Práctica de Laboratorio 3: Diseño de controladores por el método de Bode.

Elías Álvarez
Carrera de Ing. Electrónica
Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay
Email: elias.alvarez@universidadcatolica.edu.py

Tania Romero
Carrera de Ing. Electrónica
Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay
Email: tania.romero@universidadcatolica.edu.py

Docente: Lic. Montserrat González
Facultad de Ingeniería
Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay

Docente: PhD. Enrique Vargas
Facultad de Ingeniería
Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay

I. Introducción

El diseño de controladores en el dominio de la frecuencia constituye una de las técnicas más utilizadas en la ingeniería de control debido a su claridad gráfica y a la posibilidad de ajustar directamente márgenes de ganancia y de fase. En esta práctica de laboratorio se aplica el método de Bode para modificar la dinámica de una planta, de modo a garantizar estabilidad y un comportamiento transitorio deseado.

El objetivo principal es comprender cómo el diseño de compensadores influye en la respuesta de un sistema de control en tiempo discreto, evaluando tanto el margen de fase como el error en estado estacionario frente a entradas de tipo rampa. Asimismo, se busca contrastar los resultados obtenidos en Matlab con la implementación real en un PSoC, resaltando la importancia de la simulación previa y del análisis de la respuesta en frecuencia para un diseño robusto.

II. Objetivos

- Diseñar un controlador que modifique la dinámica de la planta para satisfacer condiciones específicas de la respuesta transitoria del sistema de control en lazo cerrado.
- Garantizar que el sistema regulado sea estable.
- Observar y analizar los efectos del controlador en el comportamiento del sistema.
- Considerar diferentes métodos para el ajuste de los parámetros del controlador y analizar los resultados.
- Diseñar el sistema de control en Matlab e implementar la ecuación en diferencias en el PSoC.

III. Materiales

- PC con Matlab.
- Planta analógica.
- Sistema de adquisición en PSoC.

IV. Teoría

Se recomienda consultar la referencia:
K. Ogata, *Sistemas de Control en Tiempo Discreto*, págs. 204–225, donde se desarrollan las bases teóricas necesarias para el diseño de compensadores mediante el diagrama de Bode y la evaluación de estabilidad en sistemas discretos.

V. Desarrollo

V-A. Modelado del Sistema

V-A1. Obtención de la función de transferencia en lazo abierto: La planta puede interpretarse como la conexión en cascada de dos filtros activos de primer orden. Cada uno posee la misma topología: un amplificador operacional en configuración inversora cuya impedancia de realimentación está compuesta por una resistencia en paralelo con un capacitor.

V-A1a. Impedancia de realimentación.: Para el paralelo $R_f \parallel C_f$, se obtiene:

$$Z_f \;=\; R_f \parallel \frac{1}{sC_f} \;=\; \frac{R_f}{1 + sR_fC_f} \tag{1}$$

El sistema trabaja sobre una tensión de referencia en continua $V_{\text{cc}}/2 = 2.5$ V. En los cálculos posteriores se toma dicho valor como punto de referencia.

V-A1b. Ganancia de una etapa (entrada inversora).: Con $V_{\text{ref}} = 0$, se cumple el cortocircuito virtual ($V_p = V_n = 0$). Aplicando KCL en el nodo inversor:

$$\frac{V_i}{R_i} \;=\; \frac{-V_o}{Z_f} \quad\Rightarrow\quad \frac{V_o}{V_i} \;=\; -\,\frac{Z_f}{R_i}$$

y reemplazando (1):

$$\left.\frac{V_o}{V_i}\right|_{V_{\text{ref}}=0} = -\,\frac{R_f}{R_i}\,\frac{1}{1 + sR_fC_f} \tag{2}$$

V-A1c. Encadenamiento de etapas.: Como ambas etapas AO1 y AO2 responden a la forma (2), la ganancia total en lazo abierto resulta del producto de sus transferencias:

$$G_{\text{ol}}(s) = \Big(- \frac{Z_{f1}(s)}{R_{i1}} \Big) \Big(- \frac{Z_{f2}(s)}{R_{i2}} \Big)$$

V-A1d. Resultados numéricos.: A partir de la respuesta de la figura 1 se obtuvieron:

- **Tiempo de subida:** $t_r \approx 0.0332$ s (33.2 ms)
- **Tiempo de establecimiento (2 %):** $t_s \approx 0.0603$ s (60.3 ms)
- **Frecuencia natural estimada:** $\omega_n \approx 54.2$ rad/s

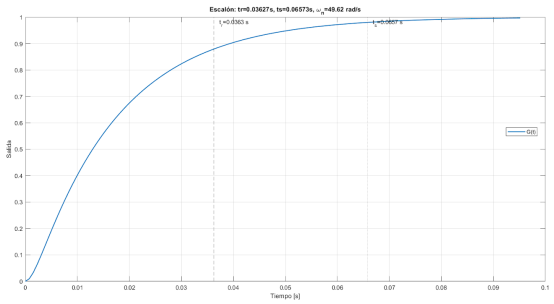

Figura 1: Escalón en lazo abierto del sistema.

V-B. Discretización de la Planta

V-B1. Se debe elegir un tiempo de muestreo adecuado: La planta a discretizar está dada por:

$$G(s) = \frac{K}{(s+a)(s+b)}.$$

Para discretizarla, se utiliza el método de la *transformación de la respuesta al impulso*. La relación general es:

$$G(z) = (1 - z^{-1})\, \mathcal{Z} \left\{ \mathcal{L}^{-1} \left(\frac{G(s)}{s} \right) \right\}. \tag{3}$$

A partir de la descomposición en fracciones parciales, se obtiene:

$$\frac{G(s)}{s} = \underbrace{\frac{K}{ab}}_{\delta} \cdot \frac{1}{s} + \underbrace{\frac{K}{b^2 - ab}}_{\beta} \cdot \frac{1}{s+b} + \underbrace{\frac{K}{a^2 - ab}}_{\alpha} \cdot \frac{1}{s+a}, \tag{4}$$

donde los coeficientes α, β y δ corresponden a los términos de la expansión.

Transformada-Z: Aplicando la transformada-Z a cada término se obtiene:

$$\mathcal{Z} \left[\frac{G(s)}{s} \right] = \delta \frac{z}{z-1} + \beta \frac{z}{z - e^{-bT}} + \alpha \frac{z}{z - e^{-aT}}.$$

Finalmente, la planta discretizada puede escribirse como:

$$G(z) = \frac{\gamma z^2 - \theta z + \psi}{(z - e^{-aT})(z - e^{-bT})}, \tag{5}$$

donde los parámetros γ, θ y ψ se expresan en función de los coeficientes de la fracción parcial (4):

$$\gamma = \alpha + \beta + \delta$$
$$\theta = \alpha + \beta + \delta + \alpha e^{-bT} + \beta e^{-aT} + \delta\big(e^{-aT} + e^{-bT}\big)$$
$$\psi = \alpha e^{-bT} + \beta e^{-aT} + \delta e^{-aT} e^{-bT}$$

Se tiene la siguiente ecuación, el tiempo de muestreo utilizado para este laboratorio será el mismo que el del laboratorio 2, $T = 1.25ms$, $a = \frac{1}{81\,000 \cdot 200 \cdot 10^{-9}}$ y $b = \frac{1}{15 \cdot 10^3 \cdot 100 \cdot 10^{-9}}$, y considerando $K = 1$ se tienen ya calculados los valores de α, β y δ:

$$\delta = \frac{K}{ab} \approx 24.3 \times 10^{-6}$$
$$\beta = \frac{K}{b^2 - ab} \approx 2.48 \times 10^{-6}$$
$$\alpha = \frac{K}{a^2 - ab} \approx -26.78 \times 10^{-6}.$$

con los siguientes valores se tiene G(z):

$$G(z) = \frac{0.22(z + 0.74)}{(z - 0.933)(z - 0.44)} \tag{6}$$

Puesto a que los métodos convencionales de la *respuesta en frecuencia* no se aplican en el plano $\mathcal{Z}$, volvemos al plano ω usando la siguiente ecuación:

$$Z = \frac{1 + (\frac{T}{2})\omega}{1 - (\frac{T}{2})\omega} \tag{7}$$

Sustituyendo la z de la ecuación 6 con la ecuación 7:

$$G(w) = \frac{-2.41 \times 10^{-9}(w + 10.675 \times 10^3)(w - 1.6 \times 10^3)}{1.089 \times 10^{-6}(w + 55.372)(w + 622.22)} \tag{8}$$

Con esto obtenemos la función de lazo abierto en el plano ω. A continuación, trazamos las curvas de Bode. Para ello, es necesario recordar que al hacer tender $\omega \to jv$, donde v representa la *frecuencia ficticia* y ω la *frecuencia real*, estando ambas relacionadas por:

$$w\big|_{w=jv} = jv = \frac{2(z-1)}{T(z+1)}\bigg|_{z=e^{j\omega T}} \tag{9}$$

De esta forma se obtiene la relación:

$$v = \frac{2}{T} \tan\left(\frac{\omega T}{2} \right) \tag{10}$$

V-B2. Comparar los resultados obtenidos con la simulación en Matlab: La planta discretizada obtenida en Matlab es:

$$H(z) = \frac{21.596 \times 10^{-3}\,(z + 0.7419)}{(z - 0.9333)(z - 0.436)}$$

Como puede observarse, la expresión de la ecuación 6:

$$G(z) = \frac{22 \times 10^{-3}\,(z + 0.74)}{(z - 0.933)(z - 0.44)}$$

presenta una buena aproximación a los valores calculados.

Por otro lado, la planta en el plano ω obtenida a partir de Matlab es:

$$H(w) = \frac{-2.0075 \times 10^{-3}\,(s + 10.84 \times 10^3)(s - 1606)}{(s + 630.8)(s + 55.41)}$$

mientras que la expresión calculada resulta (8):

$$G(w) = \frac{-2.21 \times 10^{-3}\,(w + 10.675 \times 10^3)(w - 1.6 \times 10^3)}{(w + 55.372)(w + 622.22)}$$

la cual también presenta valores aproximados.

V-C. Modificación de la Dinámica de la Planta

V-C1. Diseñar un compensador que logre un margen de fase de 60°*:* Para el diseño, en primer lugar se transforma la planta del plano $\mathcal{S}$ al plano $\mathcal{Z}$, y posteriormente al plano $\mathcal{W}$. La primera transformación se realiza empleando la relación presentada en la ecuación 3.

Luego, para pasar del plano $\mathcal{Z}$ al plano $\mathcal{W}$, se utiliza la relación de Tustin, expresada en la ecuación 7, tal como se explicó en el ítem anterior (V-B).

Diseño de compensadores – análisis preliminar:

En primer lugar, se analizan los márgenes de fase y de ganancia de la planta sin compensar.

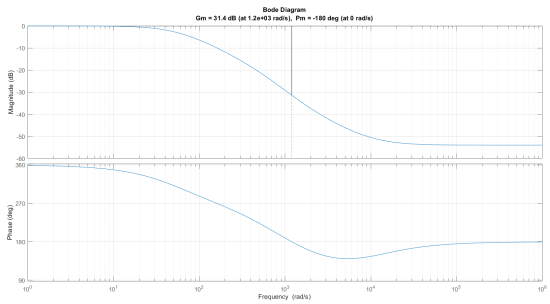

Figura 2: Diagrama de Bode de la planta sin compensación.

Se observa que el margen inicial de ganancia es de 31,400 dB, lo que permite incrementar el valor de la ganancia K.

V-C1a. Primer intento – Compensador Lead: Por criterio de diseño, se selecciona un margen de ganancia aceptable, correspondiente a valores superiores a 10,000 dB. A partir de esta condición se calcula la ganancia:

$$|G(\omega)| = 20\log(K) = 31.4 - 10 \quad \Rightarrow \quad K = 10^{21.4/20}.$$

La nueva ganancia resulta:

$$K = 11.149.$$

Con este valor de ganancia se obtiene el siguiente diagrama de Bode:

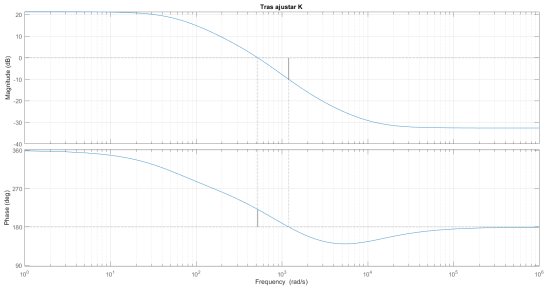

Figura 3: Diagrama de Bode de la planta con la nueva ganancia.

