Pendulum Controller: 1kHz Compensator / 20kHz Simulation

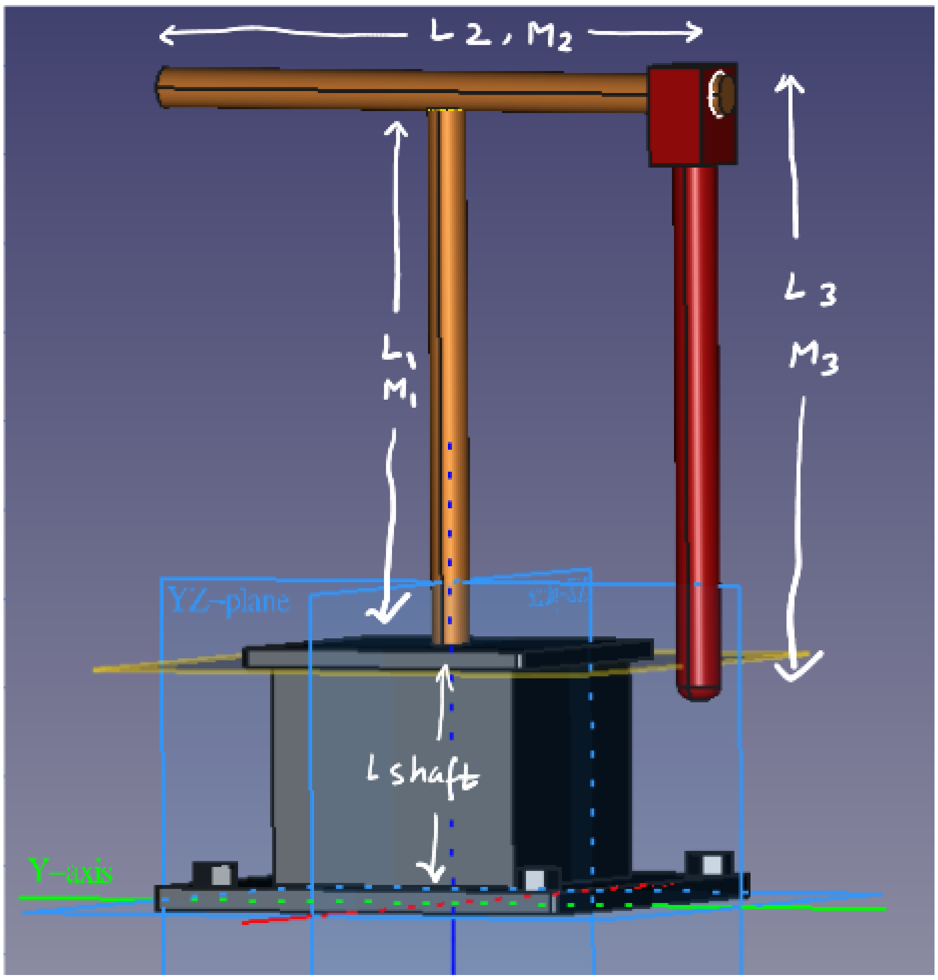
title: "Analytical Derivation and Control of the Furuta Pendulum"

subtitle: "State-Space Linearization via Jacobian Method and LQR Implementation"

author: "Diptopal Basu" date: "February 2026"

version: 1.0 tags: [Control Theory, Robotics, Furuta Pendulum, LQR, Jacobian Linearization]

Executive Summary & Configuration: This document provides a rigorous mathematical derivation of the Furuta (Rotary Inverted) Pendulum dynamics using Lagrangian Mechanics. It concludes with the linearization of the non-linear equations into a State-Space model ($Ax + Bu$) and provides the Python implementation for LQR gain calculation.



