Cascade control for DC motor

1) Current loop : "Best performance / Overdamped" (aperiodic, PI for exact 1st-order)

2) Speed loop : Magnitude Optimum (~4.3% OS, P controller using inner lag)

3) Position loop : Symmetric Optimum (PI, corrected gain)

The script prints the tuning steps and overlays with MO/SO templates.

Position loop gain Kp_p = 1/(2 * Tmu2) for stability (was 1/(8*Tmu2^2))

Data:

ω_{0nom}, rad/s	M_{nom}, Nm	M_{st}, Nm	J_1, kgm^2	J_2, kgm^2	C_{12}	$k\Phi_f$	T_e ms	M_{L1}, Nm	M_{L2}, Nm
116.6	7.16	70	0.0077	0.0023	444	0,775	3.3	4.77	2.39

It contains three loops, each nested inside the next:

Loop	Controlled Variable	Controller Type	Bandwidth	Tuning Criterion
Inner loop	Armature current / torque	PI	Fastest	**Best performance / Overdamped (aperiodic)**
Middle loop	Speed	P	Medium	**Magnitude Optimum (MO)**
Outer loop	Position	PI	Slowest	**Symmetric Optimum (SO)**

```
clear; clc; close all;
s = tf('s');

%% ---- Motor & data  -----------------------------------
w0_nom = 116.6;          % rad/s (nominal speed)
M_nom  = 7.16;           % Nm
M_st   = 70;             % Nm (stall)
J1     = 0.0077;         % kg*m^2
J2     = 0.0023;         % kg*m^2
C12    = 444;            % N*m/rad (not used in rigid shaft demo)
kPhi   = 0.775;          % V*s/rad  (also Nm/A in SI)
Te     = 3.3e-3;         % s (electrical time constant)
ML1    = 4.77;           % Nm
ML2    = 2.39;           % Nm

J  = J1 + J2;            % total inertia
kT = kPhi;               % torque const [Nm/A]
```

```
ke = kPhi;                    % back-EMF const [V*s/rad]

% If exact R, L are known, set them. We use Te = L/R.
R  = 0.5;                     % Ohm
L  = Te*R;                    % H

%% ---- Primitive plants --------------------------------------------------
Gi = 1/(L*s + R);             % (A/V): current plant (ignoring back-EMF in
inner loop)
Gm = kT/(J*s);                % (rad/s)/A: mechanical plant

fprintf('\n=== STEP 1. CURRENT LOOP (aperiodic, best performance) ===\n');
```

```
=== STEP 1. CURRENT LOOP (aperiodic, best performance) ===
```

1) CURRENT LOOP – aperiodic / best performance

Criterion:

“Best performance” (aperiodic or critically damped) means the **closed loop is purely exponential**, i.e., no oscillation, no overshoot.

The loop time constant

τ_i is chosen small enough so this loop is **5–10× faster than the speed loop**.

```
% PDF "Overdamped process": closed loop should be 1st-order.
% Use PI zero to cancel plant pole: Ki_i/Kp_i = R/L  -> exact cancellation.
% Then CL: T_i(s) = Kp_i/(L*s + Kp_i) = 1/(1 + s*tau_i).

% Choose tau_i so current loop is clearly faster than speed loop.
% Start with tau_i = Te/3 (safe, realizable). You can change it to shape
bandwidth split.
tau_i = Te/3;

Kp_i  = L / tau_i;
Ki_i  = (R/L) * Kp_i;
Ci    = Kp_i + Ki_i/s;

T_i   = minreal( feedback(Ci*Gi, 1) );          % real inner CL
T_i_ref = 1/(1 + s*tau_i);                       % reference exponential
(template)

bw_i = bandwidth(T_i);
fprintf('Chosen tau_i = %.6f s,  Current loop BW ≈ %.1f Hz\n', tau_i,
bw_i/2/pi);
```

```
Chosen tau_i = 0.001100 s,  Current loop BW ≈ 144.3 Hz
```

```
fprintf('Gains:  Kp_i = %.4f,  Ki_i = %.4f  (zero at -R/L)\n', Kp_i, Ki_i);
```

```
Gains:  Kp_i = 1.5000,  Ki_i = 454.5455  (zero at -R/L)
```

2) SPEED LOOP – Magnitude Optimum

Magnitude Optimum - “balanced amplitude response” for robustness and speed.

