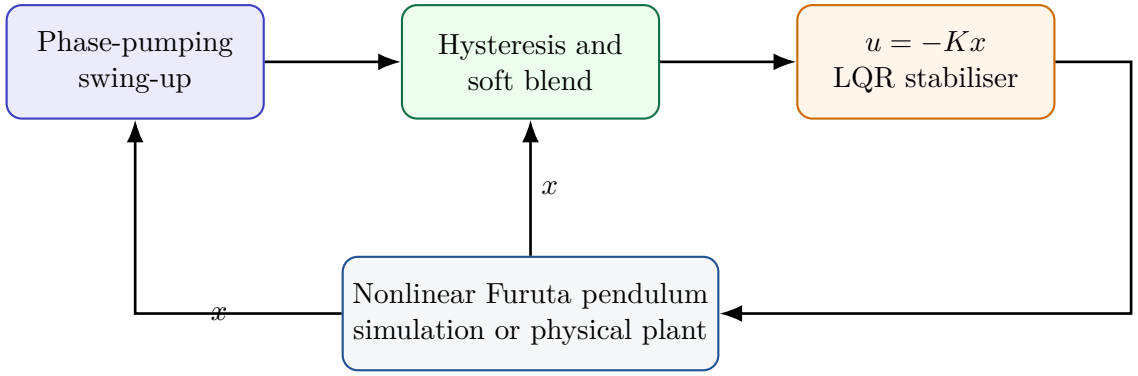
LQR Control of a Rotary Inverted Pendulum

Class 9 Laboratory Technical Manual

AI Enabled Control Engineering



Class-9 design objective. Keep the Class-8 nonlinear plant, 200 Hz timing, swing-up controller, switching thresholds, soft blend, noise model and visual interface unchanged. Replace only the upright dual-PID stabiliser by an LQR full-state feedback controller. This makes the PID–LQR comparison meaningful.

Course staff: Professor Xun Huang; TAs: Zhixiang Ju and Haozhe Wang
Version: July 2026

Contents

1 Full-State Feedback and Its Relation to Dual PID

1.1 State definition and control objective

The state vector used in all Class-9 code is

$$\boldsymbol{x} = \begin{bmatrix} \theta & \dot{\theta} & \alpha & \dot{\alpha} \end{bmatrix}^T,$$

where θ is the rotary-arm angle and $\alpha = 0$ is the upright pendulum position. The local stabilisation objective is to drive all four states toward zero while respecting the PWM saturation limit.

A full-state feedback controller has the form

$$u = -K\boldsymbol{x}, \qquad K = \begin{bmatrix} K_\theta & K_{\dot{\theta}} & K_\alpha & K_{\dot{\alpha}} \end{bmatrix}.$$

The four entries specify how strongly the control input responds to each state. The minus sign is part of the standard feedback law and is applied internally by all Class-9 programs.

IMPORTANT

Enter the entries of K exactly as produced by the Riccati calculation. Do not manually change their signs. The input convention of this RIP model may produce negative entries in K; the software still applies $u = -Kx$.

1.2 Why the Class-8 controller is already full-state feedback when $K_i = 0$

During the upright stage, the Class-8 dual-loop PID command was

$$\begin{aligned} u_{\text{PID}} = {} & K_{p,\alpha}\alpha + K_{i,\alpha}\int \alpha\, dt + K_{d,\alpha}\dot{\alpha} \\ & + K_{p,\theta}\theta + K_{i,\theta}\int \theta\, dt + K_{d,\theta}\dot{\theta}. \end{aligned}$$

If both integral gains are zero,

$$u_{\text{PID}} = K_{p,\theta}\theta + K_{d,\theta}\dot{\theta} + K_{p,\alpha}\alpha + K_{d,\alpha}\dot{\alpha}.$$

Comparing this expression with $u = -Kx$ gives

$$K = \begin{bmatrix} -K_{p,\theta} & -K_{d,\theta} & -K_{p,\alpha} & -K_{d,\alpha} \end{bmatrix}.$$

Therefore, the Class-8 stabiliser with $K_i = 0$ is mathematically a full-state feedback controller. However, it is not automatically an LQR controller.

Important distinction. LQR is a systematic method for selecting a full-state feedback gain by minimising a quadratic performance index. Every LQR controller is full-state feedback, but an arbitrary full-state feedback controller is not necessarily LQR-optimal.