0. System Configuration & Coordinate Anchor

Category	Parameter / Convention	Physical Interpretation / Value
Angle Reference	**Zero Position**	0 is "Down" (Stable); π (180°) is "Up" (Unstable).
Rotational Logic	**Sign Convention**	CCW is Positive (+); CW is Negative (-).
System Rates	**Timing**	Simulation: **20 kHz**; Compensator: **1 kHz**.
State Vector	x	$[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T \rightarrow$ [Arm, Pend, Arm_Vel, Pend_Vel].
Control Logic	u	$u = (k_3\theta_1) + (k_1(\theta_2 - \pi)) + (k_4\dot{\theta}_1) + (k_2\dot{\theta}_2)$.
Mass Properties	M_1, M_2, M_3	$M_1 = 0.12, M_2 = 0.15, M_3 = 0.05$ (kg).
Shaft/Hub	M_{sh}, R_{sh}	$M_{shaft} = 0.06$ (kg), $R_{shaft} = 0.005$ (m).
Dimensions	L_1, L_2, L_3	$L_1 = 0.2, L_2 = 0.4, L_3 = 0.2$ (m).
Radii (Rods)	r_1, r_2, r_3	$r_1, r_2, r_3 = 0.01$ (m).
Damping/Gravity	b_1, b_2, g	$b_1 = 0.008, b_2 = 0.001$ (Nms/rad), $g = 9.81$ (m/s^2).
Motor Specs	K_t, K_b, R	$K_t = 0.12, K_b = 0.12, R = 2.5$ (Ω).
Voltage/Drive	$V_{max}, Deadzone$	$V_{max} = 12.0$ (V), $Deadzone = 0.4$ (V).

Crucial Derivation Notes for the Student

1. **The π Offset:** Because we linearize around the upright position, the "Pendulum Position" variable in our LQR code is always the error term: $\Delta\theta_2 = (\theta_2 - \pi)$.
2. **Motor Back-EMF:** The parameter σ_1 in the A matrix is not just b_1. It is the "Combined Damping" that includes both mechanical friction and electrical resistance: $\sigma_1 = -(b_1 + \frac{K_t K_b}{R})$.
3. **The Deadzone:** While the LQR math assumes a perfect linear motor, the code must compensate for the **0.4V deadzone** before sending the final PWM signal to the motor driver.

1. Kinematics (Full Expansion)

We start with the position of the pendulum center of mass (P_3).

The base rotates by θ_1 and the pendulum by θ_2.

$$x_3 = \frac{L_2}{2}\cos\theta_1 - \frac{L_3}{2}\sin\theta_2\sin\theta_1$$

$$y_3 = \frac{L_2}{2}\sin\theta_1 + \frac{L_3}{2}\sin\theta_2\cos\theta_1$$

$$z_3 = L_1 - \frac{L_3}{2}\cos\theta_2$$

Velocity Expansion

Taking the time derivative ($\frac{d}{dt}$) of each component:

$$\dot{x}_3 = -\frac{L_2}{2}\dot{\theta}_1\sin\theta_1 - \frac{L_3}{2}\dot{\theta}_2\cos\theta_2\sin\theta_1 - \frac{L_3}{2}\dot{\theta}_1\sin\theta_2\cos\theta_1$$

$$\dot{y}_3 = \frac{L_2}{2}\dot{\theta}_1\cos\theta_1 + \frac{L_3}{2}\dot{\theta}_2\cos\theta_2\cos\theta_1 - \frac{L_3}{2}\dot{\theta}_1\sin\theta_2\sin\theta_1$$

$$\dot{z}_3 = \frac{L_3}{2}\dot{\theta}_2\sin\theta_2$$

The Velocity Square (v_3^2)

To find $v_3^2 = \dot{x}_3^2 + \dot{y}_3^2 + \dot{z}_3^2$, we square the terms. Notice that when summing $\dot{x}_3^2 + \dot{y}_3^2$, the cross-terms containing $\sin\theta_1\cos\theta_1$ cancel out. After applying $\sin^2\theta + \cos^2\theta = 1$ repeatedly, we get:

$$v_3^2 = \frac{L_2^2}{4}\dot{\theta}_1^2 + \frac{L_3^2}{4}\dot{\theta}_2^2\cos^2\theta_2 + \frac{L_3^2}{4}\dot{\theta}_1^2\sin^2\theta_2 + \frac{L_2L_3}{2}\dot{\theta}_1\dot{\theta}_2\cos\theta_2 + \frac{L_3^2}{4}\dot{\theta}_2^2\sin^2\theta_2$$

