AI Enabled Control Engineering

Lecture 13: Hybrid Control, Sim-to-Real Improvement, and Final Demonstration

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/

Lecture 13 roadmap

1. Why hybrid control is needed for the rotary inverted pendulum.
2. Required hybrid schemes: DQN swing-up + MPC stabilization, and DQN swing-up + PPO stabilization.
3. Sim-to-real gap and residual-network correction idea.
4. Student design space: switching logic, residual usage, and safety rules.
5. Live demonstration assessment.

Class 13 goal

Students should design, test, compare, and demonstrate a hybrid control strategy for the real rotary inverted pendulum, and try to use a residual network to reduce the sim-to-real gap.

Why hybrid control?

A single controller is usually not ideal for the whole RIP task.

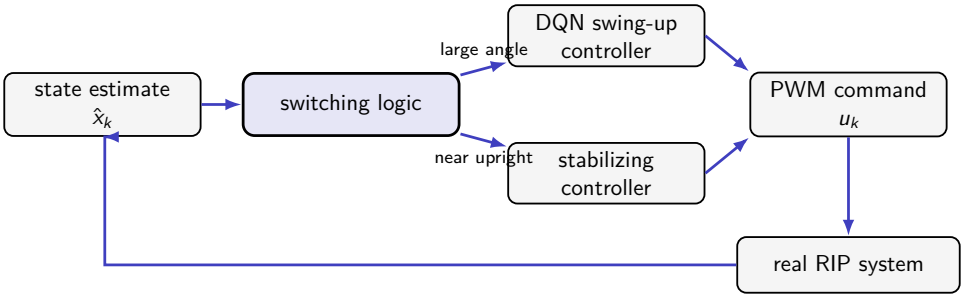
- Swing-up is a large-angle nonlinear task.
- Upright stabilization is a local precision-control task.
- DQN is suitable for learning a large-region discrete-action policy.
- MPC is suitable for constrained local stabilization.
- PPO may work well as a learned balancing controller near upright.

Main idea

Use different controllers in different regions of the state space.

$$\text{large angle: swing-up policy} \quad \Longrightarrow \quad \text{near upright: stabilizing controller}$$

Hybrid control architecture



Key design problem

The switching logic should avoid chattering, unsafe mode changes, and unstable handover between controllers.

Required hybrid scheme 1: DQN + MPC

Required experiment

Use DQN for swing-up and switch to MPC when the pendulum enters the upright region.

- DQN handles large-angle nonlinear motion.
- MPC handles local stabilization with PWM constraints.
- Students should design the switching threshold and hysteresis.
- The report should compare this hybrid controller with pure DQN deployment.

Example switching condition:

$$|\alpha_k| < \alpha_{\text{enter}}, \qquad |\dot{\alpha}_k| < \dot{\alpha}_{\text{enter}}.$$

Example safety fallback:

$$|\alpha_k| > \alpha_{\text{exit}} \quad \Rightarrow \quad \text{return to DQN swing-up or stop safely.}$$

Required hybrid scheme 2: DQN + PPO

Required experiment

Use DQN for swing-up and switch to PPO when the pendulum is close to upright.

- DQN provides the global swing-up behavior.
- PPO provides the learned local balancing policy.
- The PPO controller should only be activated inside its reliable region.
- Students should compare whether PPO or MPC gives better steady-state behavior.

Important

If the stabilizing controller does not include swing-up, students must not start it from the hanging-down position. The DQN part is responsible for bringing the pendulum close to upright.

Switching logic: avoid chattering

A naive switch may repeatedly jump between two controllers.

Use hysteresis:

$$\alpha_{\text{enter}} < \alpha_{\text{exit}}.$$

For example:

$$|\alpha| < 15^\circ \quad \Rightarrow \quad \text{enter stabilization mode,}$$

$$|\alpha| > 25^\circ \quad \Rightarrow \quad \text{leave stabilization mode.}$$

Also require a dwell time:

$$|\alpha_k| < \alpha_{\text{enter}} \quad \text{for } N_{\text{hold}} \text{ consecutive steps.}$$

Example switching pseudocode

```
if mode == DQN_SWINGUP:
    u = dqn_policy(state)

    if abs(alpha) < alpha_enter and abs(dalpha) < dalpha_enter:
        stable_counter += 1
    else:
        stable_counter = 0

    if stable_counter >= N_hold:
        mode = STABILIZE
        reset_stabilizer_if_needed()

elif mode == STABILIZE:
    u = stabilizer_policy(state) # MPC or PPO

    if abs(alpha) > alpha_exit:
        mode = DQN_SWINGUP
        stable_counter = 0

u = saturate(u, -PWM_MAX, PWM_MAX)
```

Safety

If $|\alpha|$, $|\dot{\alpha}|$, or $|\theta|$ exceeds the safety limit, stop the motor immediately.