It intentionally allows a small overshoot (≈4.3%) for faster speed recovery.

```
fprintf('\n=== STEP 2. SPEED LOOP (Magnitude Optimum) ===\n');
```

```
=== STEP 2. SPEED LOOP (Magnitude Optimum) ===
```

```
% Effective speed object includes inner current CL as a 1st-order lag.
%   T_i(s) ≈ 1/(1 + s*tau_i)
%   G_ob_speed(s) ≈ (kT/J) / (s * (1 + s*tau_i))
% For MO: Set Tmu_w = tau_i, use P controller Kp_w = 1 / (2 * K_ob * Tmu_w)
% This makes open-loop exactly 1 / (2 Tmu_w s (Tmu_w s + 1)), yielding
~4.3% OS.

K_ob = kT/J;                % DC gain of speed object excluding lag &
integrator
Tmu_w = tau_i;              % Magnitude Optimum choice: Tmu_w = tau_i

Kp_w = 1 / (2 * K_ob * Tmu_w);
Cw   = Kp_w;                % P controller (no integral needed for type-1
plant)

% Closed-loop speed (with real current CL)
G_speed_path_real = T_i * Gm;                         % Aref -> omega
Tw = minreal( feedback(Cw * G_speed_path_real, 1) );  % w_ref -> w

% MO template for overlay: Gc_MO(s) = 1 / (2 Tmu^2 s^2 + 2 Tmu s + 1)
Tw_ref = 1 / (2 * Tmu_w^2 * s^2 + 2 * Tmu_w * s + 1);

bw_w = bandwidth(Tw);
SI_w = stepinfo(Tw);
fprintf('MO parameters: K_ob = %.4f,  Tmu_w = %.6f s\n', K_ob, Tmu_w);
```

```
MO parameters: K_ob = 77.5000,  Tmu_w = 0.001100 s
```

```
fprintf('Speed P:       Kp_w = %.4f  (no Ki, uses inner loop lag for MO)
\n', Kp_w);
```

```
Speed P:       Kp_w = 5.8651  (no Ki, uses inner loop lag for MO)
```

```
fprintf('Result:        BW ≈ %.1f Hz, Overshoot = %.2f %% (target ~4.3%%)
\n', bw_w/2/pi, SI_w.Overshoot);
```

```
Result:        BW ≈ 102.2 Hz, Overshoot = 4.32 % (target ~4.3%)
```

3) POSITION LOOP – Symmetric Optimum

Symmetric Optimum gives **strong disturbance rejection** and **phase-margin balance**, typically yielding a **more oscillatory response** (~43% overshoot).

It's slower and less damped by design because the position loop naturally works on larger time constants.

```
fprintf('\n=== STEP 3. POSITION LOOP (Symmetric Optimum) ===\n');
```

```
=== STEP 3. POSITION LOOP (Symmetric Optimum) ===
```

```
% With inner speed CL ≈ 1/(1 + s*Tmu2) where for cascaded MO: Tmu2 ≈ 2*Tmu_w
Tmu2 = 2 * Tmu_w;

% The position plant is integrator on top of speed CL:
%   G_ob_pos(s) = Tw(s) / s  ≈ 1 / (s * (1 + s*Tmu2))
% Symmetric Optimum PI: Ti_p = 4*Tmu2; Kp_p = 1/(2*Tmu2 * K_ob_pos)
% Here K_ob_pos = 1 (unit scaling between position controller and speed
ref).
Ti_p = 4 * Tmu2;
K_ob_pos = 1;
Kp_p = 1 / (2 * Tmu2 * K_ob_pos);
Ki_p = Kp_p / Ti_p;

Cp   = Kp_p + Ki_p / s;
Tpos = minreal( feedback( Cp * (Tw / s), 1 ) );

% SO template: Gc_SO(s) = (4 Tmu s + 1) / (8 Tmu^3 s^3 + 8 Tmu^2 s^2 + 4
Tmu s + 1)
Tpos_ref = (4 * Tmu2 * s + 1) / (8 * Tmu2^3 * s^3 + 8 * Tmu2^2 * s^2 + 4 *
Tmu2 * s + 1);

SI_p = stepinfo(Tpos);
fprintf('SO parameters: Tmu2 = %.6f s\n', Tmu2);
```

```
SO parameters: Tmu2 = 0.002200 s
```

```
fprintf('Position PI:   Kp_p = %.4f,  Ki_p = %.4f  (Ti_p = %.6f s)\n',
Kp_p, Ki_p, Ti_p);
```