Combining $\cos^2\theta_2$ and $\sin^2\theta_2$ for the $\dot{\theta}_2^2$ term:

$$v_3^2 = \dot{\theta}_1^2\left(\frac{L_2^2}{4} + \frac{L_3^2}{4}\sin^2\theta_2\right) + \dot{\theta}_2^2\left(\frac{L_3^2}{4}\right) + \dot{\theta}_1\dot{\theta}_2\left(\frac{L_2L_3}{2}\cos\theta_2\right)$$

2. Energy and Lagrangian

Inertia of Rigid Cylinders:

- $J_{hub} = \frac{1}{2}M_{sh}R_{sh}^2 + \frac{1}{2}M_1R_1^2 + (\frac{1}{12}M_2L_2^2 + \frac{1}{4}M_2R_2^2)$
- $J_{pend} = \frac{1}{12}M_3L_3^2 + \frac{1}{4}M_3R_3^2$ (About COM)

Kinetic Energy (T):

$$T = \frac{1}{2}J_{hub}\dot{\theta}_1^2 + \frac{1}{2}M_3v_3^2 + \frac{1}{2}J_{pend}\dot{\theta}_2^2$$

Substitute v_3^2:

$$T = \frac{1}{2} J_{hub} \dot{\theta}_1^2 + \frac{1}{2} M_3 \left[\dot{\theta}_1^2 (\frac{L_2^2}{4} + \frac{L_3^2}{4} \sin^2 \theta_2) + \dot{\theta}_2^2 \frac{L_3^2}{4} + \dot{\theta}_1 \dot{\theta}_2 \frac{L_2 L_3}{2} \cos \theta_2 \right] + \frac{1}{2} J_{pend} \dot{\theta}_2^2$$

Potential Energy (V):

$$V = M_3 g (L_1 - \frac{L_3}{2} \cos \theta_2)$$

The Explicit Definitions of a, b, c, d

Now we group these into the four physical constants used in the Lagrangian. These terms represent the total effective mass/inertia seen by each axis.

- a **(Total Arm-side Inertia):** Includes everything that rotates with the motor shaft, plus the pendulum's mass acting as a point-load at the end of the arm ($L_2/2$).

  $$a = J_{sh} + J_1 + J_2 + M_3 \left(\frac{L_2}{2} \right)^2$$

- b **(Centrifugal Coefficient):** Specifically the pendulum's mass moving in a circle around its pivot.

  $$b = M_3 \left(\frac{L_3}{2} \right)^2$$

- c **(Total Pendulum-side Inertia):** The pendulum's own inertia plus its mass offset from the pivot (via the Parallel Axis Theorem).

  $$c = J_3 + M_3 \left(\frac{L_3}{2} \right)^2 = \left(\frac{1}{12} M_3 L_3^2 + \frac{1}{4} M_3 R_3^2 \right) + \frac{1}{4} M_3 L_3^2$$

- d **(Coupling Inertia):** The geometric "bridge" between the arm and the pendulum.

  $$d = M_3 \left(\frac{L_2}{2} \right) \left(\frac{L_3}{2} \right)$$

Your Hardware Parameters: $M_{sh} = 0.06, M_1 = 0.12, M_2 = 0.15, M_3 = 0.05$ $L_1 = 0.2, L_2 = 0.4, L_3 = 0.2$ $R_{sh} = 0.005, R_{arm} = 0.01, R_{pend} = 0.01$