En la figura 3 se aprecia el desplazamiento producido por la nueva ganancia $K = 11.7057$, con los siguientes resultados:

- **Margen de ganancia:** $GM = 10{,}000\,\text{dB}$
- **Margen de fase:** $PM = 41.13°$
- **Frecuencia de cruce de ganancia:** $\omega_{cg} = 1.2 \times 10^3$
- **Frecuencia de cruce de fase:** $\omega_{cp} = 523$

Se observa que el margen de fase obtenido es $PM = 41.13°$, lo que representa una diferencia respecto al valor solicitado de 60°:

$$\Delta PM = 60° - 41.13° = 18.87° \approx 20°.$$

Denominamos a este valor ϕ. Como aún faltan aproximadamente 20° para alcanzar el margen de fase deseado, se decide emplear un compensador de adelanto (Lead).

Conociendo la frecuencia de cruce actual $\omega_{cp,\text{now}}$ y la fase adicional requerida ($\phi \approx 20°$), se procede al diseño del compensador, implementando las siguientes ecuaciones en Matlab:

$$\alpha = \frac{1 - \sin(\phi)}{1 + \sin(\phi)}, \qquad T = \frac{1}{\omega_{cp,\text{now}}\sqrt{\alpha}}$$

$$C_0 = \frac{1 + Ts}{1 + \alpha Ts}, \qquad g = \frac{1}{|G(j\omega_c)C_0(j\omega_c)|}$$

$$C = g \cdot C_0$$

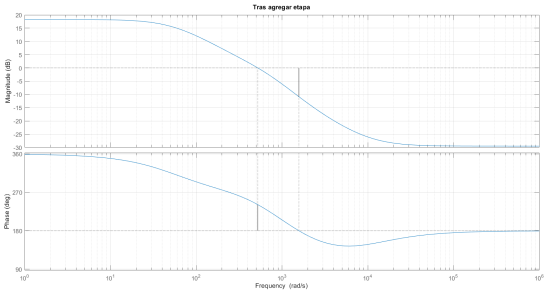

Figura 4: Diagrama de Bode con el primer compensador diseñado.

Los nuevos resultados obtenidos con este compensador son los siguientes:

- **Margen de ganancia:** $GM = 10{,}830\,\text{dB}$
- **Margen de fase:** $PM = 61.12°$
- **Frecuencia de cruce de ganancia:** $\omega_{cg} = 1.57 \times 10^3$
- **Frecuencia de cruce de fase:** $\omega_{cp} = 523$

Datos del compensador: El compensador obtenido puede expresarse de distintas formas.

Forma explícita con todos los decimales

$$C(z) = \frac{1.18473440049107\,z - 0.737363460561365}{z - 0.374311732736374}$$

Forma explícita con decimales redondeados

$$C_1(z) = \frac{1.185\,z - 0.7374}{z - 0.3743}$$

El período de muestreo asociado es:

$$T_s = 0.001245213061220\ \text{s}$$

Forma cero-polo-ganancia (ZPK)
Con redondeo:

$$C_1(z) = 1.1847 \cdot \frac{(z - 0.6224)}{(z - 0.3743)}$$

Con todos los decimales:

$$C(z) = 1.184734400491065 \cdot \frac{(z - 0.622387144541200)}{(z - 0.374311732736374)}$$

donde los parámetros identificados son:

- Cero: $Z = 0.622387144541200$
- Polo: $P = 0.374311732736374$
- Ganancia: $K = 1.184734400491065$
- Sample time: $t_s = 0.0012452s$

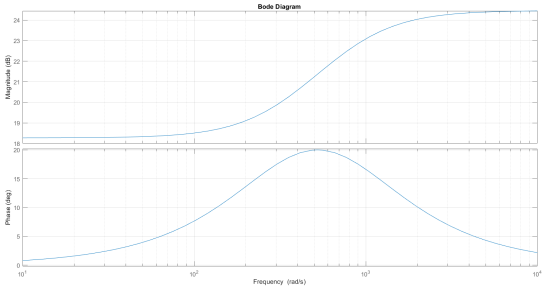

Figura 5: Diagrama de Bode del primer compensador diseñado.

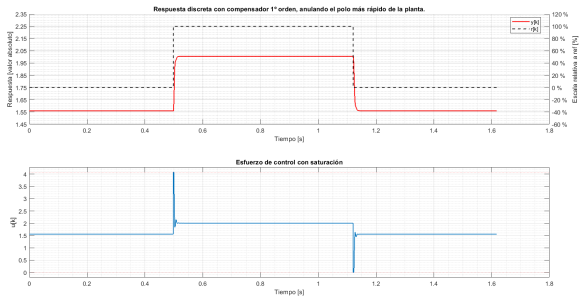

Figura 6: Esfuerzo del primer compensador diseñado.

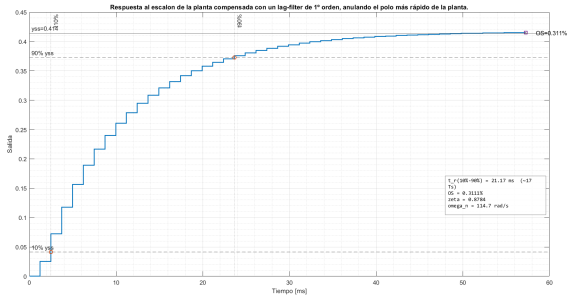

Figura 7: El step del primer compensador diseñado.

El resultado obtenido para este diseño no es el más óptimo. Con una ganancia $K = 1.18$, el sistema presenta un error significativo debido a que es de tipo 0. Al aplicar una entrada escalón, el error en estado estacionario calculado es:

$$e = \frac{1}{1 + K_p} = 45.8\,\%.$$

Este valor resulta demasiado elevado. Además, al observar la figura 6, se aprecia que el esfuerzo de control presenta saturación tanto al inicio como al final de la respuesta.

V-C1b. Segundo intento – Compensador Proporcional: A partir del análisis del diagrama de Bode de la planta sin compensar (figura 2), se determina la frecuencia en la que el margen de fase alcanza el valor deseado de 60°.

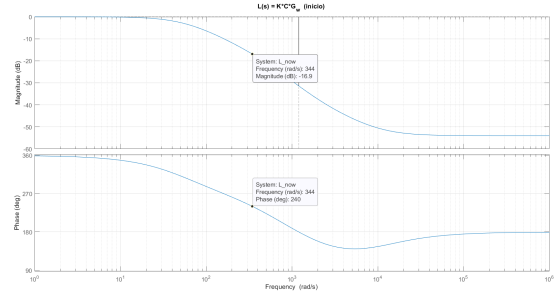

Figura 8: Análisis de la ganancia de la planta.

Como se aprecia en la figura 8, la frecuencia correspondiente a un margen de fase de 60° es de aproximadamente 344,000 Hz. El margen de ganancia en dicha frecuencia es de 16,900 dB. De este modo, si se incrementa la ganancia en ese valor, es posible forzar el cruce de tal manera que se alcance el margen de fase requerido:

$$20\log(K) = MG_{\text{inicial}} - MG_{\text{deseado}}.$$

Dado que $MG_{\text{inicial}} - MG_{\text{deseado}} = 16.9$, la nueva ganancia es aproximadamente:

$$K \approx 7.$$

Con esta ganancia se obtienen los siguientes resultados:

- **Margen de ganancia:** $GM = 14{,}470\,\text{dB}$
- **Margen de fase:** $PM = 60.28°$
- **Frecuencia de cruce de ganancia:** $\omega_{cg} = 1.2 \times 10^3$
- **Frecuencia de cruce de fase:** $\omega_{cp} = 344$

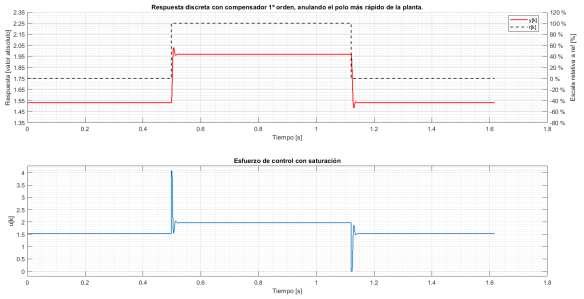

Figura 9: Esfuerzo de control con el compensador proporcional.

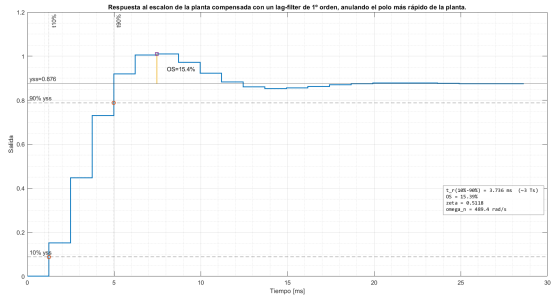

Figura 10: Respuesta al escalón con el compensador proporcional.

Con la nueva ganancia K se calcula el error en estado estacionario:

$$e \approx \frac{1}{1+K_p} \approx 0.125 \quad \Rightarrow \quad 12.5\,\%.$$

Este error es considerablemente menor al obtenido con el primer compensador. Sin embargo, la respuesta presenta un *overshoot* más pronunciado, lo que se traduce en esfuerzos de control abruptos al inicio, como puede observarse en la figura 9. Esto provoca la saturación del VDAC incluso con escalones pequeños.

Por lo tanto, se concluye que este segundo diseño tampoco constituye una solución óptima.

V-C2. El compensador debe permitir el seguimiento de una entrada rampa con un error en estado estable igual a $1/K_v$*:* Para cumplir con este requerimiento, es necesario diseñar un compensador que incorpore un polo en el origen. De esta forma, el sistema podrá alcanzar un error en estado estacionario finito frente a una entrada rampa.

Los márgenes iniciales obtenidos con este compensador se muestran en la figura 11.

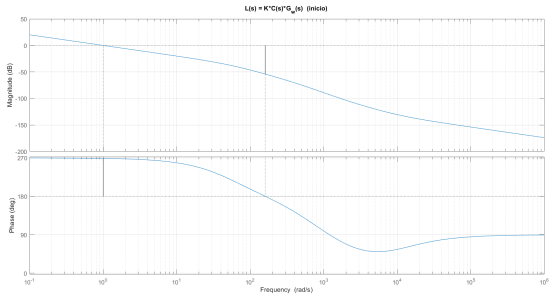

Figura 11: Diagrama de Bode con Margen de Ganancia inicial.

Como criterio de diseño, se establece un margen de ganancia deseada es de 10,000 dB. Por lo tanto, se debe determinar el valor de la ganancia K que cumpla con esta condición.

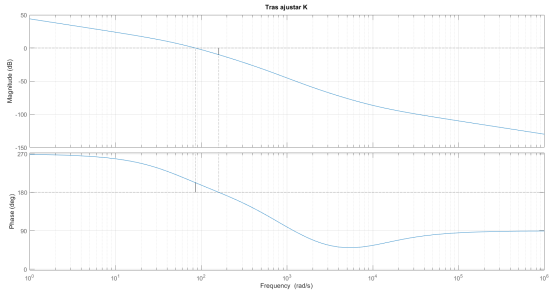

Figura 12: Diagrama de Bode nuevo de margen de ganancia (GM).

Con el nuevo margen de ganancia se obtienen los siguientes valores a partir de la figura 12:

- **Ganancia:** $K = 158.932$
- **Margen de ganancia:** $GM = 10{,}000\,\text{dB}$
- **Margen de fase:** $PM = 22.56°$
- **Frecuencia de cruce de ganancia:** $\omega_{cg} = 160$
- **Frecuencia de cruce de fase:** $\omega_{cp} = 85.7$

Se observa que el margen de fase obtenido es de 22.56°, por lo que para alcanzar el criterio de 60° aún faltan

aproximadamente 37.44°. En consecuencia, se incorpora un compensador de adelanto (Lead) con el objetivo de aportar la fase adicional requerida.