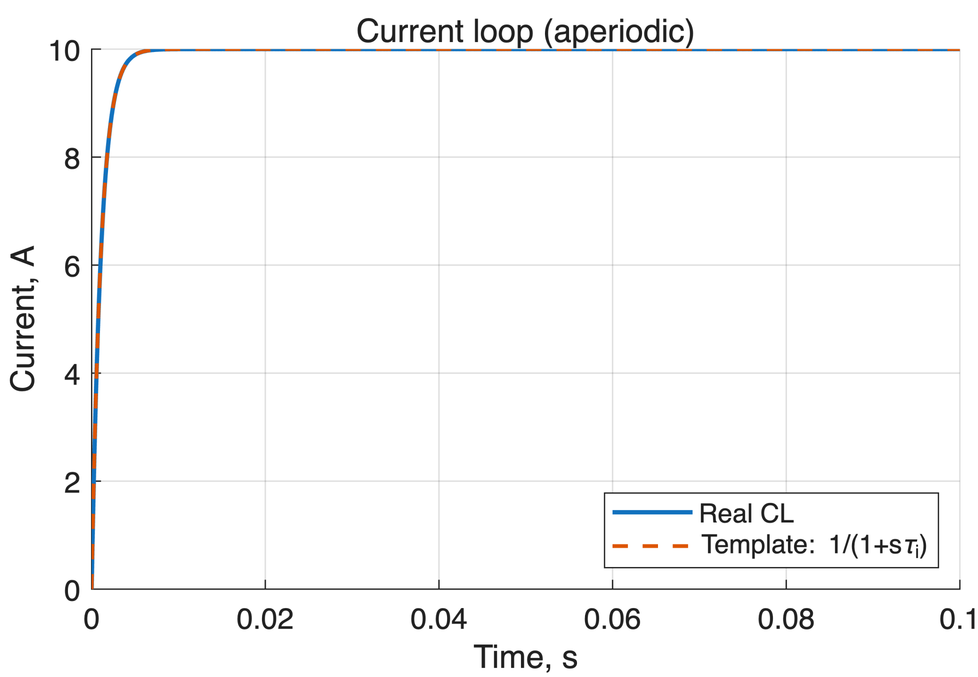
```
Position PI:   Kp_p = 227.2727,  Ki_p = 25826.4463  (Ti_p = 0.008800 s)
```

```
fprintf('Result:        Overshoot ≈ %.2f %%, Ts ≈ %.3f s (SO is
normally more oscillatory than MO, target ~43%% OS)\n', SI_p.Overshoot,
SI_p.SettlingTime);
```

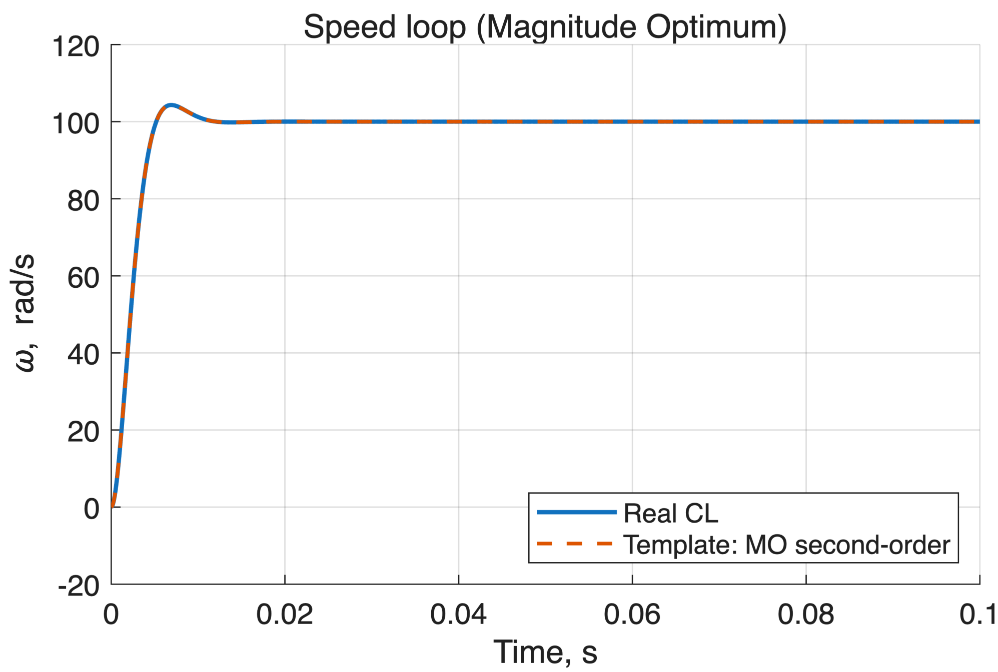
```
Result:        Overshoot ≈ 53.71 %, Ts ≈ 0.030 s (SO is normally more oscillatory than MO, target ~43% OS)
```

