}}(\dot{\theta}^* - \dot{\theta})\big),$$

with fixed controller gains k_p, k_d inside moteus (its inner loop runs at $\sim 40\,\text{kHz}$, faster than the agent–Spine loop). The *observation* is a per-servo dictionary with `position` (rad), `velocity` (rad/s), `torque` (N·m), `temperature` (°C), and `voltage` (V). The full backend observation dictionary is also returned via `info` for custom reward shaping and termination logic.

We provide an example ***rollout_policy_servos.py*** that builds a wrapped environment compatible with Stable-Baselines3 PPO. For more information, see Upkie's official repository: `https://github.com/upkie`.

TODO: Implement the environment and a policy, then test them using ***rollout_policy_servos.py*** (or your own code). You are not expected to achieve excellent performance—training can be tricky—but you should show that training improves the policy. Note that Upkie may exhibit diverse patterns for stabilizing itself. Write a brief report of your design and results.