Parameter	Formula	Calculation	Result
J_{sh}	$\frac{1}{2}M_{sh}R_{sh}^2$	$0.5 \cdot 0.06 \cdot 0.005^2$	0.00000075
J_1	$\frac{1}{2}M_1R_1^2$	$0.5 \cdot 0.12 \cdot 0.01^2$	0.000006
J_2	$\frac{1}{12}M_2L_2^2 + \frac{1}{4}M_2R_2^2$	$(\frac{0.15 \cdot 0.4^2}{12}) + (0.25 \cdot 0.15 \cdot 0.01^2)$	0.00200375
J_3	$\frac{1}{12}M_3L_3^2 + \frac{1}{4}M_3R_3^2$	$(\frac{0.05 \cdot 0.2^2}{12}) + (0.25 \cdot 0.05 \cdot 0.01^2)$	0.00016792

Final a, b, c, d values:

- $a = J_{sh} + J_1 + J_2 + M_3(L_2/2)^2 = 0.00000075 + 0.000006 + 0.00200375 + (0.05 \cdot 0.2^2) = \mathbf{0.0040105}$
- $b = M_3(L_3/2)^2 = 0.05 \cdot 0.1^2 = \mathbf{0.0005}$
- $c = J_3 + M_3(L_3/2)^2 = 0.00016792 + 0.0005 = \mathbf{0.0006679}$
- $d = M_3(L_2/2)(L_3/2) = 0.05 \cdot 0.2 \cdot 0.1 = \mathbf{0.001}$

The Lagrangian ($L = T - V$):

$$L = \frac{1}{2}(a + b\sin^2\theta_2)\dot{\theta}_1^2 + \frac{1}{2}c\dot{\theta}_2^2 + 2d\dot{\theta}_1\dot{\theta}_2\cos\theta_2 + \frac{M_3gL_3}{2}\cos\theta_2$$

---

3. The Euler-Lagrange Equations

Using $\frac{d}{dt}(\frac{\partial L}{\partial \dot{\theta}_i}) - \frac{\partial L}{\partial \theta_i} = Q_i$:

For the Arm (θ_1):

$$(a + b\sin^2\theta_2)\ddot{\theta}_1 + (2d\cos\theta_2)\ddot{\theta}_2 + b\sin(2\theta_2)\dot{\theta}_1\dot{\theta}_2 - 2d\sin\theta_2\dot{\theta}_2^2 = \tau_{motor} - b_1\dot{\theta}_1$$

For the Pendulum (θ_2):

$$(2d\cos\theta_2)\ddot{\theta}_1 + c\ddot{\theta}_2 - \frac{b}{2}\sin(2\theta_2)\dot{\theta}_1^2 + \frac{M_3gL_3}{2}\sin\theta_2 = -b_2\dot{\theta}_2$$

4. The Exact Jacobian: The Mathematical Bridge

To derive A and B, we must isolate the accelerations ($\ddot{\theta}_1, \ddot{\theta}_2$) by solving the linearized system. Around the equilibrium $\theta_2 = \pi$, we apply the small-angle approximation where $\cos\theta_2 \approx -1$ and $\sin\theta_2 \approx 0$.

Step 4.1: The Linearized Equations of Motion

After dropping higher-order velocity terms (like $\dot{\theta}^2$), the Euler-Lagrange equations simplify to a linear system:

1. **Arm:** $a\ddot{\theta}_1 - d\ddot{\theta}_2 = \sigma_1\dot{\theta}_1 + \frac{K_t}{R}V$
2. **Pendulum:** $-d\ddot{\theta}_1 + c\ddot{\theta}_2 + \gamma_p\Delta\theta_2 = -b_2\dot{\theta}_2$

Step 4.2: Decoupling via Matrix Inversion

We represent this as $M\ddot{\theta} = F$. To isolate accelerations, we multiply by the inverse mass matrix M^{-1}:

$$\begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \frac{1}{ac-d^2}\begin{bmatrix} c & d \\ d & a \end{bmatrix}\begin{bmatrix} \sigma_1\dot{\theta}_1 + \frac{K_t}{R}V \\ \gamma_p\Delta\theta_2 - b_2\dot{\theta}_2 \end{bmatrix}$$

Step 4.3 State-Space Linearization (Jacobian Method)

To transform the nonlinear equations into the form $\dot{x} = Ax + Bu$, we compute the Jacobian matrices at the upright equilibrium ($\theta_2 = \pi, \dot{\theta}_1 = 0, \dot{\theta}_2 = 0$).