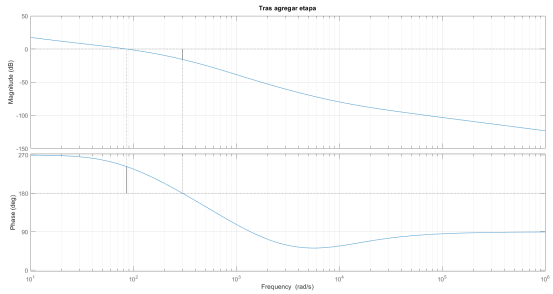

Figura 13: Diagrama de Bode del Compensador - Etapa Lead.

V-C2a. Datos del compensador C_2: El compensador obtenido puede expresarse de las siguientes formas:

Forma explícita

$$C_2(z) = \frac{0.1856\,z^2 + 0.009526\,z - 0.1761}{z^2 - 1.805\,z + 0.805}$$

con un período de muestreo asociado de:

$$T_s = 0.0012452 \text{ s}.$$

Forma cero–polo–ganancia (ZPK) – redondeada

$$C_2(z) = 0.18562 \cdot \frac{(z+1)(z-0.9487)}{(z-1)(z-0.805)}.$$

Los parámetros identificados son:

- **Ceros:** $Z = \{-1.0,\ 0.948680356607820\}$
- **Polos:** $P = \{1.0,\ 0.805044751405714\}$
- **Ganancia:** $K = 0.185620357009816$
- **Período de muestreo:** $T_s = 0.0012452$ s

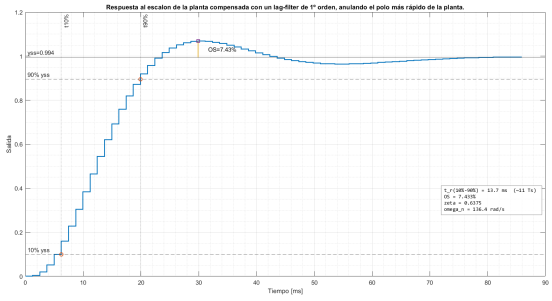

Figura 14: Diagrama de Bode del Compensador - Etapa Lead.

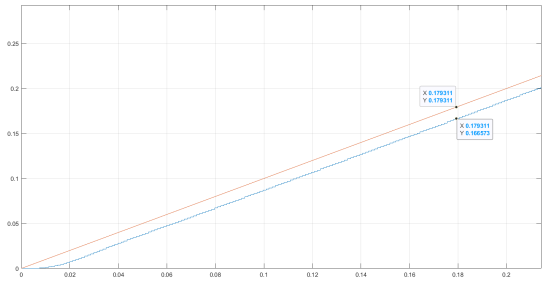

Figura 15: Diagrama de Bode del Compensador - Etapa Lead.

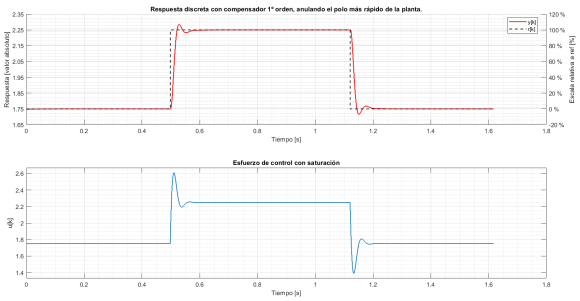

Figura 16: Diagrama de Bode del Compensador - Etapa Lead.

V-D. *Implementación del Sistema*

La implementación se realizó en plataforma PSoC, ejecutando el controlador discreto a período fijo de muestreo $T = 1{,}245$ ms (promediado). Se programó un firmware que permite: (i) abrir/cerrar el lazo para ensayos en lazo abierto, (ii) imponer distintos valores mínimos y máximos para el escalón y (iii) cambiar el período del escalón desde el puerto serie. El esfuerzo de control se genera con un `VDAC8` hacia la planta (referenciada a VDDA/2) y la salida de la planta se adquiere con un ADC-SAR para cerrar el lazo. La referencia es generada mediante software.

V-D1. Ecuación en diferencias y saturación: Cada compensador discreto se implementa a partir de la forma:

$$C(z) = \frac{K\left(E_0 + E_1 z^{-1} + E_2 z^{-2}\right)}{U_0 + U_1 z^{-1} + U_2 z^{-2}}. \tag{11}$$

Sabiendo entonces que $\mathcal{Z}^{-1}\{C_{den}(z)\,U(z)\} = \mathcal{Z}^{-1}\{K C_{num}(z)\,E(z)\}$, se llega a la ecuación en diferencias:

$$\begin{aligned} &U_0\,u[k] + U_1\,u[k-1] + U_2\,u[k-2] = \\ &K\Big(E_0\,e[k] + E_1\,e[k-1] + E_2\,e[k-2]\Big), \end{aligned} \tag{12}$$

y despejando $u[k]$:

$$\begin{aligned} u[k] \;=\; \frac{1}{U0}[&-U_1\,u[k-1] \;-\; U_2\,u[k-2] + \\ &K\left(E_0\,e[k] + E_1\,e[k-1] + E_2\,e[k-2]\right)] \end{aligned} \tag{13}$$

y luego se aplica saturación (montaje: `SAT_MIN` = 0,000 V, `SAT_MAX` = 4,080 V):

$$u_{DAC}[k] = \begin{cases} \texttt{SAT\_MAX}, & \text{si } u[k] > \texttt{SAT\_MAX}, \\ \texttt{SAT\_MIN}, & \text{si } u[k] < \texttt{SAT\_MIN}, \\ u[k], & \text{en otro caso.} \end{cases} \quad (14)$$

Finalmente se escribe al DAC: `VDAC8_SetValue(volt_to_dac(u_sat));` en firmware, las ecuaciones (13) y (14) se calculan en la función `actualizarEsfuerzo()`.

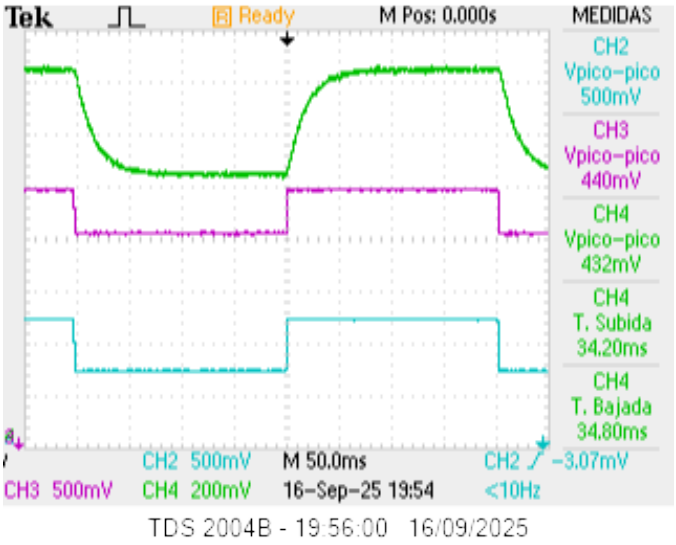

Figura 17: Comparación de un VDAC sin carga (traza cian) vs. VDAC cargado (traza violeta) con la planta y respuesta de la planta (traza verde) a la excitación en lazo abierto.

V-D2. Modo lazo abierto y parametrización: Para verificar la planta y *debuguear* la cadena de adquisición/actuación, se incorporó un modo de **lazo abierto** que *bypassea* el compensador y permite inyectar una referencia (escalón) directamente al DAC. La amplitud, los límites mínimo/máximo y el período del escalón se ajustan *online* por comandos serie.

V-D3. Observaciones de hardware (carga del VDAC): En ensayos de lazo abierto se observó **error de carga** en el VDAC al excitar directamente la planta (Fig. 17), por lo que se propuso colocar primero la etapa con resistencia de entrada 82,000 kΩ y luego la de 15,000 kΩ mitigó parcialmente el problema (Fig. 17). Con la de 15,000 kΩ al inicio, la excursión quedó limitada a ∼ 250,000 mV–300,000 mV pico a pico, frente a ∼ 500,000 mV p–p del DAC en vacío. Tras reordenar, la excursión aumentó hasta ∼ 440,000 mV p–p (Fig. 17). **Solución propuesta para siguientes laboratorios:** añadir un *buffer* a la salida del VDAC; desacoplar VDDA/2 y alimentación con capacitores de 1,000 µF a 10,000 µF (y cerámicos 100,000 nF a 470,000 nF en paralelo); utilizar el el modo *bypass/buffered* de los DAC/ADC, manteniendo el cambio hecho con el orden de las etapas de la planta para aumentar la impendancia vista desde la entrada,

V-D4. Desempeño medido vs. esperado: El tiempo de subida medido fue 34,200 ms frente a ∼ 40,000 ms estimados por simulación. La diferencia concuerda con tolerancias de componentes (±10,000 % en resistencias y hasta ±20,000 % en capacitores) y con la atenuación observada en la cadena de actuación (de 440,000 mV a 432,000 mV). Aun con dichas limitaciones, la dinámica global (parámetros y forma de la respuesta) se mantuvo alineada con el diseño en Matlab.

V-D5. Compensadores implementados: Se cargaron en firmware tres compensadores: (i) proporcional, (ii) proporcional con adelanto de primer orden y (iii) adelanto con integrador para seguimiento de rampa y ESS = 0 para el escalón . Las respuestas temporales cada uno se muestran en las Figs. 18, 19 y 20 respectivamente.

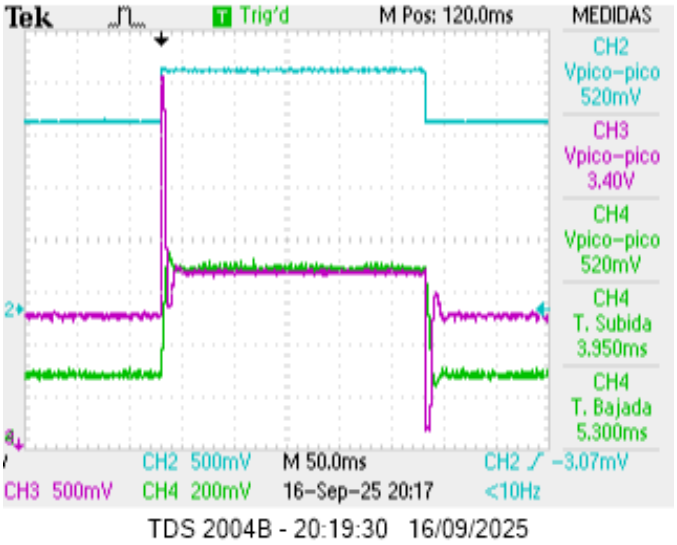

Figura 18: Compensador proporcional: referencia tipo escalon (traza cian), esfuerzo (traza violeta) y salida de la planta (traza verde).

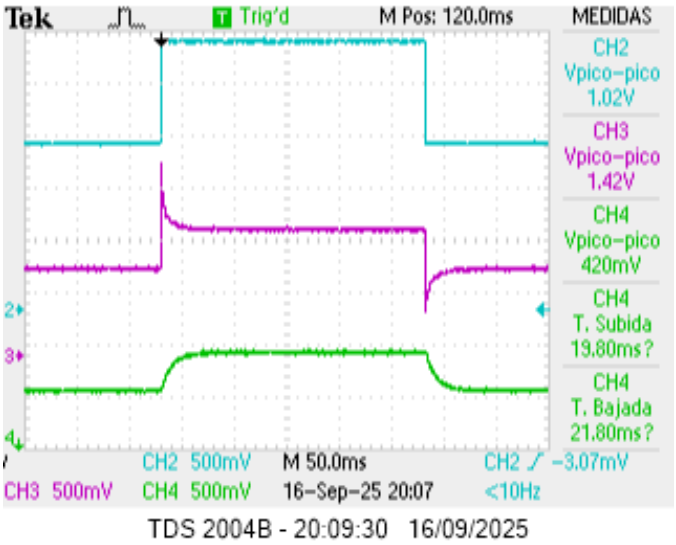

Figura 19: Compensador adelanto: referencia tipo escalon (traza cian), esfuerzo (traza violeta) y salida de la planta (traza verde).