1.3 Why LQR remains a local controller

The gain is designed from a model linearised around $\alpha = 0$. Far from the upright equilibrium, neglected nonlinear terms become important. Therefore, LQR alone cannot reliably lift the pendulum from the hanging-down position. Class 9 retains the Class-8 phase-pumping swing-up stage and uses LQR only after $|\alpha|$ enters the upright neighbourhood.

2 Linear Quadratic Regulator Theory

2.1 Continuous-time infinite-horizon LQR

Consider the continuous-time linear system

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + Bu(t),$$

with the infinite-horizon quadratic cost

$$J_c(\boldsymbol{x}_0, u) = \int_0^\infty \left[\boldsymbol{x}^T(t)Q\boldsymbol{x}(t) + u^T(t)Ru(t)\right] dt,$$

where

$$Q = Q^T \succeq 0, \qquad R = R^T \succ 0.$$

The objective is to determine a state-feedback control law that stabilises the system while minimising both the state deviation and the control effort.

Under the standard assumptions that (A, B) is stabilisable and $(Q^{1/2}, A)$ is detectable, the continuous algebraic Riccati equation has a unique stabilising solution $P = P^T \succeq 0$.

2.1.1 Derivation from the HJB equation

The Hamilton–Jacobi–Bellman (HJB) equation was introduced in Professor Huang's lectures as the continuous-time differential form of Bellman's principle of optimality. It states that an optimal decision must minimise the sum of the current instantaneous cost and the rate of change of the optimal future cost.

Define the optimal cost-to-go, or value function, as

$$V^*(\boldsymbol{x}) = \inf_{u(\cdot)} \int_0^\infty \left[\boldsymbol{x}^T(t)Q\boldsymbol{x}(t) + u^T(t)Ru(t)\right] dt,$$

subject to the initial condition

$$\boldsymbol{x}(0) = \boldsymbol{x}.$$

Because the system, the cost function and the integration interval are time-invariant, the infinite-horizon value function does not explicitly depend on time. Its stationary HJB equation is therefore

$$0 = \min_u \left\{\boldsymbol{x}^T Q\boldsymbol{x} + u^T Ru + \nabla_{\boldsymbol{x}} V^*(\boldsymbol{x})^T \left(A\boldsymbol{x} + Bu\right)\right\}.$$

For a linear system with a quadratic cost, assume that the optimal value function has the quadratic form

$$V^*(\boldsymbol{x}) = \boldsymbol{x}^T P\boldsymbol{x}, \qquad P = P^T \succeq 0.$$

Its gradient is

$$\nabla_{\boldsymbol{x}} V^*(\boldsymbol{x}) = 2P\boldsymbol{x}.$$

Substituting this expression into the HJB equation gives

$$0 = \min_u \left\{\boldsymbol{x}^T Q\boldsymbol{x} + u^T Ru + 2\boldsymbol{x}^T P\left(A\boldsymbol{x} + Bu\right)\right\}.$$

For a fixed state $\boldsymbol{x}$, the expression inside the braces is a strictly convex quadratic function of u because $R \succ 0$. Therefore, its unique minimum is obtained by setting its derivative with respect to u equal to zero:

$$\frac{\partial}{\partial u}\left[\boldsymbol{x}^T Q\boldsymbol{x} + u^T Ru + 2\boldsymbol{x}^T P(A\boldsymbol{x} + Bu)\right] = 0.$$

Hence,

$$2Ru + 2B^T P\boldsymbol{x} = 0,$$

which gives the optimal input

$$u^* = -R^{-1}B^T P\boldsymbol{x}.$$

Defining

$$K_c = R^{-1}B^T P,$$

the optimal feedback law is

$$\boxed{u^* = -K_c \boldsymbol{x}}.$$

Substituting this optimal input back into the HJB equation gives

$$0 = \boldsymbol{x}^T \left(A^T P + PA - PBR^{-1}B^T P + Q \right) \boldsymbol{x}.$$

Since this equality must hold for every state $\boldsymbol{x}$, the matrix inside the quadratic form must be zero:

$$\boxed{A^T P + PA - PBR^{-1}B^T P + Q = 0}.$$

This equation is the continuous algebraic Riccati equation (CARE). After its stabilising solution P has been obtained, the continuous-time LQR gain is

$$\boxed{K_c = R^{-1}B^T P}.$$

The resulting closed-loop system is

$$\dot{\boldsymbol{x}} = (A - BK_c)\,\boldsymbol{x}.$$

For the stabilising Riccati solution, all eigenvalues of $A - BK_c$ lie in the open left half-plane, and the minimum cost is

$$\boxed{J_c^*(\boldsymbol{x}_0) = \boldsymbol{x}_0^T P \boldsymbol{x}_0}.$$

2.1.2 Alternative derivation by completing the square

The same optimal control law can also be verified without explicitly using the HJB equation.

