AI Enabled Control Engineering

Lecture 11: Sim-to-Real Deployment of Reinforcement Learning Controllers

Professor Xun Huang

TA: Zhixiang Ju Haozhe Wang

Aeronautics and Astronautics
College of Engineering
Peking University

huangxun@pku.edu.cn
https://xunger99.github.io/xunger/

Recap: from Class 11 to Class 12

- In Class 11, we trained DQN, PPO, and TD3 controllers in the RIP simulation environment.
- We inspected training curves, simulation rollouts, hyperparameters, and algorithm differences.
- Today we deploy trained policies to the real STM32-based rotary inverted pendulum.
- The main question is not only whether the controller works, but why sim-to-real performance changes.

Lecture 12 roadmap

1. Sim-to-real deployment workflow.
2. Deploy DQN, PPO, and TD3 controllers to the real RIP.
3. Record real-system logs and compare them with simulation rollouts.
4. Analyze sim-to-real mismatch and write the deployment report.

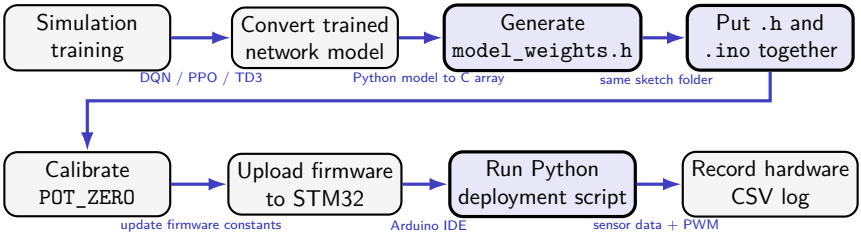
Class 12 goal

Main goal

Deploy the policies trained in simulation to the physical rotary inverted pendulum and evaluate their real-system behavior.

- prepare model header files and upload STM32 firmware;
- calibrate the potentiometer zero value before deployment;
- compare DQN, PPO, and TD3 in simulation and on real hardware;
- explain model mismatch, actuator nonlinearity, friction, sensor noise, and delay.

Sim-to-real deployment pipeline



Key deployment rule: Place `model_weights.h` in the same folder as the STM32 `.ino` firmware. Then calibrate `POT_ZERO`, upload the firmware, run the Python script, and save the hardware log.

Step 1: potentiometer zero calibration

1. Upload the raw sensor test firmware or use the deployment firmware serial output.
2. Hold the pendulum exactly near the upright position and keep it still.
3. Read the current potentiometer raw value `pot` from the serial monitor or log.
4. Set this value as `POT_ZERO` in the firmware.
5. Re-upload the firmware after modifying `POT_ZERO`.

Step 2: put model headers beside firmware

```
DQN sketch folder:
  deploy_dqn.ino
  model_weights.h

PPO sketch folder:
  deploy_ppo.ino
  ppo_model_weights.h
  dqn_model_weights.h

TD3 sketch folder:
  deploy_td3.ino
  td3_model_weights.h
```

Common mistake

If the header file is missing or has a mismatched format, Arduino IDE will fail during compilation. Always confirm that the header name matches the `#include` line in the firmware.

Step 3: upload firmware and close Serial Monitor

1. Open the corresponding `deploy_*.ino` file in Arduino IDE.
2. Select the correct STM32 board and serial port.
3. Upload the firmware.
4. Close Arduino Serial Monitor after checking startup information.
5. Run the Python deployment script from a terminal.

Reason

Only one program can occupy the serial port. If Arduino Serial Monitor is still open, the Python logger cannot connect to STM32.

Step 4: run the PC deployment logger

```
# DQN
python deploy_dqn.py

# PPO
python deploy_ppo.py

# TD3
python deploy_td3.py
```

The script will:

- find or open the serial port;
- wait for the student to press Enter;
- send `GO` to STM32;
- record all `LOG` lines into a CSV file;
- send `S` when the experiment stops or falls.

Deployment start modes

Algorithm	Policy type	Student operation
DQN	swing-up + stabilization	start from natural hanging-down state
TD3	swing-up + stabilization	start from natural hanging-down state
PPO	upright stabilization only	hold near upright, press Enter, and release

Important start condition

The DQN and TD3 policies trained in simulation include the swing-up stage, so they can be tested from the natural hanging-down position. The PPO policy used in this class is a balance-only policy, so students must gently hold the pendulum near the upright position, press Enter in the Python terminal, and release at the same time.

Before pressing Enter

Before sending the `GO` command from Python, check:

- the correct model header has been compiled into the firmware;
- `POT_ZERO` has been updated for the current device;
- motor power is connected only after wiring is checked;
- the rotating arm has enough free space;
- the emergency stop command is ready in the Python terminal.

Safety rule

Keep hands away from the rotating arm after pressing Enter. If the pendulum falls or the motor behaves abnormally, stop the experiment immediately.

What should be recorded?

The Python logger should save one CSV file for each hardware test.

A useful deployment log should include:

$$t, \quad \theta, \quad \alpha, \quad \dot{\theta}, \quad \dot{\alpha}, \quad u, \quad \text{mode}.$$

Recommended additional fields:

$$\text{pot\_raw}, \quad \text{enc}, \quad \text{action index}, \quad \text{done flag}.$$

Why log mode?