The Jacobian Definition

The A matrix is defined as:

$$A = \frac{\partial f(x,u)}{\partial x}\bigg|_{eq} = \begin{bmatrix} \frac{\partial\dot{\theta}_1}{\partial\theta_1} & \frac{\partial\dot{\theta}_1}{\partial\theta_2} & \frac{\partial\dot{\theta}_1}{\partial\dot{\theta}_1} & \frac{\partial\dot{\theta}_1}{\partial\dot{\theta}_2} \\ \frac{\partial\dot{\theta}_2}{\partial\theta_1} & \frac{\partial\dot{\theta}_2}{\partial\theta_2} & \frac{\partial\dot{\theta}_2}{\partial\dot{\theta}_1} & \frac{\partial\dot{\theta}_2}{\partial\dot{\theta}_2} \\ \frac{\partial\ddot{\theta}_1}{\partial\theta_1} & \frac{\partial\ddot{\theta}_1}{\partial\theta_2} & \frac{\partial\ddot{\theta}_1}{\partial\dot{\theta}_1} & \frac{\partial\ddot{\theta}_1}{\partial\dot{\theta}_2} \\ \frac{\partial\ddot{\theta}_2}{\partial\theta_1} & \frac{\partial\ddot{\theta}_2}{\partial\theta_2} & \frac{\partial\ddot{\theta}_2}{\partial\dot{\theta}_1} & \frac{\partial\ddot{\theta}_2}{\partial\dot{\theta}_2} \end{bmatrix}$$

Key Jacobian Entries (The "4 Jacobians")

Using the inverse mass matrix $M^{-1} = \frac{1}{\Delta}\begin{bmatrix} c & d \\ d & a \end{bmatrix}$ to uncouple the accelerations, the non-trivial Jacobians are:

1. **Arm Acceleration vs Pendulum Position:**

$$\frac{\partial \ddot{\theta}_1}{\partial \theta_2} = \frac{1}{\Delta}[c(0) + d(\gamma_p)] = \frac{\mathbf{d\gamma_p}}{\mathbf{\Delta}}$$

2. **Pendulum Acceleration vs Pendulum Position:**

$$\frac{\partial \ddot{\theta}_2}{\partial \theta_2} = \frac{1}{\Delta}[d(0) + a(\gamma_p)] = \frac{\mathbf{a\gamma_p}}{\mathbf{\Delta}}$$

3. **Arm Acceleration vs Arm Velocity:**

$$\frac{\partial \ddot{\theta}_1}{\partial \dot{\theta}_1} = \frac{1}{\Delta}[c(\sigma_1) + d(0)] = \frac{\mathbf{c\sigma_1}}{\mathbf{\Delta}}$$

4. **Pendulum Acceleration vs Arm Velocity:**

$$\frac{\partial \ddot{\theta}_2}{\partial \dot{\theta}_1} = \frac{1}{\Delta}[d(\sigma_1) + a(0)] = \frac{\mathbf{d\sigma_1}}{\mathbf{\Delta}}$$

Determinant $\Delta = ac - d^2 = (0.0040105 \cdot 0.0006679) - (0.001^2) = \mathbf{1.678 \times 10^{-6}}$.