Suppose that $P = P^T \succeq 0$ is the stabilising solution of the CARE,

$$A^T P + PA - PBR^{-1}B^T P + Q = 0.$$

Define the quadratic function

$$V(\boldsymbol{x}) = \boldsymbol{x}^T P \boldsymbol{x}.$$

Along the system trajectory,

$$\begin{aligned} \dot{V} &= \dot{\boldsymbol{x}}^T P \boldsymbol{x} + \boldsymbol{x}^T P \dot{\boldsymbol{x}} \\ &= \boldsymbol{x}^T (A^T P + PA) \boldsymbol{x} + 2\boldsymbol{x}^T PBu. \end{aligned}$$

Using the CARE to replace $A^T P + PA$ gives

$$A^T P + PA = -Q + PBR^{-1}B^T P.$$

Therefore,

$$\dot{V} = -\boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{x}^T PBR^{-1}B^T P \boldsymbol{x} + 2\boldsymbol{x}^T PBu.$$

Rearranging the instantaneous cost yields

$$\begin{aligned} \boldsymbol{x}^T Q \boldsymbol{x} + u^T R u &= -\dot{V} + u^T R u + 2\boldsymbol{x}^T PBu + \boldsymbol{x}^T PBR^{-1}B^T P \boldsymbol{x} \\ &= -\dot{V} + \left(u + R^{-1}B^T P \boldsymbol{x} \right)^T R \left(u + R^{-1}B^T P \boldsymbol{x} \right). \end{aligned}$$

Integrating from 0 to a finite time T gives

$$J_T = V(\boldsymbol{x}_0) - V(\boldsymbol{x}(T)) + \int_0^T \left(u + R^{-1}B^T P\boldsymbol{x}\right)^T R \left(u + R^{-1}B^T P\boldsymbol{x}\right) dt.$$

For the stabilising solution, $\boldsymbol{x}(T) \to 0$ as $T \to \infty$, and hence

$$V(\boldsymbol{x}(T)) \to 0.$$

Thus,

$$J_c = \boldsymbol{x}_0^T P\boldsymbol{x}_0 + \int_0^\infty \left(u + R^{-1}B^T P\boldsymbol{x}\right)^T R \left(u + R^{-1}B^T P\boldsymbol{x}\right) dt.$$

Because $R \succ 0$, the integral term is always non-negative. It reaches its minimum value of zero if and only if

$$u + R^{-1}B^T P\boldsymbol{x} = 0.$$

Therefore,

$$\boxed{u^* = -R^{-1}B^T P\boldsymbol{x} = -K_c\boldsymbol{x}},$$

and the minimum cost is

$$\boxed{J_c^* = \boldsymbol{x}_0^T P\boldsymbol{x}_0}.$$

The HJB derivation explains how the Riccati equation arises from Bellman's optimality principle, while the completing-the-square derivation directly verifies that the resulting feedback law achieves the global minimum of the quadratic cost.

2.2 Meaning of the weights

A larger diagonal entry of Q penalises the corresponding state more heavily. For example, a large q_α prioritises keeping the pendulum close to the upright position. A larger R penalises PWM use and usually produces a less aggressive controller. The relative scaling matters more than any single number considered alone.

2.3 Discrete-time infinite-horizon LQR

For the sampled system

$$\boldsymbol{x}_{k+1} = A_d\boldsymbol{x}_k + B_d u_k,$$

the infinite-horizon cost is

$$J_d = \sum_{k=0}^{\infty} \left(\boldsymbol{x}_k^T Q\boldsymbol{x}_k + u_k^T R u_k\right).$$

The discrete algebraic Riccati equation (DARE) is

$$P = A_d^T P A_d - A_d^T P B_d (R + B_d^T P B_d)^{-1} B_d^T P A_d + Q,$$

and the optimal gain is

$$K_d = (R + B_d^T P B_d)^{-1} B_d^T P A_d, \qquad u_k = -K_d\boldsymbol{x}_k.$$

3 Continuous-Model and Discrete-Model Designs

3.1 Linear model used in this laboratory

The upright linearised PWM-input model of the rotary inverted pendulum is

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bu,$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -11.3205 & -49.9955 & 0.4820 \\ 0 & 0 & 0 & 1 \\ 0 & 16.9206 & 171.2602 & -1.6510 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 0.8171 \\ 0 \\ -1.2213 \end{bmatrix}.$$

The real-time controller runs at 200 Hz, therefore the sampling period is

$$T_s = 0.005 \text{ s}.$$

In this laboratory, two different LQR design approaches are investigated.