STEP RESPONSES + TEMPLATES OVERLAID

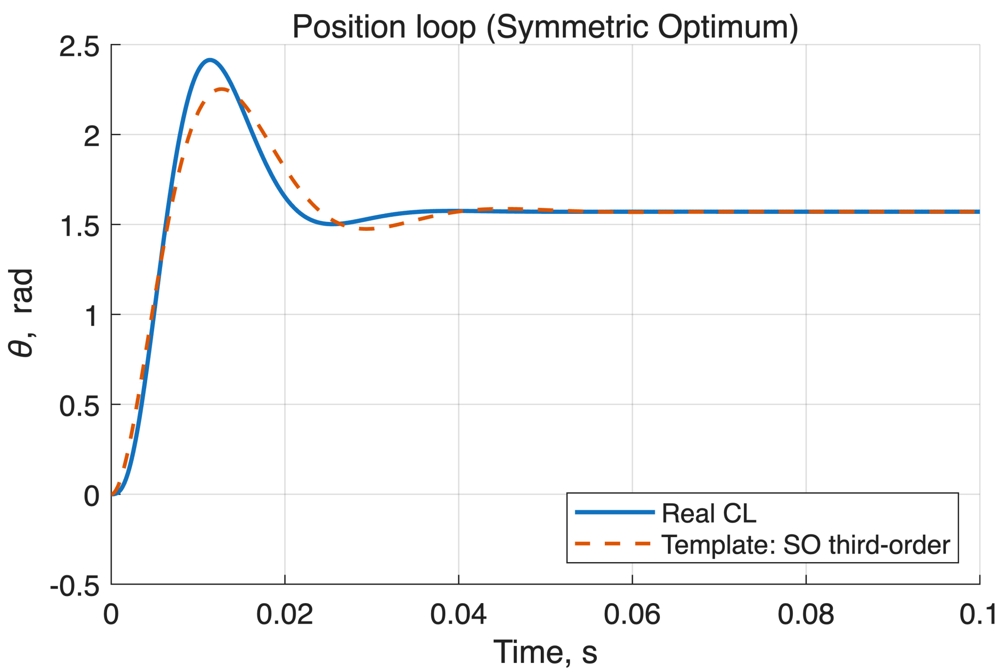
```
t = 0:1e-4:0.1;

% Current loop (10 A)
i_ref = 10 * ones(size(t));
yi    = lsim(T_i, i_ref, t);
yi_ref= lsim(T_i_ref, i_ref, t);

figure; hold on; grid on;
plot(t, yi, 'LineWidth',1.6);
plot(t, yi_ref, '--', 'LineWidth',1.4);
title('Current loop (aperiodic)'); xlabel('Time, s'); ylabel('Current, A');
legend('Real CL','Template: 1/(1+s\tau_i)','Location','southeast');
```




```

% Speed loop (100 rad/s)
w_ref = 100 * ones(size(t));
yw    = lsim(Tw, w_ref, t);
yw_ref= lsim(Tw_ref, w_ref, t);

figure; hold on; grid on;
plot(t, yw, 'LineWidth',1.6);
plot(t, yw_ref, '--', 'LineWidth',1.4);
title('Speed loop (Magnitude Optimum)'); xlabel('Time, s'); ylabel('\omega,
rad/s');
legend('Real CL','Template: MO second-order','Location','southeast');
```




```
% Position loop (90 deg = \pi/2 rad)
theta_ref = (pi/2) * ones(size(t));
yth    = lsim(Tpos, theta_ref, t);
yth_ref= lsim(Tpos_ref, theta_ref, t);

figure; hold on; grid on;
plot(t, yth, 'LineWidth',1.6);
plot(t, yth_ref, '--', 'LineWidth',1.4);
title('Position loop (Symmetric Optimum)'); xlabel('Time, s');
ylabel('\theta, rad');
legend('Real CL','Template: SO third-order','Location','southeast');
```



Console summary

```
fprintf('\n=== SUMMARY ===\n');
```

```
=== SUMMARY ===
```

```
fprintf('Current loop   : tau_i = %.5f s  -> strictly exponential (no OS)
\n', tau_i);
```

```
Current loop   : tau_i = 0.00110 s  -> strictly exponential (no OS)
```

```
fprintf('Speed loop     : MO with Tmu = %.5f s (P ctrl) -> target OS ≈ 4.3%
%, got %.2f%%\n', Tmu_w, SI_w.Overshoot);
```

```
Speed loop     : MO with Tmu = 0.00110 s (P ctrl) -> target OS ≈ 4.3%, got 4.32%
```

```
fprintf('Position loop : SO with Tmu2 = %.5f s (Ti_p=4*Tmu2) -> target OS ≈
43%%, got %.2f%%\n\n', Tmu2, SI_p.Overshoot);
```

```
Position loop : SO with Tmu2 = 0.00220 s (Ti_p=4*Tmu2) -> target OS ≈ 43%, got 53.71%
```

Loop Hierarchy (Bandwidth Separation)

script correctly ensures:

$$f_{current} > 5\, f_{speed} > 5\, f_{position}$$

This guarantees proper **decoupling** — each outer loop “sees” the inner loops as nearly instantaneous.