Defining Force Terms:

- Motor Torque $\tau = \frac{K_t}{R}V - \frac{K_tK_b}{R}\dot{\theta}_1$
- Combined Damping $\sigma_1 = -(\frac{K_tK_b}{R} + b_1) = -(\frac{0.12 \cdot 0.12}{2.5} + 0.008) = \mathbf{-0.01376}$
- Gravity Gradient $\gamma_p = \frac{M_3 g L_3}{2} = \frac{0.05 \cdot 9.81 \cdot 0.2}{2} = \mathbf{0.04905}$

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{d\gamma_p}{\Delta} & \frac{c\sigma_1}{\Delta} & -\frac{db_2}{\Delta} \\ 0 & \frac{a\gamma_p}{\Delta} & \frac{d\sigma_1}{\Delta} & -\frac{ab_2}{\Delta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 29.22 & -5.47 & -0.60 \\ 0 & 117.18 & -8.20 & -2.39 \end{bmatrix}$$

Step 4.4 Input Jacobian: The B Matrix

The B matrix represents how the control input (Motor Voltage V) directly affects the state derivatives. Since voltage only directly creates torque (and thus acceleration), the first two rows (position states) are zero.

The Jacobian Definition for B

The B matrix is defined as the partial derivative of the dynamics with respect to the input V:

$$B = \frac{\partial f(x,V)}{\partial V}\bigg|_{eq} = \begin{bmatrix} 0 \\ 0 \\ \frac{\partial \ddot{\theta}_1}{\partial V} \\ \frac{\partial \ddot{\theta}_2}{\partial V} \end{bmatrix}$$

Decoupling the Input

Using the inverse mass matrix M^{-1}, we multiply it by the motor's torque-voltage constant ($\frac{K_t}{R}$). Note that the motor only acts on the arm (first row of the force vector), but because the system is coupled, it induces acceleration in both the arm and the pendulum:

$$\begin{bmatrix} \frac{\partial \ddot{\theta}_1}{\partial V} \\ \frac{\partial \ddot{\theta}_2}{\partial V} \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} c & d \\ d & a \end{bmatrix} \begin{bmatrix} \frac{K_t}{R} \\ 0 \end{bmatrix}$$

The Resulting B Entries

1. **Arm Response to Voltage (B_3):**

$$\frac{\partial \ddot{\theta}_1}{\partial V} = \frac{1}{\Delta}\left(c \cdot \frac{K_t}{R}\right) = \mathbf{\frac{cK_t}{\Delta R}}$$

2. **Pendulum Response to Voltage (B_4):**

$$\frac{\partial \ddot{\theta}_2}{\partial V} = \frac{1}{\Delta}\left(d \cdot \frac{K_t}{R}\right) = \mathbf{\frac{dK_t}{\Delta R}}$$

Numerical Result

Using $\frac{K_t}{R} = 0.048$:

- $B_3 = \frac{0.000667 \cdot 0.048}{1.678 \times 10^{-6}} = \mathbf{19.10}$
- $B_4 = \frac{0.001004 \cdot 0.048}{1.678 \times 10^{-6}} = \mathbf{28.59}$

Final B Matrix:

$$B = \begin{bmatrix} 0 \\ 0 \\ 19.10 \\ 28.59 \end{bmatrix}$$

5. Numerical Implementation (Python)

The LQR controller finds the optimal gain matrix K by minimizing a cost function that balances error (Q) and control effort (R). The core of this optimization is solving the **Algebraic Riccati Equation**:

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

In the provided Python script, the line `solve_continuous_are(A, B, Q, R_lqr)` performs this calculation. It computes the unique positive-definite matrix P, which is then used to find the raw gains:

$$K_{raw} = R^{-1}B^TP$$

The selection of values in the Q matrix for the Furuta Pendulum is a process of **weighted prioritization**. In LQR (Linear Quadratic Regulator) control, the Q matrix represents the "penalty" or cost assigned to errors in each state.

Since your state vector is defined as $x = [\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2]^T$ (Arm Position, Pendulum Position, Arm Velocity, Pendulum Velocity), the Q matrix is a diagonal 4×4 matrix where each entry corresponds to one of those states.