Sim-to-real gap

A policy trained in simulation may fail on the real system.

Common reasons:

- inaccurate friction and motor dead-zone model;
- mismatch between simulated PWM and real motor torque;
- sensor noise and angle calibration error;
- delay and nonuniform sampling;
- unmodeled mechanical vibration and loose wires.

Residual network idea

The model predicts:

$$x_{k+1}^{\text{model}} = F_{\text{model}}(x_k, u_k).$$

The real system gives:

$$x_{k+1}^{\text{real}}.$$

Define a residual:

$$r_k = x_{k+1}^{\text{real}} - x_{k+1}^{\text{model}}.$$

Train a residual network:

$$\hat{r}_k = R_\phi(x_k, u_k).$$

Then the corrected prediction becomes:

$$x_{k+1}^{\text{corr}} = F_{\text{model}}(x_k, u_k) + R_\phi(x_k, u_k).$$

How can the residual network be used?

Students may freely design how to use the residual network.
Possible choices:

- improve MPC prediction:

$$x_{k+1} = F_{\text{model}}(x_k, u_k) + R_\phi(x_k, u_k);$$

- correct the training simulation environment;
- generate an additional compensation term:

$$u_k = u_{\text{base}} + \Delta u_\phi;$$

- use it only for offline sim-to-real analysis.

Example 1: residual network for one-step MPC prediction

The nominal discrete model used by linear MPC is

$$x_{k+1}^{\text{nom}} = A_d x_k + B_d u_k.$$

The real hardware transition is approximated by the next estimated state:

$$x_{k+1}^{\text{real}} \approx \hat{x}_{k+1}.$$

Define the one-step model residual:

$$r_k = x_{k+1}^{\text{real}} - x_{k+1}^{\text{nom}}.$$

Train a residual network:

$$\hat{r}_k = R_\phi(\xi_k).$$

Then the corrected prediction becomes

$$x_{k+1}^{\text{corr}} = A_d x_k + B_d u_k + R_\phi(\xi_k).$$

Residual network input and output

Use a short history window as the network input.
For example, with history length $L = 6$,

$$\xi_k = [\hat{x}_{k-L+1}, u_{k-L+1}, \hat{x}_{k-L+2}, u_{k-L+2}, \ldots, \hat{x}_k, u_k].$$

Each step contains

$$\hat{x}_k = \begin{bmatrix} \hat{\theta}_k & \hat{\dot{\theta}}_k & \hat{\alpha}_k & \hat{\dot{\alpha}}_k \end{bmatrix}^\top, \qquad u_k.$$

Therefore, the input dimension is

$$L \times (4 + 1) = 6 \times 5 = 30.$$

The output is the four-dimensional one-step residual:

$$R_\phi(\xi_k) = \begin{bmatrix} d\theta_{\mathrm{res},k} \\ d\dot{\theta}_{\mathrm{res},k} \\ d\alpha_{\mathrm{res},k} \\ d\dot{\alpha}_{\mathrm{res},k} \end{bmatrix}.$$

Residual network improvement example

The following table gives one example of validation-set improvement for one-step prediction error.

$\lvert\alpha\rvert$ range	samples	θ	$\dot{\theta}$	α	$\dot{\alpha}$
0°–2°	25946	39.8%	23.1%	16.4%	16.3%
2°–4°	11741	41.0%	25.6%	19.2%	19.2%
4°–6°	3740	41.3%	28.2%	22.7%	23.2%
6°–8°	1065	46.8%	32.1%	26.5%	27.1%
8°–10°	246	46.6%	35.6%	32.0%	33.3%
All samples	–	40.6%	24.8%	18.4%	18.3%

Observation

The residual network reduces the nominal one-step prediction error. The improvement becomes more significant when $|\alpha|$ is larger, where the linear model mismatch is stronger.

How the residual enters MPC prediction

Without residual correction:

$$x_{k+1} = A_d x_k + B_d u_k.$$

With residual correction:

$$x_{k+1} = A_d x_k + B_d u_k + r_k.$$

For a two-step prediction:

$$x_{k+2} = A_d(A_d x_k + B_d u_k + r_k) + B_d u_{k+1},$$

that is,

$$x_{k+2} = A_d^2 x_k + A_d B_d u_k + B_d u_{k+1} + A_d r_k.$$

Thus the residual affects the future prediction sequence as

$$r_k, \qquad A_d r_k, \qquad A_d^2 r_k, \qquad \cdots .$$

Why only one-step residual correction?