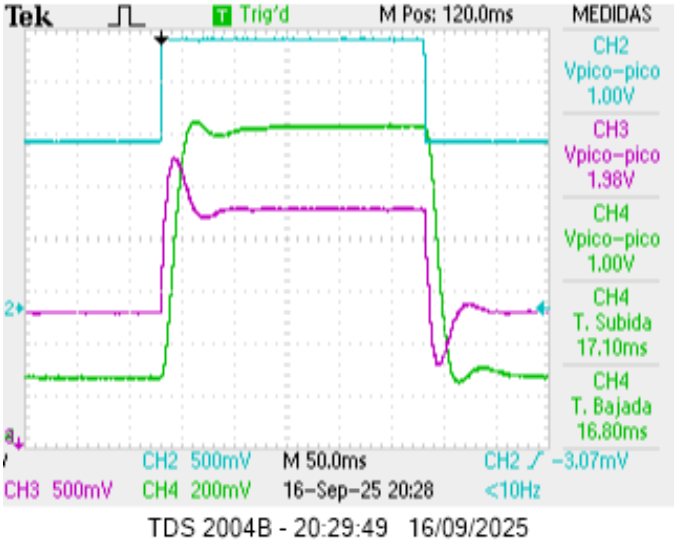

Figura 20: Compensador de adelanto + integrador: referencia tipo escalon (traza cian), esfuerzo (traza violeta) y salida de la planta (traza verde).

VI. Resultados

VI-A. Respuestas temporales (simulado vs. experimental)

En esta sección se presentan comparaciones *en el dominio del tiempo* entre simulación y experimento para: planta sin compensar, C1 (proporcional), C2 (lead), C3 (integrador+lead). En cada figura se incluyen las métricas *RMSE*, *NRMSE* y $e_{\text{máx}}$ (definidas en las ecuaciones (15)–(17)). Se muestran versiones *con* y *sin* remoción de offset para evidenciar el descalce de referencia señalado en la implementación.

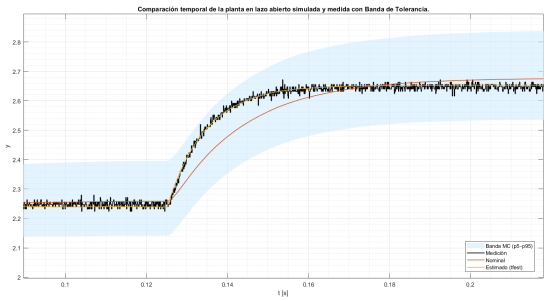

Figura 21: Planta sin compensación en lazo abierto: respuesta al escalón (simulado vs. experimental, con bandas de tolerancia para $\sigma = \frac{tolerance}{3}$).

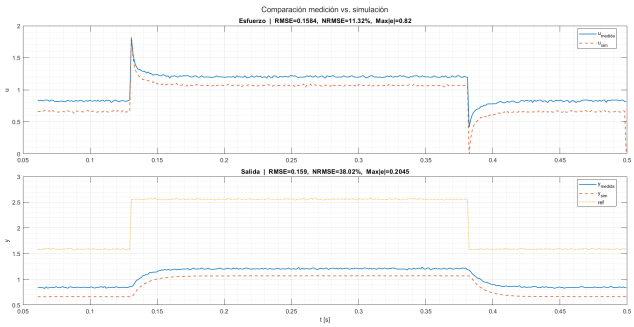

Figura 22: Compensador de adelanto: respuesta al escalón y esfuerzo (simulado vs. experimental, *con* offset).

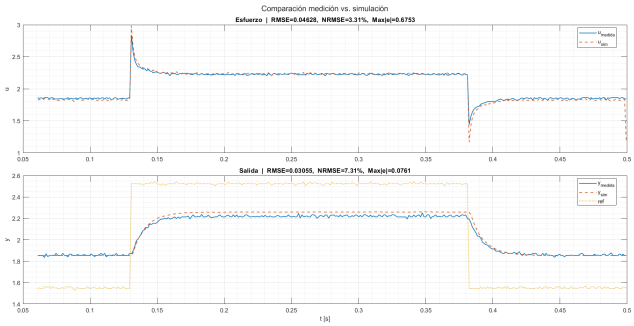

Figura 23: Compensador de adelanto: respuesta al escalón y esfuerzo (simulado vs. experimental, *sin* offset).

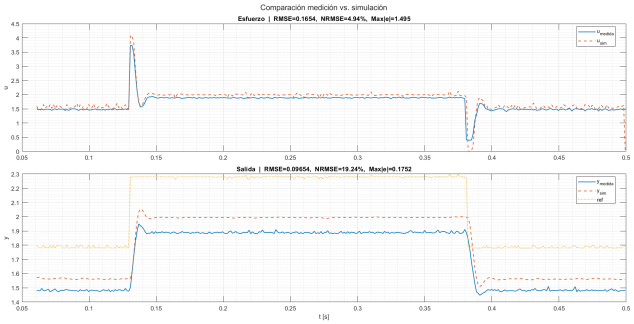

Figura 24: Compensador Proporcional: respuesta al escalón y esfuerzo (simulado vs. experimental, *con* offset).

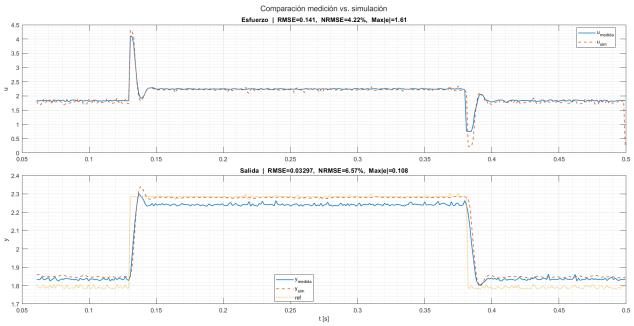

Figura 25: Compensador Proporcional: respuesta al escalón y esfuerzo (simulado vs. experimental, *sin* offset).

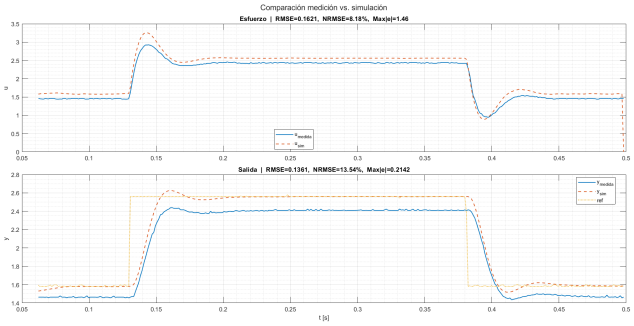

Figura 26: Compensador de adelanto + integrador: respuesta al escalón y esfuerzo (simulado vs. experimental, *con* offset).

Cuadro I: Comparativa temporal (simulado vs. experimental) y NRMSE (con/sin offset).

Sistema	$\mathbf{t_r}$ [ms] (sim/exp)	$\mathbf{M_p}$ [%] (sim/exp)	$\mathbf{e_{ss}}$ [%] (sim/exp)	**NRMSE** [%] (con/sin off)
Lazo abierto	40.0 / 34.2	0 / 0	0 / 1.8182	19.14 / 18.43
C1 (lead)	21.17 / 19.8	0 / 0	45.77 / 42–46.79	38.02 / 7.31
Proporcional	3.736 / 3.950	15.4 / 10–20	12.5 / $\approx$ 15.825	19.24 / 6.57
Lead + integrador	13.7 / 17.10	7.433 / $\approx$ 5	0 / 0	13.54 / 5.61

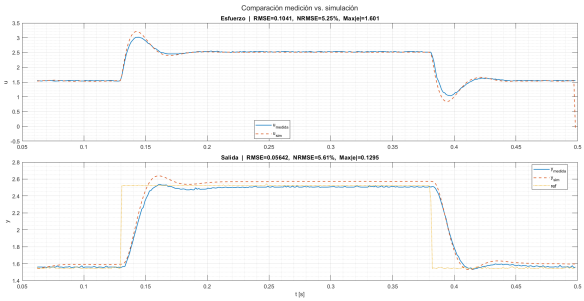

Figura 27: Compensador de adelanto + integrador: respuesta al escalón y esfuerzo (simulado vs. experimental, *sin* offset).

VI-B. Métricas de ajuste y desempeño temporal

Las métricas usadas en cada gráfica (y en la tabla resumen) son:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\big(y[k]-\hat{y}[k]\big)^2}, \tag{15}$$

$$\mathrm{NRMSE} = \frac{\mathrm{RMSE}}{y_{\max}-y_{\min}}, \tag{16}$$

$$e_{\max} = \max_k |y[k]-\hat{y}[k]|. \tag{17}$$

En la tabla I, los índices de *tiempo de subida* t_r y *sobreimpulso* M_p se obtienen de las curvas de esta sección. El error en estado estacionario e_{ss} se reporta cuando aplica (lazo cerrado), y el *error cuadratico medio normalizado* entre las mediciones y simulaciones se especifica tanto *con* como *sin* el offset de las señales.

VI-C. Error de velocidad (seguimiento de rampa)

Para el compensador C2 (integrador+lead) se ensayó seguimiento a rampa $r[k] = kT$ (pendiente 1 u/s). El error en régimen para sistemas tipo 1 en discreto es

$$e_{ss} = \frac{T}{K_{v,z}}, \qquad K_{v,z} = \lim_{z\to 1}(z-1)\,L(z). \tag{18}$$

Con el $L(z)$ obtenido, se midió/estimó $K_v \simeq 78.43 \Rightarrow e_{ss}^{(\mathrm{teo})} \approx 1/78.43 \approx 1.28\,\%$, en concordancia con el valor observado tras un transitorio breve.

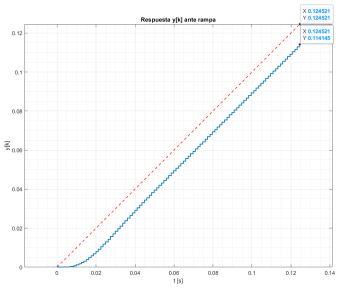

Figura 28: Seguimiento de rampa con Compensador integrador + lead.

VI-D. Discusión de discrepancias

Las diferencias entre curvas simuladas y experimentales se explican principalmente por: (i) **limitación/saturación** en la señal de esfuerzo debido a la carga del DAC (la excitación queda "achatada"), y (ii) **descalce de referencia** (offset). A ello se suman tolerancias de componentes pasivos, lo que desplaza levemente parámetros característicos de la planta. Aun así, la *dinámica global* buscada (forma de la respuesta y tiempos) se mantuvo acorde a la simulación.

VII. Conclusiones

El conjunto de ensayos y comparaciones *dominio del tiempo* demuestra que, a pesar de las limitaciones de implementación y de la dispersión de la planta real, el desempeño experimental se mantiene coherente con el diseño y dentro de las bandas esperadas. A continuación se sintetizan los hallazgos principales.

Ajuste simulación–experimento

Al simular con la **misma entrada** medida y remover el **offset estático** entre DAC y ADC, el ajuste mejora de forma notable (véase la tabla I). Lo que persiste se explica por *efectos dinámicos* no ideales: como saturaciones suaves del actuador y atenuaciones por carga.

Limitaciones de hardware observadas

Se verificó que la **corriente de salida del DAC** no es suficiente para excitar directamente la planta —no logramos conseguir los rangos exactos de corriente que puede suministrar en el datasheet—, lo que produce *achatamiento* y *recorte* del esfuerzo de control y, en consecuencia, discrepancias de forma en las respuestas. Además, la referencia VDDA/2 y las líneas de alimentación presentan *desacople insuficiente*, favoreciendo derivas de nivel que se manifiestan como offset y ligeras variaciones de ganancia. Estas condiciones explican parte del desajuste restante aun tras corregir el offset en posprocesado.

Planta real vs. modelo nominal

La planta implementada difiere levemente del modelo continuo asumido por *tolerancias* de pasivos, ordenamiento de etapas e impedancia de entrada vista por la fuente del esfuerzo. Aun así, las respuestas medidas se ubican mayormente **dentro de la banda de tolerancias** obtenida por análisis estadístico, lo que respalda la *validez del modelo* para propósitos de diseño y la *robustez* del procedimiento seguido.