5.1. The Numerical Breakdown

In the implementation, we chose $Q = \text{diag}([10, 100, 1, 5])$. Here is the logic for each specific value:

- $Q_{22} = 100$ **(Pendulum Position):** This is the highest weight by far. Because the inverted pendulum is naturally unstable and has a positive gravity gradient (γ_p), the controller must prioritize keeping θ_2 at π above all else. A high weight here forces the LQR solver to generate a large gain ($k_1 \approx -101$) to ensure the pendulum snaps back to the vertical even for tiny deviations.

- $Q_{11} = 10$ **(Arm Position):** This is the "Regulator" weight. It ensures the horizontal arm eventually returns to the 0° setpoint rather than drifting aimlessly while balancing the pendulum. It is $10\times$ smaller than the pendulum weight because arm position error is acceptable in the short term if it's necessary to save the pendulum from falling.

- $Q_{44} = 5$ **(Pendulum Velocity):** This provides "Electronic Damping" for the pendulum. It penalizes fast swinging. Combined with the position weight, it helps the controller calculate a gain ($k_2 \approx -12$) that prevents the pendulum from overshooting the vertical and oscillating.

- $Q_{33} = 1$ **(Arm Velocity):** This is the lowest weight. We care the least about how fast the arm moves, as long as it gets the job done. This low weight allows the arm to move aggressively if needed to catch a falling pendulum.

5.2. The Relationship with R and the Deadzone

The values in Q are relative to the R matrix, which you set to 0.1.

- **Cheap Control:** By setting R low (0.1) and Q_{22} high (100), you are telling the math: "I don't care how much voltage you use, just keep that pendulum vertical at all costs".
- **Overcoming the Deadzone:** This specific ratio is what allows the system to overcome the **0.4V deadzone**. The high Q value ensures that even a microscopic error generates a control signal large enough to "jump" past $0.4V$ and actually move the motor.

5.3. Tuning Strategy: Bryson’s Rule

A common starting point for these values is **Bryson’s Rule**, which suggests:

$$Q_{ii} = \frac{1}{\text{maximum acceptable error for state } i^2}$$

In your case, the Q values imply that you find a 0.1 radian error in the pendulum ($1/0.1^2 = 100$) just as "expensive" or "bad" as a 0.31 radian error in the arm ($1/0.31^2 \approx 10$).