1. **Continuous-model LQR design:**

   The controller is designed directly from the continuous-time model (A, B). The continuous algebraic Riccati equation (CARE) is solved to obtain the optimal feedback gain

   $$u = -K_c x.$$

2. **Discrete-model LQR design:**

   First, the continuous model is converted into a discrete-time model using zero-order-hold (ZOH) discretisation:

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

   Then the discrete algebraic Riccati equation (DARE) is solved to obtain

   $$u_k = -K_d x_k.$$

Although both controllers are finally executed by a digital controller with the same sampling period, their design processes are different. Therefore, the obtained feedback gains

$$K_c \neq K_d$$

are generally not identical.

3.2 LQR cost function and weight selection

For continuous-time LQR, the performance index is

$$J_c = \int_0^\infty (x^T Q x + u^T R u)\, dt,$$

while for discrete-time LQR,

$$J_d = \sum_{k=0}^{\infty} (x_k^T Q x_k + u_k^T R u_k).$$

The weighting matrices are defined as

$$Q = \text{diag}(q_1, q_2, q_3, q_4), \qquad R = r.$$

Students should select suitable values of Q and R according to the control objectives.

The elements of Q correspond to different state variables. A larger weight means that the corresponding state deviation is penalised more strongly. Therefore, the selection of each diagonal element should reflect the priority of different control objectives, such as:

- reducing pendulum angle error;
- limiting rotary arm motion;
- reducing angular velocity oscillation;
- balancing stabilization performance and control effort.

The matrix R represents the penalty on the control input. A smaller R allows stronger control actions but may increase actuator effort, while a larger R produces smoother but slower responses.

3.3 Required calculations

STUDENT TASK

Students must complete two independent LQR designs and report the results.

1. **Continuous-model design**

   Starting from

   $$(A, B),$$

   solve the continuous algebraic Riccati equation (CARE):

   $$A^T P_c + P_c A - P_c B R^{-1} B^T P_c + Q = 0.$$

   Then calculate

   $$K_c = R^{-1} B^T P_c.$$

2. **Discrete-model design**

   First obtain the ZOH discrete model:

   $$A_d, B_d$$

   with

   $$T_s = 0.005s.$$

   Then solve the discrete algebraic Riccati equation (DARE):

   $$P_d = A_d^T P_d A_d - A_d^T P_d B_d (R + B_d^T P_d B_d)^{-1} B_d^T P_d A_d + Q.$$

   The discrete feedback gain is

   $$K_d = (R + B_d^T P_d B_d)^{-1} B_d^T P_d A_d.$$

Report the complete calculation process, including:

- selected Q and R;
- discretised matrices A_d, B_d;
- Riccati equation solutions;

- final gains K_c and K_d.

The two gains should be different because they are obtained by optimizing two different mathematical models.

3.4 Example calculation procedure

Students may use Python or MATLAB to perform the calculations.

```python
import numpy as np
from scipy.linalg import solve_continuous_are
from scipy.linalg import solve_discrete_are
from scipy.signal import cont2discrete

# Define continuous model
A = ...
B = ...

# Select Q and R
Q = ...
R = ...

Ts = 0.005

# -------- Continuous LQR --------
Pc = solve_continuous_are(A, B, Q, R)
Kc = np.linalg.inv(R) @ B.T @ Pc


# -------- Discrete LQR --------
Ad, Bd, _, _, _ = cont2discrete(
    (A, B, np.eye(4), np.zeros((4,1))),
    Ts,
    method="zoh"
)

Pd = solve_discrete_are(Ad, Bd, Q, R)

Kd = np.linalg.inv(
    R + Bd.T @ Pd @ Bd
) @ Bd.T @ Pd @ Ad

print(Kc)
print(Kd)
```

3.5 Controller implementation and comparison

Both controllers are evaluated using the same nonlinear rotary inverted pendulum simulation environment with a 5 ms control interval.

For the continuous-model controller:

$$u_k = -K_c x_k,$$

where K_c is obtained from the continuous CARE design.

For the discrete-model controller:

$$u_k = -K_d x_k,$$

where K_d is obtained from the discrete DARE design.

The purpose of this comparison is to investigate the difference between designing an optimal controller before and after discretisation.