Acciones recomendadas

- Añadir un **buffer operacional** rail–to–rail y de baja impedancia a la salida del DAC para eliminar el error de carga y evitar recortes del esfuerzo.
- **Biaspassear** la referencia VDDA/2 y las líneas de alimentación, mejorando estabilidad de nivel y rechazo de ruido.

Conclusión general

En conjunto, los resultados muestran que el **método de diseño es robusto**: aun bajo carga del DAC, offsets y dispersión de componentes, las respuestas experimentales se mantienen *razonablemente cercanas* a las simuladas, especialmente con el compensador con integrador + adelanto, que satisface los objetivos de seguimiento y estabilidad con un compromiso adecuado entre rapidez y exactitud.

Apéndice A
Códigos de Matlab

```
close all
clear all
close all
clear all

addpath('..\Lab1\')
addpath('..\Lab2\')

%% Definicion de parametros
R_1 = 15e3;
R_3 = 15e3;
C_2 = 100e-9;

R_2 = 82e3;
R_4 = 82e3;
C_1 = 0.22e-6;

%% Generar funcion de transferencia d
numStage = [-R_3/R_1 -R_4/R_2];
denStage = { [C_2*R_3 1], [C_1*R_4 1] };

% Usamos celdas para guardar los tf de cada stage
Gstage = cell(1,2);
G = 1;
for i = 1:2
Gstage{i} = tf(numStage(i), denStage{i});
G = G*Gstage{i};
end

%% Analizamos en el tiempo
[tr, ts, wn] = plot_step_info(G);
disp('Planta continua G(s):')
G
zpk(G)



%% Paso 1: Definir periodo de muestreo
% Se busca una mejora de 4 en el tiempo de rising , o sea tr = 10 ms
N = 4;
T = tr/(8*N);
%Gcl = feedback(G,1);
%figure;
%step(Gcl)
%T = stepinfo(Gcl).SettlingTime/8;

%% Paso 2: Digitalizar con ZOH
Gd = c2d(G, T, 'zoh');
disp('Planta digital G(z):')
Gd
zpk(Gd)

%% === K para PM deseada ===
Gw = d2c(Gd,'tustin');        % continuo equivalente por Tustin
figure;
margin(Gw);grid on;
%% === LOOP INTERACTIVO: ajustar K, agregar LEAD/LAG, o auto-ajustar specs ===
% Requiere tener Gw (continuo por Tustin) ya definido.
s = tf('s');
C_total = tf(1);
K = 1;

fprintf('\n=== Estado inicial (K=1, C=1) ===\n');
L_now = K*C_total*Gw;
```

```
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('GM = %.2f dB, PM = %.2f°, Wcg = %.3g rad/s, Wcp = %.3g rad/s\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('L(s) = K*C*G_w  (inicio)');

seguir = true;
while seguir
fprintf('\nOpciones:\n');
fprintf('  [1] Ajustar K a un GM deseado (dB)\n');
fprintf('  [2] Agregar etapa LEAD/LAG (normalizada en Wcp)\n');
fprintf('  [3] Auto-ajustar a (GM_dB, PM)  - o usar K fijo y PM\n');
fprintf('  [4] Fijar K explícito (sin cálculos)\n');
fprintf('  [0] Terminar\n');
op = str2double(input('Elegí opción: ','s'));

switch op
case 1
GM_des_dB = str2double(input(' GM deseado (dB): ','s'));
L_now = K*C_total*Gw;
[GM_now, ~, ~, ~] = margin(L_now);
GM_now_dB = 20*log10(GM_now);
if ~isfinite(GM_now_dB)
error('GM no definido; agregá una etapa o cambiá el rango antes de ajustar K.');
end
% K_new saca GM a GM_des_dB (aprox directo en dB)
K = K * 10^((GM_now_dB - GM_des_dB)/20);
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> K = %.6g; GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', ...
K, 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras ajustar K');

case 2
tipo = lower(strtrim(input(' Tipo ("lead" o "lag"): ','s')));
phi  = str2double(input(' Fase a compensar (grados, positivo): ','s'));
[C_total, rep] = add_stage_at_Wcp(Gw, C_total, K, tipo, phi, s);
fprintf('  -> Etapa %s %+g° aplicada en wc=%.3g rad/s (normalizada)\n', rep.type, rep.phi_deg,
    rep.wc_used);
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras agregar etapa');

case 3
usa_k_fijo = lower(strtrim(input(' ¿Usar K fijo? [y/n]: ','s')));
if strcmpi(usa_k_fijo,'y')
K_fixed = str2double(input(' Valor de K fijo: ','s'));
PM_des  = str2double(input(' PM deseada (grados): ','s'));
[K, C_total, rep] = auto_hit_specs_fixedK(Gw, C_total, K_fixed, PM_des, s);
fprintf('  -> AUTO (K fijo=%.6g) listo: PM=%.2f°, GM=%.2f dB, Wcg=%.3g, Wcp=%.3g\n', ...
K, rep.PM, rep.GMdB, rep.Wcg, rep.Wcp);
else
GM_des_dB = str2double(input(' GM deseado (dB): ','s'));
PM_des    = str2double(input(' PM deseada (grados): ','s'));
[K, C_total, rep] = auto_hit_specs(Gw, C_total, K, GM_des_dB, PM_des, s);
fprintf('  -> AUTO listo (iters=%d): GM=%.2f dB, PM=%.2f°, Wcg=%.3g, Wcp=%.3g\n', ...
rep.iters, rep.GMdB, rep.PM, rep.Wcg, rep.Wcp);
end
L_now = K*C_total*Gw;
figure; margin(L_now); grid on; title('Tras auto-ajuste');

case 4
K = str2double(input(' K = ','s'));
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> K fijado. GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras fijar K');

```

```
case 0
seguir = false;
continue;

otherwise
fprintf('Opción inválida.\n');
end
end


%% Resultado final
K = 7;
C_total = K;
% C_total = 11.7057*d2c(C1,'tustin',T);
L_final = C_total*Gw;
[GMf, PMf, Wcgf, Wcpf] = margin(L_final);
fprintf('Márgenes: GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GMf), PMf, Wcgf, Wcpf);
% % figure; bode(C_total); grid on;
% figure; margin(L_final); grid on; title('L_{final}(s) = K · C_{total}(s) · G_w(s)');
% % C1 = c2d(C_total,T,'tustin');
%  figure;step(feedback(C1*Gd,1));grid on;title('Respuesta al escalon');
% figure;step(feedback(C1,Gd));grid on;title('Esfuerzo al escalon');



umin = 0;
umax =  4.08;
refmin = 1.75;
refmax = 2.25;
n_per_seg = 500;

Gc1f = feedback(Gd*C1,1);
%figure;
%step(Gc1f)
info = stepinfo(Gc1f);


info_ext = plot_step_annot(Gc1f, 'la planta compensada con un lag-filter de 1º orden, anulando el polo más
    rápido de la planta.');
compensador = zpk(C1);

% [td, refd, yd, ud, ed, coefs] = sim_compensador_first_order( ...
%     Gd, compensador.Z{1}, compensador.P{1}, compensador.K, T, umin, umax, refmin, refmax, n_per_seg);

[td, refd, yd, ud, ed, coefs] = sim_compensador_first_order( ...
Gd, 0, 0, K, T, umin, umax, refmin, refmax, n_per_seg);

% [td, refd, yd, ud, ed, coefs] = sim_compensador_second_order( ...
%     Gd, compensador.Z{1}(1), ...
%     compensador.Z{1}(2), ...
%     compensador.P{1}(1), ...
%     compensador.P{1}(2), compensador.K, ...
%     T, umin, umax, refmin, refmax, n_per_seg);

Nini = 100;              % descartar al inicio
Nfin = 100;              % descartar al final
idx0 = Nini + 1;         % índice inicial válido
idx1 = length(td) - Nfin; % índice final válido

td   = td(idx0:idx1)-td(idx0);
refd = refd(idx0:idx1);
yd   = yd(idx0:idx1);
ud   = ud(idx0:idx1);
ed   = ed(idx0:idx1);

% Si querés que el tiempo arranque en 0