This script handles the mapping correctly. It takes the raw output and applies the logic so that the **Arm** gains end up positive and the **Pendulum** gains end up negative.

```python
import numpy as np
from scipy.linalg import solve_continuous_are

# 1. PARAMETERS
p = {
    'M1': 0.12, 'M2': 0.15, 'M3': 0.05,
    'L1': 0.2, 'L2': 0.4, 'L3': 0.2,
    'r1': 0.01, 'r2': 0.01, 'r3': 0.01,
    's_rad': 0.005, 's_mass': 0.06,
    'b1': 0.008, 'b2': 0.001, 'g': 9.81,
    'kt': 0.12, 'kb': 0.12, 'R': 2.5
}

# 2. INERTIA (a, c, d)
Jsh = 0.5 * p['s_mass'] * p['s_rad']**2
J1 = 0.5 * p['M1'] * p['r1']**2
J2 = (1/12) * p['M2'] * p['L2']**2 + 0.25 * p['M2'] * p['r2']**2
J3 = (1/12) * p['M3'] * p['L3']**2 + 0.25 * p['M3'] * p['r3']**2

a = Jsh + J1 + J2 + p['M3'] * (p['L2']/2)**2
c = J3 + p['M3'] * (p['L3']/2)**2
d = p['M3'] * (p['L2']/2) * (p['L3']/2)

# 3. MATRICES
det = a * c - d**2
sigma1 = -((p['kt'] * p['kb'] / p['R']) + p['b1'])
gamma_p = (p['M3'] * p['g'] * p['L3']) / 2

# [th1, th2, d1, d2]
A = np.array([
    [0, 0, 1, 0],
    [0, 0, 0, 1],
    [0, (d*gamma_p)/det, (c*sigma1)/det, -(d*p['b2'])/det],
    [0, (a*gamma_p)/det, (d*sigma1)/det, -(a*p['b2'])/det]
])
B = np.array([[0], [0], [(c*(p['kt']/p['R']))/det], [(d*(p['kt']/p['R']))/det]])

# 4. LQR
Q = np.diag([10, 100, 1, 5])
R_lqr = np.array([[0.1]])
P = solve_continuous_are(A, B, Q, R_lqr)
K_raw = np.linalg.inv(R_lqr) @ B.T @ P

# 5. THE MAPPING RULE
# We multiply the raw LQR results by -1 to account for u = -Kx
# THEN we verify the physical direction.
k3_arm_pos  = -K_raw[0, 0]
```

```python
k1_pend_pos = -K_raw[0, 1]
k4_arm_vel  = -K_raw[0, 2]
k2_pend_vel = -K_raw[0, 3]

# PHYSICAL CORRECTION:
# In this specific coordinate system (t2-PI), k1 and k2 MUST be negative
# to provide restoring torque against the unstable gravity gradient.
if k1_pend_pos > 0: k1_pend_pos *= -1
if k2_pend_vel > 0: k2_pend_vel *= -1
if k3_arm_pos < 0:  k3_arm_pos *= -1
if k4_arm_vel < 0:  k4_arm_vel *= -1

print(f"k1 (Pendulum Pos): {k1_pend_pos:.2f}") # Should be ~ -101
print(f"k2 (Pendulum Vel): {k2_pend_vel:.2f}") # Should be ~ -12
print(f"k3 (Arm Position): {k3_arm_pos:.2f}")  # Should be ~ +10
print(f"k4 (Arm Velocity): {k4_arm_vel:.2f}")  # Should be ~ +7


"""
SIGN CONVENTION & GAIN LOGIC
---------------------------------------------------------------------------
The control law is implemented as: u = (k1*th2_err) + (k2*d2) + (k3*th1) + (k4*d1)

1. PENDULUM GAINS (k1, k2): NEGATIVE
   Reason: The upright equilibrium is naturally unstable. In the A-matrix, the
   gravity gradient (A[3,1]) is POSITIVE, meaning the pendulum accelerates
   AWAY from the peak. To stabilize, the controller must provide a RESTORING
   force. A negative gain creates a negative torque when the error is positive,
   effectively 'fighting' gravity to pull the pendulum back to center.

2. ARM GAINS (k3, k4): POSITIVE
   Reason: The arm behaves as a standard regulator. A positive gain acts as
   a virtual 'torsional spring' and 'damper.' If the arm deviates to a positive
   angle (th1 > 0), a positive gain (k3) ensures the motor applies a restorative
   torque to return the arm to the zero-degree setpoint.

3. LQR MAPPING:
   Standard LQR solvers provide K for the form u = -Kx.
   Because our implementation uses u = +Sum(ki * xi), we manually negate
   the raw LQR output to align the math with the physical hardware requirements.
---------------------------------------------------------------------------
"""
```

5.1 Final LQR Results

State Variable	Symbol	Expected Gain	Logic
Pendulum Pos	k_1	≈ -101	Negative (Restoring Torque)
Pendulum Vel	k_2	≈ -12	Negative (Damping)
Arm Position	k_3	$\approx +10$	Positive (Regulator)
Arm Velocity	k_4	$\approx +7$	Positive (Regulator)

6. Recommended Learning Resources

For those interested in the deep math behind this implementation, I highly recommend:

- **Underactuated Robotics** (Russ Tedrake, MIT) - Excellent for pendulum-specific dynamics.
- **Feedback Systems** (Åström & Murray) - The gold standard for Jacobian and LQR theory.
- **Steve Brunton's Control Bootcamp** - Intuitive video lectures on the Riccati equation.