The teacher demonstration simulation and physical-system experiment use the validated discrete-model LQR controller.

4 Running the Nonlinear Simulation Environment

4.1 Files and dependencies

File	Purpose
`rip_lqr_sim.py`	The four entries of K are editable and initially zero.

Install the required packages:

```
python3 -m pip install numpy scipy matplotlib PyQt5
```

Run either file from a terminal:

```
python3 rip_lqr_sim.py
python3 rip_lqr_sim.py
```

4.2 Student-panel workflow

1. Open `rip_lqr_sim.py`.
2. Enter K in the exact order $[K_\theta, K_{\dot\theta}, K_\alpha, K_{\dot\alpha}]$.
3. Keep the initial condition, duration, swing-up settings, blend settings and noise settings fixed when comparing K_c and K_d.
4. Click **SAVE / Apply Settings**. Unsaved values are deliberately ignored by **GO**.
5. Click **GO**. The program displays the nonlinear 3D motion and saves a PNG result figure and, when selected, a CSV log.
6. Repeat with the other gain and use clear file identifiers such as `SIM\_CONT\_R1` and `SIM\_DISC\_R1`.

4.3 Optional headless execution

The student file can also run without the GUI. Replace the four placeholders by your calculated values:

```
python3 rip_lqr_sim.py --headless --duration 10 \
  --k-theta K1 --k-theta-dot K2 --k-alpha K3 --k-alpha-dot K4
```

5 Running the Physical System

5.1 Files

File	Purpose
`rip_lqr_hardware.ino`	STM32 firmware: 200 Hz sensing, velocity filtering, swing-up, LQR, blend, motor drive and telemetry.
`rip_lqr_hardware.py`	Student physical-system panel with editable K.

5.2 Before running

The electrical wiring is unchanged from Class 8. Follow the Class-8 hardware wiring guide and its safety inspection procedure. Flash `rip_lqr_hardware.ino` using the same STM32 board configuration used in Class 8. The Class-9 Python panels require:

```
python3 -m pip install numpy matplotlib PyQt5 pyserial
```

DANGER

LQR is local and the pendulum, arm and cables move quickly during swing-up. Keep the complete rotation region clear. Do not touch the apparatus while the motor supply is connected. Press **STOP** immediately if the arm repeatedly hits a limit, a wire moves, or the measured angles are clearly incorrect.

5.3 Calibration and experiment sequence

1. Connect the STM32 to the computer and start the appropriate panel:

```
python3 rip_lqr_hardware.py
```

2. Select the serial port and click **Connect**. Wait for the Class-9 LQR firmware-ready message.
3. Hold the pendulum upright and click **Hold upright** → **Calibrate** $\alpha = 0$.
4. Let the pendulum hang down and click **Hang down** → **Calibrate** $\alpha = \pi$.
5. In the student panel, enter the discrete-design gain K_d in state order. Do not enter $-K_d$.
6. Check PWM limit, enter/exit angles, blend, velocity filter and motor sign. Click **SAVE / Send Parameters to STM32** and wait for the acknowledgement.
7. Clear the motion area and click **GO**. The firmware performs swing-up first and blends to LQR near the upright region.
8. Save the automatically generated CSV and PNG files. Repeat the required trials without changing the selected settings.

5.4 What runs on the STM32

At each 5 ms update the firmware:

1. samples the potentiometer and encoder;
2. converts them to α and θ;
3. estimates $\dot{\alpha}$ and $\dot{\theta}$ using numerical differences followed by the Class-8 low-pass filter;
4. computes the phase-pumping swing-up command;
5. computes $u_{\mathrm{LQR}} = -Kx$;
6. applies hysteresis and soft blending;
7. saturates the command and drives the motor;
8. transmits telemetry to the Python panel.

The Python program is used for configuration, visualisation and logging; the real-time control command is calculated on the STM32.

6 Quick Troubleshooting

Symptom	Check
GO is disabled	Connect, calibrate, enter a nonzero gain in the student panel, click SAVE, and wait for STM32 acknowledgement.
Firmware mismatch	Flash `rip_lqr_hardware.ino`; the panel expects protocol version 3.
Immediate divergence near upright	Check state order, gain signs, motor sign, calibration and angle convention. The program expects $u = -Kx$.
Pendulum never reaches LQR stage	Check swing PWM, calibration, enter angle and mechanical freedom. LQR does not replace swing-up.
Very noisy PWM	Check sensor wiring, calibration, velocity-filter coefficient and encoder direction.
Simulation uses old values	Click SAVE after every gain or setting change.