```

```
197
198
199
200
201 figure;
202
203 % ---------- Subplot 1 ----------
204 axAbs = subplot(2,1,1);  % eje izquierdo (absoluto)
205 hold(axAbs,'on'); grid(axAbs,'on');
206
207 % y[k] en rojo (abs)
208 hY = stairs(axAbs, td, yd, 'r-', 'LineWidth', 1.4);
209
210 % Armamos eje derecho transparente
211 axPct = axes('Position', get(axAbs,'Position'), ...
212 'Color','none', 'YAxisLocation','right', ...
213 'XLim', get(axAbs,'XLim'), 'XTick',[], 'Box','off');
214 hold(axPct,'on');
215
216 % r[k] en negro, graficado en %
217 ref_pct = 100*(refd - refmin)/(refmax - refmin);
218 hR = stairs(axPct, td, refd, 'k--', 'LineWidth', 1.2);
219
220 % ======= Cálculo de límites con margen =======
221 yd_pct  = 100*(yd - refmin)/(refmax - refmin);
222 pctAll  = [yd_pct; ref_pct];
223
224 % Valores extremos en % con margen del 5 %
225 rawMin = min(pctAll);
226 rawMax = max(pctAll);
227 span   = rawMax - rawMin;
228 pctMin = rawMin - 0.05*span;
229 pctMax = rawMax + 0.05*span;
230
231 % Redondeamos a múltiplos de 20 para ticks
232 stepPct = 20;
233 tickMin = stepPct*floor(pctMin/stepPct);
234 tickMax = stepPct*ceil (pctMax/stepPct);
235 pctTicks = tickMin:stepPct:tickMax;
236
237 % Convertimos a absolutos
238 valTicks = refmin + (pctTicks/100)*(refmax - refmin);
239
240 % Aplicamos a ambos ejes
241 set(axAbs,'YLim',[valTicks(1) valTicks(end)],'YTick',valTicks);
242 set(axPct,'YLim',[valTicks(1) valTicks(end)],'YTick',valTicks,...
243 'YTickLabel',compose('%.0f %%',pctTicks));
244
245 % Etiquetas
246 xlabel(axAbs,'Tiempo [s]');
247 ylabel(axAbs,'Respuesta [valor absoluto]');
248 ylabel(axPct,'Escala relativa a ref [%]');
249 title(axAbs,'Respuesta discreta con compensador 1º orden, anulando el polo más rápido de la planta.');
250 legend(axAbs, [hY, hR], {'y[k]', 'r[k]'}, 'Location', 'best');
251
252 % ---------- Subplot 2: esfuerzo de control ----------
253 axU = subplot(2,1,2);
254 stairs(axU, td, ud, 'LineWidth',1.2); grid(axU,'on');
255 yline(axU, umax,'r:'); yline(axU, umin,'r:');
256 xlabel(axU,'Tiempo [s]'); ylabel(axU,'u[k]');
257 title(axU,'Esfuerzo de control con saturación');
258
259 % --- Margen de 5% en eje Y ---
260 uMin = min(ud);
261 uMax = max(ud);
262 span = uMax - uMin;
263 ylim(axU, [uMin - 0.05*span, uMax + 0.05*span]);
```

```
264
265
266 grid(axAbs,'on');     % grilla principal
267 grid(axAbs,'minor'); % grilla secundaria
268
269 grid(axPct,'on');
270 grid(axPct,'minor');
271
272 grid(axU,'on');
273 grid(axU,'minor');
274 %% ================== SETUP INICIAL ==================
275 % Asumimos que ya construiste G (continuo), calculaste T, y obtuviste:
276 %   Gd = c2d(G, T, 'zoh');
277 %   Gw = d2c(Gd, 'tustin');    % "plano w" para diseñar
278 %
279 % Si no, descomenta y ajustá a tu caso:
280 % R_1 = 15e3; R_3 = 15e3; C_2 = 100e-9; R_2 = 82e3; R_4 = 82e3; C_1 = 0.22e-6;
281 % numStage = [-R_3/R_1 -R_4/R_2];
282 % denStage = { [C_2*R_3 1], [C_1*R_4 1] };
283 % Gstage = cell(1,2); G = 1;
284 % for i = 1:2, Gstage{i} = tf(numStage(i), denStage{i}); G = G*Gstage{i}; end
285 % [tr, ts, wn] = plot_step_info(G); N = 4; T = tr/(8*N);
286 % Gd = c2d(G, T, 'zoh'); Gw = d2c(Gd, 'tustin');
287
288 assert(exist('Gw','var')==1 && exist('Gd','var')==1 && exist('T','var')==1, ...
289 'Definí primero Gw, Gd y T antes de correr este script.');
290
291 s = tf('s');
292
293 %% ================== COMPENSADOR INICIAL ==================
294 % Tu elección: integrador por defecto (podés cambiar a tf(1) si no querés I)
295 C_total = 1/s;      % <-- SI NO QUERÉS integrador, poné: C_total = tf(1);
296 K = 1;
297
298 %% ================== ESTADO INICIAL (CONTINUO, PLANO w) ==================
299 fprintf('\n=== Estado inicial (K=1, C inicial) ===\n');
300 L_now = K*C_total*Gw;
301 [GM, PM, Wcg, Wcp] = margin(L_now);
302 fprintf('GM = %.2f dB, PM = %.2f°, Wcg = %.3g rad/s, Wcp = %.3g rad/s\n', 20*log10(GM), PM, Wcg, Wcp);
303 figure; margin(L_now); grid on; title('L(s) = K*C(s)*G_w(s)  (inicio)');
304
305 %% ================== LOOP INTERACTIVO ==================
306 seguir = true;
307 while seguir
308 fprintf('\nOpciones:\n');
309 fprintf('  [1] Ajustar K a un GM deseado (dB)\n');
310 fprintf('  [2] Agregar etapa LEAD/LAG (normalizada en Wcp)\n');
311 fprintf('  [3] Auto-ajustar a (GM_dB, PM)  - o usar K fijo y PM\n');
312 fprintf('  [4] Fijar K explícito (sin cálculos)\n');
313 fprintf('  [5] Ir a DISCRETO (c2d con prewarp) y simular\n');
314 fprintf('  [0] Terminar\n');
315 op = str2double(input('Elegí opción: ','s'));
316
317 switch op
318 case 1   % Ajustar K a GM deseado
319 GM_des_dB = str2double(input(' GM deseado (dB): ','s'));
320 L_now = K*C_total*Gw;
321 [GM_now, ~, ~, ~] = margin(L_now);
322 GM_now_dB = 20*log10(GM_now);
323 if ~isfinite(GM_now_dB)
324 error('GM no definido; agregá una etapa o cambiá el rango antes de ajustar K.');
325 end
326 K = K * 10^((GM_now_dB - GM_des_dB)/20);
327 L_now = K*C_total*Gw;
328 [GM, PM, Wcg, Wcp] = margin(L_now);
329 fprintf('  -> K = %.6g; GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', ...
```

```
K, 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras ajustar K');

case 2  % Agregar etapa lead/lag normalizada en Wcp
tipo = lower(strtrim(input(' Tipo ("lead" o "lag"): ','s')));
phi  = str2double(input(' Fase a compensar (grados, positivo): ','s'));
[C_total, rep] = add_stage_at_Wcp(Gw, C_total, K, tipo, phi, s);
fprintf('  -> Etapa %s %+g° aplicada en wc=%.3g rad/s (normalizada)\n', rep.type, rep.phi_deg,
    rep.wc_used);
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras agregar etapa');

case 3  % Auto-ajuste
usa_k_fijo = lower(strtrim(input(' ¿Usar K fijo? [y/n]: ','s')));
if strcmpi(usa_k_fijo,'y')
K_fixed = str2double(input(' Valor de K fijo: ','s'));
PM_des  = str2double(input(' PM deseada (grados): ','s'));
[K, C_total, rep] = auto_hit_specs_fixedK(Gw, C_total, K_fixed, PM_des, s);
fprintf('  -> AUTO (K fijo=%.6g) listo: PM=%.2f°, GM=%.2f dB, Wcg=%.3g, Wcp=%.3g\n', ...
K, rep.PM, rep.GMdB, rep.Wcg, rep.Wcp);
else
GM_des_dB = str2double(input(' GM deseado (dB): ','s'));
PM_des    = str2double(input(' PM deseada (grados): ','s'));
[K, C_total, rep] = auto_hit_specs(Gw, C_total, K, GM_des_dB, PM_des, s);
fprintf('  -> AUTO listo (iters=%d): GM=%.2f dB, PM=%.2f°, Wcg=%.3g, Wcp=%.3g\n', ...
rep.iters, rep.GMdB, rep.PM, rep.Wcg, rep.Wcp);
end
L_now = K*C_total*Gw;
figure; margin(L_now); grid on; title('Tras auto-ajuste');

case 4  % Fijar K
K = str2double(input(' K = ','s'));
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> K fijado. GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras fijar K');

case 5  % Discretizar C(s) -> C(z) con prewarp y simular
Lc = K*C_total*Gw;
[~, ~, ~, Wcp_c] = margin(Lc);
if ~(isfinite(Wcp_c) && Wcp_c>0)
warning('No hay Wcp continuo definido; uso prewarp=0 (Tustin simple).');
opt = c2dOptions('tustin');
else
opt = c2dOptions('tustin','PrewarpFrequency', Wcp_c);
end
C2 = c2d(C_total, T, opt);

% Renormalizar en la frecuencia digital equivalente
if isfinite(Wcp_c) && Wcp_c>0
Omega_c = 2*atan(Wcp_c*T/2);                    % Ωc
Cz_ejO  = evalfr(C2, exp(1j*Omega_c));
gfix    = 1/abs(Cz_ejO);
C2      = gfix * C2;
end

% Lazo discreto y márgenes
Lz = K * C2 * Gd;
figure; margin(Lz); grid on; title('Discreto: Lz = K*C(z)*Gd(z)');
[GMz, PMz, Wcgz, Wcpz] = margin(Lz);
fprintf('DISCRETO: GM=%.2f dB, PM=%.2f°, Wcg=%.3g rad/s, Wcp=%.3g rad/s\n', 20*log10(GMz), PMz, Wcgz,
    Wcpz);

% Respuesta al escalón (y[k])
Ts = Gd.Ts; if Ts<=0, Ts = T; end
```

```
N  = 2000; tvec = (0:N-1)'*Ts;
CLz_y = feedback(Lz, 1);
[y, tstep] = step(CLz_y, tvec);
figure; stairs(tstep, y); grid on; title('Discreto: y[k] ante escalón'); xlabel('t [s]');

% Esfuerzo u[k] = C(z)/(1+C(z)Gd(z)) * r[k]  (para r=step)
T_u_r = feedback(C2, Gd*C2);   % C2 / (1 + C2*Gd)
[u, tstep2] = step(T_u_r, tvec);
figure; stairs(tstep2, u); grid on; title('Discreto: esfuerzo u[k] ante escalón'); xlabel('t [s]');

case 0
seguir = false;
continue;

otherwise
fprintf('Opción inválida.\n');
end
end

%% ================== RESUMEN FINAL (CONTINUO Y DISCRETO, TUSTIN SIMPLE) ==================
% --- Continuo (diseño sobre Gw) ---
L_final = K*C_total*Gw;
[GMf, PMf, Wcgf, Wcpf] = margin(L_final);

fprintf('\n=== RESULTADO FINAL (CONTINUO, DISEÑO) ===\n');
fprintf('K = %.6g\n', K);
disp('C_total(s) ='); disp(zpk(C_total));
fprintf('Márgenes: GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GMf), PMf, Wcgf, Wcpf);

figure; bode(C_total); grid on; title('Compensador total C_{total}(s)');
figure; margin(L_final); grid on; title('L_{final}(s) = K · C_{total}(s) · G_w(s)');

% --- Discretizar el compensador (Tustin ""normal, sin prewarp) ---
C1  = K*c2d(C_total, T, 'tustin');

% --- Lazo discreto y márgenes ---
Lz = C1*Gd;
figure; margin(Lz); grid on; title('Final discreto: Lz = K·C(z)·G_d(z)');
[GMz, PMz, Wcgz, Wcpz] = margin(Lz);
fprintf('=== DISCRETO FINAL (Tustin simple) ===\nGM=%.2f dB, PM=%.2f°, Wcg=%.3g, Wcp=%.3g\n', ...
20*log10(GMz), PMz, Wcgz, Wcpz);


% Salida y[k] ante escalón (CL: (K*C2*Gd)/(1+K*C2*Gd))
CLz_y = feedback(Lz, 1);
[y, tstep] = step(CLz_y);
figure; stairs(tstep, y); grid on; xlabel('t [s]'); ylabel('y[k]');
title(' y[k] ante escalón');



% Esfuerzo u[k] ante escalón: T_{u<-r}(z) = (K*C2)/(1+K*C2*Gd)
T_u_r = feedback(C1, Gd);   % equivalente a (K*C2) / (1 + K*C2*Gd)
[u, tstep2] = step(T_u_r);
figure; stairs(tstep2, u); grid on; xlabel('t [s]'); ylabel('u[k]');
title('Final discreto: esfuerzo u[k] ante escalón');


%% ================== COMPENSADOR INICIAL ==================
% Tu elección: integrador por defecto (podés cambiar a tf(1) si no querés I)
C_total = 1/s;      % <-- SI NO QUERÉS integrador, poné: C_total = tf(1);
K = 1;

fprintf('\n=== Estado inicial (K=1, C inicial) ===\n');
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
```

```
fprintf('GM = %.2f dB, PM = %.2f°, Wcg = %.3g rad/s, Wcp = %.3g rad/s\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('L(s) = K*C(s)*G_w(s)  (inicio)');

seguir = true;
while seguir
fprintf('\nOpciones:\n');
fprintf('  [1] Ajustar K a un GM deseado (dB)\n');
fprintf('  [2] Agregar etapa LEAD/LAG (normalizada en Wcp)\n');
fprintf('  [3] Auto-ajustar a (GM_dB, PM)  - o usar K fijo y PM\n');
fprintf('  [4] Fijar K explícito (sin cálculos)\n');
fprintf('  [5] Ir a DISCRETO (c2d con prewarp) y simular\n');
fprintf('  [0] Terminar\n');
op = str2double(input('Elegí opción: ','s'));

switch op
case 1  % Ajustar K a GM deseado
GM_des_dB = str2double(input(' GM deseado (dB): ','s'));
L_now = K*C_total*Gw;
[GM_now, ~, ~, ~] = margin(L_now);
GM_now_dB = 20*log10(GM_now);
if ~isfinite(GM_now_dB)
error('GM no definido; agregá una etapa o cambiá el rango antes de ajustar K.');
end
K = K * 10^((GM_now_dB - GM_des_dB)/20);
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> K = %.6g; GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', ...
K, 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras ajustar K');

case 2  % Agregar etapa lead/lag normalizada en Wcp
tipo = lower(strtrim(input(' Tipo ("lead" o "lag"): ','s')));
phi  = str2double(input(' Fase a compensar (grados, positivo): ','s'));
[C_total, rep] = add_stage_at_Wcp(Gw, C_total, K, tipo, phi, s);
fprintf('  -> Etapa %s %+g° aplicada en wc=%.3g rad/s (normalizada)\n', rep.type, rep.phi_deg,
    rep.wc_used);
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras agregar etapa');

case 3  % Auto-ajuste
usa_k_fijo = lower(strtrim(input(' ¿Usar K fijo? [y/n]: ','s')));
if strcmpi(usa_k_fijo,'y')
K_fixed = str2double(input(' Valor de K fijo: ','s'));
PM_des  = str2double(input(' PM deseada (grados): ','s'));
[K, C_total, rep] = auto_hit_specs_fixedK(Gw, C_total, K_fixed, PM_des, s);
fprintf('  -> AUTO (K fijo=%.6g) listo: PM=%.2f°, GM=%.2f dB, Wcg=%.3g, Wcp=%.3g\n', ...
K, rep.PM, rep.GMdB, rep.Wcg, rep.Wcp);
else
GM_des_dB = str2double(input(' GM deseado (dB): ','s'));
PM_des    = str2double(input(' PM deseada (grados): ','s'));
[K, C_total, rep] = auto_hit_specs(Gw, C_total, K, GM_des_dB, PM_des, s);
fprintf('  -> AUTO listo (iters=%d): GM=%.2f dB, PM=%.2f°, Wcg=%.3g, Wcp=%.3g\n', ...
rep.iters, rep.GMdB, rep.PM, rep.Wcg, rep.Wcp);
end
L_now = K*C_total*Gw;
figure; margin(L_now); grid on; title('Tras auto-ajuste');

case 4  % Fijar K
K = str2double(input(' K = ','s'));
L_now = K*C_total*Gw;
[GM, PM, Wcg, Wcp] = margin(L_now);
fprintf('  -> K fijado. GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GM), PM, Wcg, Wcp);
figure; margin(L_now); grid on; title('Tras fijar K');

case 5  % Discretizar C(s) -> C(z) con prewarp y simular
```

```
Lc = K*C_total*Gw;
[~, ~, ~, Wcp_c] = margin(Lc);
if ~(isfinite(Wcp_c) && Wcp_c>0)
warning('No hay Wcp continuo definido; uso prewarp=0 (Tustin simple).');
opt = c2dOptions('tustin');
else
opt = c2dOptions('tustin','PrewarpFrequency', Wcp_c);
end
C2 = c2d(C_total, T, opt);

% Renormalizar en la frecuencia digital equivalente
if isfinite(Wcp_c) && Wcp_c>0
Omega_c = 2*atan(Wcp_c*T/2);                  % Ωc
Cz_ej0  = evalfr(C2, exp(1j*Omega_c));
gfix    = 1/abs(Cz_ej0);
C2      = gfix * C2;
end

% Lazo discreto y márgenes
Lz = K * C2 * Gd;
figure; margin(Lz); grid on; title('Discreto: Lz = K*C(z)*Gd(z)');
[GMz, PMz, Wcgz, Wcpz] = margin(Lz);
fprintf('DISCRETO: GM=%.2f dB, PM=%.2f°, Wcg=%.3g rad/s, Wcp=%.3g rad/s\n', 20*log10(GMz), PMz, Wcgz,
    Wcpz);

% Respuesta al escalón (y[k])
Ts = Gd.Ts; if Ts<=0, Ts = T; end
N  = 2000; tvec = (0:N-1)'*Ts;
CLz_y = feedback(Lz, 1);
[y, tstep] = step(CLz_y, tvec);
figure; stairs(tstep, y); grid on; title('Discreto: y[k] ante escalón'); xlabel('t [s]');

% Esfuerzo u[k] = C(z)/(1+C(z)Gd(z)) * r[k]  (para r=step)
T_u_r = feedback(C2, Gd*C2);    % C2 / (1 + C2*Gd)
[u, tstep2] = step(T_u_r, tvec);
figure; stairs(tstep2, u); grid on; title('Discreto: esfuerzo u[k] ante escalón'); xlabel('t [s]');

case 0
seguir = false;
continue;

otherwise
fprintf('Opción inválida.\n');
end
end


%% Resultado final
C_total = K*C_total;
L_final = C_total*Gw;
[GMf, PMf, Wcgf, Wcpf] = margin(L_final);
fprintf('Márgenes: GM = %.2f dB, PM = %.2f°, Wcg = %.3g, Wcp = %.3g\n', 20*log10(GMf), PMf, Wcgf, Wcpf);
figure; bode(C_total); grid on;
figure; margin(L_final); grid on; title('L_{final}(s) = K · C_{total}(s) · G_w(s)');
C2 = c2d(C_total,T,'tustin');
figure;step(feedback(C2*Gd,1));grid on;title('Respuesta al escalon');
figure;step(feedback(C2,Gd));grid on;title('Esfuerzo al escalon');


Lz = Gd*C2;
% Salida y[k] ante la rampa (CL: (K*C2*Gd)/(1+K*C2*Gd))
CLz_y = feedback(Lz, 1);
N = 100*T;              % duración total
t = 0:T:N;              % vector de tiempo
rampa = t;              % rampa pendiente 1

[y, tstep] = lsim(CLz_y, rampa, t);
```

```
593
594 figure;
595 stairs(tstep, y, 'LineWidth',1.5); hold on;
596 plot(tstep, rampa, 'r--','LineWidth',1.2);
597
598 xlabel('t [s]');
599 ylabel('y[k]');
600 title('Respuesta y[k] ante rampa');
601 grid on; grid minor;
602
603 % Forzar relación 1:1 en ejes
604 axis equal
605
606
607
608
609 umin = 0;
610 umax =  4.08;
611 refmin = 1.75;
612 refmax = 2.25;
613 n_per_seg = 500;
614
615 Gc1f = feedback(Gd*C2,1);
616 %figure;
617 %step(Gc1f)
618 info = stepinfo(Gc1f);
619
620
621 info_ext = plot_step_annot(Gc1f, 'la planta compensada con un lag-filter de 1º orden, anulando el polo más
        rápido de la planta.');
622 compensador = zpk(C2);
623
624
625 [td, refd, yd, ud, ed, coefs] = sim_compensador_second_order( ...
626 Gd, compensador.Z{1}(1), ...
627 compensador.Z{1}(2), ...
628 compensador.P{1}(1), ...
629 compensador.P{1}(2), compensador.K, ...
630 T, umin, umax, refmin, refmax, n_per_seg);
631
632 figure;
633 [y,td]=lsim(Gc1f,td);
634 stairs(td,y);
635 hold on;
636 grid on;
637 plot(td,td);
638
639
640 Nini = 100;              % descartar al inicio
641 Nfin = 100;              % descartar al final
642 idx0 = Nini + 1;         % índice inicial válido
643 idx1 = length(td) - Nfin; % índice final válido
644
645 td   = td(idx0:idx1)-td(idx0);
646 refd = refd(idx0:idx1);
647 yd   = yd(idx0:idx1);
648 ud   = ud(idx0:idx1);
649 ed   = ed(idx0:idx1);
650
651 % Si querés que el tiempo arranque en 0
652
653
654
655
656
657 figure;
658
```

```
% ---------- Subplot 1 ----------
axAbs = subplot(2,1,1);  % eje izquierdo (absoluto)
hold(axAbs,'on'); grid(axAbs,'on');

% y[k] en rojo (abs)
hY = stairs(axAbs, td, yd, 'r-', 'LineWidth', 1.4);

% Armamos eje derecho transparente
axPct = axes('Position', get(axAbs,'Position'), ...
'Color','none', 'YAxisLocation','right', ...
'XLim', get(axAbs,'XLim'), 'XTick',[], 'Box','off');
hold(axPct,'on');

% r[k] en negro, graficado en %
ref_pct = 100*(refd - refmin)/(refmax - refmin);
hR = stairs(axPct, td, refd, 'k--', 'LineWidth', 1.2);

% ======= Cálculo de límites con margen =======
yd_pct  = 100*(yd - refmin)/(refmax - refmin);
pctAll  = [yd_pct; ref_pct];

% Valores extremos en % con margen del 5 %
rawMin = min(pctAll);
rawMax = max(pctAll);
span   = rawMax - rawMin;
pctMin = rawMin - 0.05*span;
pctMax = rawMax + 0.05*span;

% Redondeamos a múltiplos de 20 para ticks
stepPct = 20;
tickMin = stepPct*floor(pctMin/stepPct);
tickMax = stepPct*ceil (pctMax/stepPct);
pctTicks = tickMin:stepPct:tickMax;

% Convertimos a absolutos
valTicks = refmin + (pctTicks/100)*(refmax - refmin);

% Aplicamos a ambos ejes
set(axAbs,'YLim',[valTicks(1) valTicks(end)],'YTick',valTicks);
set(axPct,'YLim',[valTicks(1) valTicks(end)],'YTick',valTicks,...
'YTickLabel',compose('%.0f %%',pctTicks));

% Etiquetas
xlabel(axAbs,'Tiempo [s]');
ylabel(axAbs,'Respuesta [valor absoluto]');
ylabel(axPct,'Escala relativa a ref [%]');
title(axAbs,'Respuesta discreta con compensador 1º orden, anulando el polo más rápido de la planta.');
legend(axAbs, [hY, hR], {'y[k]', 'r[k]'}, 'Location', 'best');

% ---------- Subplot 2: esfuerzo de control ----------
axU = subplot(2,1,2);
stairs(axU, td, ud, 'LineWidth',1.2); grid(axU,'on');
yline(axU, umax,'r:'); yline(axU, umin,'r:');
xlabel(axU,'Tiempo [s]'); ylabel(axU,'u[k]');
title(axU,'Esfuerzo de control con saturación');

% --- Margen de 5% en eje Y ---
uMin = min(ud);
uMax = max(ud);
span = uMax - uMin;
ylim(axU, [uMin - 0.05*span, uMax + 0.05*span]);


grid(axAbs,'on');     % grilla principal
grid(axAbs,'minor'); % grilla secundaria

grid(axPct,'on');
```

```
grid(axPct,'minor');

grid(axU,'on');
grid(axU,'minor');
```