Theoretically, we may correct every predicted step:

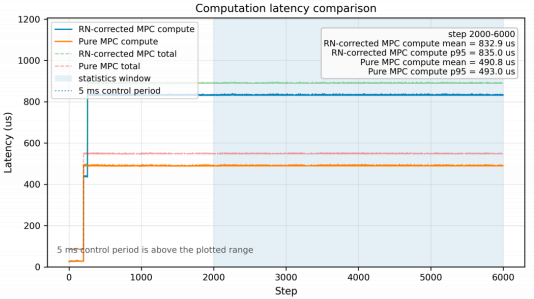
$$x_{k+i+1} = A_d x_{k+i} + B_d u_{k+i} + R_\phi(\xi_{k+i}).$$

However, each residual-network inference adds computation latency.

In our real-time controller,

$$T_s = 5\,\mathrm{ms}, \qquad f_s = 200\,\mathrm{Hz}.$$

If the residual network is evaluated many times inside the MPC horizon, the controller may exceed the $5\,\mathrm{ms}$ control period.



Latency observation

Residual-corrected MPC increases the computation time by several hundred microseconds compared with pure MPC. This is still below $5\,\mathrm{ms}$, but it reduces the real-time safety margin.

What does the residual network learn?

The residual contains several unmodeled effects:

- friction and backlash;
- PWM-to-torque mapping error;
- parameter identification error;
- sensor noise and delay;
- local nonlinear dynamics not captured by the linearized model.

The training target is

$$r_k = x_{k+1}^{\text{real}} - x_{k+1}^{\text{nom}}.$$

Important distinction

r_k is a one-step prediction residual, not the current state error. In MPC, it should enter future prediction as an additive model correction.

Example 2: offline model-reference adaptation idea

A related idea is model-reference adaptation.
Instead of directly correcting the next state prediction, it tries to make the real system behave like a chosen reference model.
Reference model:

$$x_{m,k+1} = f_m(x_{m,k}) + g_m(x_{m,k})u_k.$$

Real system with an input correction:

$$u_{a,k} = u_k + \delta u_k.$$

A neural network learns

$$\delta u_k = T_\phi(x_{a,k}, x_{m,k}, u_k),$$

so that the real system output follows the reference model output.

Offline version for our lab

Students can train T_ϕ offline from hardware logs, then use it as a fixed compensation module during later experiments.

Reference from LipNet-MRAC

The attached paper proposes a learning-based model-reference adaptive control method.

Its key ideas are:

- use a neural network to adapt the input command;
- make the real robot response close to a predefined reference model;
- exploit a Lipschitz network to provide a stability-related condition;
- demonstrate the method on a flying inverted pendulum experiment.

How we simplify it in this class

We do not require online LipNet adaptation. Students may borrow the idea and train an offline residual or compensation network from logs. The main grading focus remains the hybrid controller.

Required experiments

1. DQN swing-up + MPC stabilization.
2. DQN swing-up + PPO stabilization.
3. Optional bonus schemes:
   - DQN + LQR;
   - DQN + residual-corrected MPC;
   - pure RL policy with improved sim-to-real tuning;
   - any safe hybrid strategy with better performance.

Minimum requirement

Every group must complete and compare the two required hybrid schemes. Additional successful schemes can be used for bonus discussion and live demonstration.

Report requirements

- Describe the switching logic and thresholds.
- Present DQN+MPC hardware control curves.
- Present DQN+PPO hardware control curves.
- Compare $\alpha(t)$, $\theta(t)$, PWM, mode switching time, and stabilization duration.
- Analyze sim-to-real mismatch.
- Discuss whether residual-network correction helps, if used.

Main grading focus

The report mainly evaluates hybrid control design, experimental comparison, and sim-to-real analysis.

Final live demonstration

At the end of the lab, each group will demonstrate its best controller on the real RIP system.

- Each group selects one best-performing scheme.
- The controller should be started under the correct condition.
- The group should briefly explain the controller structure and switching rule.
- The experiment will be judged by actual control performance and safety.

Live assessment

The live demonstration is the closing assessment of the RIP experiment series. A successful demonstration should show stable control and clear understanding of the hybrid strategy.

Final demo scoring guideline

Item	Criterion
Safety	Correct start procedure, no unsafe motor behavior, emergency stop ready.
Hybrid logic	Clear mode switching, no severe chattering, reasonable thresholds.
Control effect	Successful swing-up and stable upright behavior.
Data quality	Logs and plots are complete, including mode and PWM.
Explanation	Group can explain why the method works or fails.
Bonus	Residual correction or other creative hybrid strategy improves performance.

Summary of the final lab

- Hybrid control combines the strengths of different controllers.
- DQN can be used for large-angle swing-up.
- MPC or PPO can be used for upright stabilization.
- Residual networks provide one possible way to reduce sim-to-real mismatch.
- The final goal is not only to run code, but to design, test, compare, and explain a complete control strategy.

Take-home message

AI-enabled control is an engineering loop: modeling, simulation, learning, deployment, logging, diagnosis, and redesign.