PID Tuning Cheatsheet for a Servo with a Fast Current (Torque) Loop

**Goal:** Practical recipe to pick **PID** gains for a **single outer position loop** when the drive’s **current loop is fast** (~20 kHz), so torque ≈ proportional to current command.

## 1) Assumptions & Symbols

* Inner current loop tightly closed → torque follows command:
* Use an effective torque gain (set if your controller outputs current; otherwise fold it into PID gains).
* Mechanical plant about motor shaft (load reflected): inertia , viscous damping .
* Laplace variable , position , speed .
* Mechanical pole (no control):

## 2) Plant and Loop Definitions

**Torque → motion:**

**Controller (PID):**

Ideal form

Practical derivative (noise-friendly)

**Open loop (unity position feedback):**

**Closed-loop complementary sensitivity (command → position):**

**Sensitivity (disturbance rejection; error to reference):**

**With ideal PID**, these expand to

**Stability margins** (from Bode of ):

Recommended: PM ≈ 45–60°, GM ≥ 6–10 dB.

## 3) One‑Shot Tuning Recipe (Position Loop Only)

Shapes the loop to look like at crossover (good damping).

**A. “Velocity feedback”:** place controller zero near the mechanical pole

**B. Pick target crossover** . Start with .

**C. Proportional gain (unity-gain at crossover)**

**D. Add slow integral (remove steady-state error)**

**E. Filter derivative**

**Summary (compute in this order):** 1.   
2. choose   
3.   
4.   
5. choose and set   
6.

## 4) How to Verify

* **Bode of**  → read PM/GM.
* **Step of**  → rise/overshoot/settling; integrator ⇒ zero steady-state error.
* **Bode of**  → low-frequency small is good.
* **Control effort** → check torque limits; add anti‑windup if needed.

## 5) Worked Example

Given kg·m², N·m·s/rad, N·m/A:

* rad/s
* choose rad/s
* s

* s ⇒
* s

## 6) Python Snippet (Step & Bode)

import control as ctl, matplotlib.pyplot as plt, math  
  
# Example parameters  
J, b, K\_tau = 0.01, 0.001, 0.10  
omega\_c = 1.0  
T\_d = J/b  
Kp = b\*omega\_c/K\_tau  
Kd = Kp\*T\_d  
Ti = 8/omega\_c  
Ki = Kp/Ti  
tau\_d = 1/(10\*omega\_c)  
  
s = ctl.TransferFunction.s  
G = 1/(s\*(J\*s+b))  
C = Kp + Ki/s + (Kd\*s)/(1+tau\_d\*s)  
  
L = C\*K\_tau\*G  
T = ctl.feedback(L,1)  
S = 1/(1+L)  
  
gm, pm, wg, wp = ctl.margin(L)  
print(f"GM={gm:.2g} ({20\*math.log10(gm):.1f} dB) @ {wg:.3g} rad/s, PM={pm:.1f}° @ {wp:.3g} rad/s")  
  
t, y = ctl.step\_response(T)  
plt.figure(); plt.plot(t,y); plt.grid(True); plt.title("Closed-loop Step"); plt.xlabel("Time [s]"); plt.ylabel("Position [rad]")  
plt.figure(); ctl.bode(L, dB=True); plt.suptitle("Open-loop Bode"); plt.show()

## 7) Practical Notes

* Too slow? Increase → raise ; keep , reduce ; keep .
* Ringing / low PM? Lower or increase phase lead slightly (increase ); if integral is aggressive, increase .
* Noisy control? Increase or lower .
* Friction (Coulomb/stiction): treat as disturbance; stronger integral helps but watch overshoot/windup.
* Gearing & load reflection: if gear ratio ,
* Resonances/compliance: consider notch filters or add an inner velocity loop.
* Anti‑windup: clamp integrator or back‑calculate when torque saturates.