Listing 1: Hoja cálculos utilizada para el diseño de los compensadores.

```
function C = lag_norm_at_wc(phi_deg, wc, s)
phi_deg = max(1, min(60, phi_deg));
sphi = sind(phi_deg);
beta = (1 + sphi) / (1 - sphi);        % >1
T = 1 / (wc * sqrt(beta));
C0 = (1 + T*s) / (1 + (T/beta)*s);
g = 1 / abs(freqresp(C0, 1j*wc));      % |C(jwc)|=1
C = g * C0;
end


function C = lag_gain_at_wc_db(drop_dB, wc, s, kfactor)
% Lag que REDUCE el módulo ~drop_dB en torno a wc sin tocar K.
% NO normalizado (justamente para cambiar GM). Ubica esquinas a wc/kfactor.
% Aproxima |C( j)| ~  1/ para  >> 1/T (efecto lag clásico).
if nargin<4 || isempty(kfactor), kfactor = 10; end
beta = max(1.05, 10^(drop_dB/20));      % objetivo de caída de |C |1/beta (dB ~ -drop_dB)
% Colocar esquinas bien por debajo:  z=1/T,  p =1/(T); con  >1 => polo más bajo.
% Elegimos  z = wc/kfactor => T = kfactor/wc
T = kfactor / wc;
C = (1 + T*s) / (1 + beta*T*s);         % lag correcto (módulo  ~1/ a altas frecuencias)
end
function [K_out, C_out, report] = auto_hit_specs_fixedK(Gw, C_in, K_fixed, PM_des, s)
% No toca K. Ajusta PM usando:
%  (1) Lead normalizado en el Wcp actual para sumar fase hasta PM_des
%  (2) Si por el lead se movió GM y querés recuperar un poquito de fase,
%      mete un lead suavito extra (<=20°). (Lag normalizado opcional si querés bajar fase)
tolPM = 1.0;     % deg
maxit = 4;

K = K_fixed;
C = C_in;

for it = 1:maxit
L = K*C*Gw;
[~, PM, ~, Wcp] = margin(L);
if ~(isfinite(Wcp) && Wcp > 0)
error('No hay Wcp con K fijo; probá agregar una etapa o cambiar K.');
end

phi_need = PM_des - PM;
if phi_need > 0.5
phi_apply = min(max(phi_need, 1), 60);
C = lead_norm_at_wc(phi_apply, Wcp, s) * C;
elseif phi_need < -0.5
% Si te pasaste de PM, podés usar lag NORMALIZADO suave para restar fase
phi_lag = min(max(abs(phi_need), 1), 30);
C = lag_norm_at_wc(phi_lag, Wcp, s) * C;
else
break;  % dentro de tolerancia
end

% ¿ya estamos dentro de tolerancia?
L2 = K*C*Gw;
[~, PM2] = margin(L2);
if isfinite(PM2) && abs(PM2 - PM_des) <= tolPM
break;
end
end