The mode field tells us whether the controller is in swing-up, stabilization, fallback, or safety-stop mode. It is important for diagnosing failed deployment.

DQN deployment: what to check

DQN uses a discrete action set.

During deployment, check:

- whether the learned policy can swing the pendulum up from the natural hanging-down state;
- whether the action sequence is reasonable rather than random-like switching;
- whether the pendulum can enter and remain near the upright region;
- whether PWM saturation appears too often.

Typical failure

If the real pendulum cannot swing up although simulation works, possible reasons include insufficient motor torque, wrong action scaling, friction mismatch, or wrong angle calibration.

PPO deployment: what to check

In this class, the PPO policy is used as an upright stabilization policy.

The correct start procedure is:

1. hold the pendulum near the upright position;
2. run the Python deployment script;
3. press Enter to send `GO`;
4. release the pendulum gently at the same time.

Evaluation

A good PPO deployment should keep α close to zero after release and avoid high-frequency PWM oscillation.

TD3 deployment: what to check

TD3 uses a continuous actor output.

During deployment, check:

- whether the continuous action is correctly mapped to PWM;
- whether the actor output is saturated too frequently;
- whether the pendulum can swing up and stabilize in the real system;
- whether the real response is smoother or more unstable than DQN.

Important

A continuous actor may generate smoother commands in simulation, but real motor dead zone and friction can strongly change its behavior.

Simulation-to-hardware comparison

For each algorithm, compare simulation and hardware results.

Item	Simulation	Hardware
Initial condition	controlled setting	real manual / natural start
Sensor signal	clean or simulated noise	real noise and drift
Motor response	ideal mapping	dead zone and saturation
Time step	fixed simulation step	real sampling jitter possible
Control curve	expected rollout	measured CSV log

Goal

The goal is not only to show whether the controller works, but also to explain why simulation and hardware differ.

Common sim-to-real mismatch sources

- **Angle calibration error:** wrong `POT_ZERO` shifts the definition of $\alpha = 0$.
- **Motor dead zone:** small PWM commands may not move the motor.
- **Friction and backlash:** real mechanical resistance is not perfectly modeled.

- **Sensor noise:** angle difference can amplify noise in velocity estimation.
- **Delay and timing jitter:** serial communication and real-time loop timing may differ from simulation.

How to diagnose a failed deployment

Symptom	Possible reason
Motor does not move	motor power off, wrong PWM mapping, motor dead zone
Moves in wrong direction	sign error in angle, encoder, or PWM command
Falls immediately near upright	wrong `POT_ZERO`, policy not trained for this state region
Strong oscillation	excessive gain, noisy velocity estimate, delay, unstable policy
Works in simulation only	model mismatch, friction, saturation, unmodeled delay

Do not change many parameters at the same time. Change one factor, record the result, and compare the log.

Report requirements

Each group should include:

- deployment procedure for DQN, PPO, and TD3;
- model header file name and firmware file name;
- calibrated `POT_ZERO` value;
- simulation rollout plots for the deployed models;
- hardware deployment plots and CSV logs;
- comparison of DQN, PPO, and TD3 real-system performance;
- sim-to-real mismatch analysis.

Main focus

The report should connect the simulation training result with the real hardware behavior.

Suggested quantitative metrics

For DQN and TD3, record the swing-up performance:

$$T_{\text{swing}}, \qquad T_{\text{upright}}, \qquad N_{\text{fall}}.$$

For all deployed algorithms, evaluate the stabilization stage:

$$\text{mean}(|\alpha|), \qquad \text{std}(\alpha), \qquad \text{mean}(|\theta|), \qquad \text{std}(\theta).$$

Also report PWM usage:

$$\text{mean}(|u|), \qquad \text{max}(|u|), \qquad \text{saturation ratio}.$$

DQN and TD3 include swing-up, so T_{swing} should be recorded. PPO is used as an upright stabilization policy in this class, so only the stabilization-stage accuracy and PWM usage are evaluated.

Required plots for deployment analysis

For each algorithm, plot the following time histories:

$$\theta(t), \qquad \alpha(t), \qquad \dot{\theta}(t), \qquad \dot{\alpha}(t).$$

Also plot the PWM command:

$$u(t).$$

For PWM statistics, include at least one summary figure:

PWM histogram or bar chart of $\mathrm{mean}(|u|)$, $\max(|u|)$, saturation ratio.

Comparison requirement

Use the same angle convention, time unit, and PWM scale when comparing DQN, PPO, and TD3. For DQN and TD3, mark the swing-up stage and the stabilization stage on the plot if possible.

Suggested report structure

1. Brief description of deployed models and firmware.
2. Sensor calibration and hardware start conditions.
3. DQN deployment result and analysis.
4. PPO deployment result and analysis.
5. TD3 deployment result and analysis.
6. Cross-algorithm comparison.
7. Sim-to-real mismatch discussion.
8. Possible improvements for the next class.

Summary of Class 12

- We moved from simulation-trained policies to real STM32 deployment.
- We learned the workflow: model weights, firmware, sensor calibration, Python logger, and hardware CSV logs.
- DQN and TD3 are tested as swing-up plus stabilization policies.
- PPO is tested as a balance-only policy near the upright region.
- The most important analysis is the difference between simulation behavior and real hardware behavior.

Take-home message

Successful AI-enabled control requires not only training a policy, but also deploying it safely, recording real data, and explaining the sim-to-real gap.