```

```
K_out = K;
C_out = C;

Lf = K_out*C_out*Gw;
[GMf, PMf, Wcgf, Wcpf] = margin(Lf);
report = struct('GMdB',20*log10(GMf), 'PM',PMf, 'Wcg',Wcgf, 'Wcp',Wcpf, 'iters',it);
end

function [K_out, C_out, report] = auto_hit_specs(Gw, C_in, K_in, GM_des_dB, PM_des, s)
% AUTO que prioriza: (1) ajustar K ~ GM, (2) sumar LEAD (normalizado) para PM,
% (3) si GM < objetivo, sumar LAG "de ganancia" (NO normalizado) para subir GM,
% (4) si el LAG te bajó PM, agrega un LEAD chiquito para recuperar.
%
% Devuelve K_out, C_out y reporte (GM, PM, Wcg, Wcp, iters).

tolGM = 0.5;     % dB
tolPM = 1.0;     % deg
maxit = 4;

K = K_in;
C = C_in;

for it = 1:maxit
% --- (A) Ajustar K para GM deseado (aprox directo en dB)
L = K*C*Gw;
[GM, PM, ~, ~] = margin(L); GMdB = 20*log10(GM);
if ~isfinite(GMdB), error('GM no definido; añadí una etapa antes.'); end
K = K * 10^((GMdB - GM_des_dB)/20);

% --- (B) LEAD normalizado para alcanzar PM deseada en Wcp actual
L = K*C*Gw;
[~, PM, ~, Wcp] = margin(L);
if ~(isfinite(Wcp) && Wcp>0), error('No hay Wcp tras ajustar K.'); end

phi_need = PM_des - PM;             % fase a sumar
if phi_need > 0.5
phi_apply = min(max(phi_need, 1), 60);    % limitar 1..60°
C = lead_norm_at_wc(phi_apply, Wcp, s) * C;
end

% --- Recalculamos márgenes
L = K*C*Gw;
[GM, PM, ~, Wcp] = margin(L); GMdB = 20*log10(GM);

% --- (C) Si GM quedó por debajo, metemos un LAG de ganancia (NO normalizado)
if isfinite(GMdB) && (GMdB < GM_des_dB - tolGM)
dB_needed = GM_des_dB - GMdB;                  % cuánto GM falta
C = lag_gain_at_wc_db(dB_needed, Wcp, s, 10) * C;   % kfactor=10 => esquinas bien por debajo

% recuperar PM si el lag bajó demasiado
L2 = K*C*Gw;
[~, PM2, ~, Wcp2] = margin(L2);
if isfinite(PM2) && PM2 < PM_des - tolPM
phi_fix = min(max(PM_des - PM2, 1), 30);          % LEAD suave de corrección
C = lead_norm_at_wc(phi_fix, Wcp2, s) * C;
end
end

% --- (D) Chequeo de convergencia
Lf = K*C*Gw;
[GMf, PMf, ~, ~] = margin(Lf); GMf_dB = 20*log10(GMf);
doneGM = isfinite(GMf_dB) && abs(GMf_dB - GM_des_dB) <= tolGM;
donePM = isfinite(PMf)    && abs(PMf    - PM_des)    <= tolPM;
if doneGM && donePM, break; end
end

K_out = K; C_out = C;
```

```
Lf = K_out*C_out*Gw; [GMf, PMf, Wcgf, Wcpf] = margin(Lf);
report = struct('GMdB',20*log10(GMf), 'PM',PMf, 'Wcg',Wcgf, 'Wcp',Wcpf, 'iters',it);
end

function [C_out, report] = add_stage_at_Wcp(Gw, C_in, K, comp_type, phi_deg, s)
% Agrega UNA etapa (lead o lag) normalizada en el Wcp del lazo actual L = K*C*G.
% No modifica K. Devuelve C_out y un pequeño reporte con los nuevos márgenes.

% 1) Wcp del lazo actual
L_now = K * C_in * Gw;
[GM_now, PM_now, Wcg_now, Wcp_now] = margin(L_now);
if ~(isfinite(Wcp_now) && Wcp_now > 0)
error('No hay cruce de ganancia (Wcp) con el lazo actual. Ajustá K o añadí otra etapa antes.');
end
wc = Wcp_now;

% 2) Construir la etapa normalizada en wc
comp_type = lower(strtrim(comp_type));
if strcmpi(comp_type,'lead')
C_stage = lead_norm_at_wc(phi_deg, wc, s);      % |C(j wc)| = 1
elseif strcmpi(comp_type,'lag')
C_stage = lag_norm_at_wc(phi_deg, wc, s);       % |C(j wc)| = 1
else
error('comp_type debe ser "lead" o "lag".');
end

% 3) Encadenar etapa (K NO cambia)
C_out = C_stage * C_in;

% 4) Reporte (con K fijo)
L_new = K * C_out * Gw;
[GMf, PMf, Wcgf, Wcpf] = margin(L_new);
report = struct('GMdB', 20*log10(GMf), 'PM', PMf, 'Wcg', Wcgf, 'Wcp', Wcpf, ...
'wc_used', wc, 'type', comp_type, 'phi_deg', phi_deg);
end

function C = lead_norm_at_wc(phi_deg, wc, s)
phi_deg = max(1, min(60, phi_deg));
sphi = sind(phi_deg);
alpha = (1 - sphi) / (1 + sphi);       % 0<alpha<1
T = 1 / (wc * sqrt(alpha));
C0 = (1 + T*s) / (1 + alpha*T*s);
g = 1 / abs(freqresp(C0, 1j*wc));      % |C(jwc)|=1
C = g * C0;
end
```

Listing 2: Funciones creadas para este laboratorio.

```
function info_ext = plot_step_annot(sys, nameStr, Ts_override)
% STEP discreto con STAIRS; anota 10-90% (en s y en T), OS, zeta y omega_n.
% Grafica en milisegundos y marca el OS en el pico.

if nargin < 2 || isempty(nameStr), nameStr = inputname(1); end
if isempty(nameStr), nameStr = 'sys'; end

% ----- Respuesta y métricas básicas -----
[y, t] = step(sys); y = y(:); t = t(:);
t_ms = t * 1000;    % graficamos en milisegundos

% Estimar yss por promedio en cola
Ntail = max(10, round(0.05*length(y)));
yss = mean(y(end-Ntail+1:end));

% Límites 10-90
if abs(yss) < 1e-12
t10 = NaN; t90 = NaN; tr_10_90 = NaN; y10 = 0; y90 = 0;
else
```

```
y10 = 0.10*yss; y90 = 0.90*yss;
sgn = sign(yss);
idx10 = find(sgn*y >= sgn*y10, 1, 'first');
idx90 = find(sgn*y >= sgn*y90, 1, 'first');
t10 = tern(~isempty(idx10), t(idx10), NaN);
t90 = tern(~isempty(idx90), t(idx90), NaN);
tr_10_90 = tern(~isnan(t10)&&~isnan(t90)&&t90>=t10, t90 - t10, NaN);
end
t10_ms = t10 * 1000; t90_ms = t90 * 1000;

% Pico y %OS (para zeta y omega_n)
[ypeak_raw, idxpk] = max(sign(yss).*y);
ypeak = sign(yss)*ypeak_raw;
tpeak = t(idxpk); tpeak_ms = tpeak * 1000;
OS_percent = tern(yss ~= 0, max(0, (ypeak - yss)/abs(yss)*100), NaN);

% zeta desde %OS:  OS% = 100*exp(-zeta*pi/sqrt(1-zeta^2))
if isnan(OS_percent) || OS_percent <= 0
zeta_est = NaN;
else
logterm = log(OS_percent/100);
zeta_est = -logterm / sqrt(pi^2 + logterm^2);
end

% ----- Ts y rise en T -----
Ts = NaN;
try
if isprop(sys,'Ts') && ~isempty(sys.Ts) && sys.Ts > 0, Ts = sys.Ts; end
catch, end
if nargin >= 3 && ~isempty(Ts_override) && Ts_override > 0, Ts = Ts_override; end
tr_10_90_T = tern(~isnan(Ts) && ~isnan(tr_10_90), tr_10_90/Ts, NaN);

% ----- Estimacion de omega_n -----
omega_n = NaN;
if ~isnan(zeta_est) && zeta_est < 1 && OS_percent > 0 && tpeak > 0
omega_n = pi / ( tpeak * sqrt(1 - zeta_est^2) );
end
if isnan(omega_n)
S = stepinfo(y, t);    % Ts(2%) ~ 4/(zeta*omega_n)
if isfield(S,'SettlingTime') && ~isempty(S.SettlingTime) && S.SettlingTime > 0 && ~isnan(zeta_est) &&
    zeta_est > 0
omega_n = 4 / ( zeta_est * S.SettlingTime );
end
end
if isnan(omega_n) && ~isnan(tr_10_90) && tr_10_90 > 0
k_rise = 1.4;  % aprox para 10-90% en 2do orden subamortiguado
omega_n = k_rise / tr_10_90;
end

% ----- Grafico -----
figure;
stairs(t_ms, y, 'LineWidth', 1.5); grid on;grid minor; hold on;
xlabel('Tiempo [ms]'); ylabel('Salida');
title(sprintf('Respuesta al escalon de %s', nameStr));

% Líneas guía 10% y 90% y marcadores (en ms)
if ~isnan(tr_10_90)
yline(y10, '--', '10% yss', 'LabelHorizontalAlignment','left');
yline(y90, '--', '90% yss', 'LabelHorizontalAlignment','left');
xline(t10_ms, ':', 't10%');
xline(t90_ms, ':', 't90%');
plot([t10_ms t90_ms], [y10 y90], 'o', 'MarkerSize', 6, 'LineWidth', 1.2);
end
% Línea de yss
yline(yss, '-', sprintf('yss=%.3g', yss), 'LabelHorizontalAlignment','left');

% ----- Marca del OS en el pico -----
```

```
if ~isnan(OS_percent) && OS_percent > 0
% línea vertical desde yss a ypeak en tpeak
plot([tpeak_ms tpeak_ms], [yss ypeak], '-', 'LineWidth', 1.2);
% texto al costado derecho de la línea
dx = 0.02 * (t_ms(end) - t_ms(1));
text(tpeak_ms + dx, yss + 0.5*(ypeak - yss), ...
sprintf('OS=%.3g%%', OS_percent), ...
'Interpreter','none', 'Margin',2);
% marcar el pico
plot(tpeak_ms, ypeak, 's', 'MarkerSize', 6, 'LineWidth', 1.2);
end

% ----- Cuadro de metricas (abajo-derecha; chico) -----
w = 0.15; h = 0.12; rmargin = 0.1; x = 1 - rmargin - w; y0 = 0.30;
txtLines = {
    sprintf('t_r(10%%-90%%) = %.4g ms  (~%.4g Ts)', safeNum(tr_10_90*1000), safeNum(tr_10_90_T))
    sprintf('OS = %.4g%%',          safeNum(OS_percent))
    sprintf('zeta = %.4g',          safeNum(zeta_est))
    sprintf('omega_n = %.4g rad/s', safeNum(omega_n))
};
annotation('textbox', [x y0 w h], ...
'String', txtLines, 'Interpreter','none', ...
'FontName','Consolas', 'FontSize',9, ...
'EdgeColor',[0.5 0.5 0.5], 'FaceAlpha',0.88, 'BackgroundColor','w');

hold off;

% ----- Salida -----
info_ext = struct( ...
'yss', yss, ...
't10', t10, ...
't90', t90, ...
'tr_10_90', tr_10_90, ...
'tr_10_90_T', tr_10_90_T, ...
'OS_percent', OS_percent, ...
'zeta_est', zeta_est, ...
'omega_n', omega_n, ...
'Ts', Ts);
end

% --------- helpers ---------
function y = tern(cond, a, b)
if cond, y = a; else, y = b; end
end

function v = safeNum(x)
if isnan(x) || isinf(x), v = NaN; else, v = x; end
end

function [td, refd, yd, ud, ed, coefs] = sim_compensador_second_order( ...
Gd, z0_0, z0_1, p0_0, p0_1, Kc, T, umin, umax, refmin, refmax, n_per_seg)

% ===== Planta: y(k+1)=b0*u(k)+b1*u(k-1)+b2*u(k-2)-a1*y(k)-a2*y(k-1) =====
% Compensador (posición, 2º orden):
% u(k) = (p0_0+p0_1)*u(k-1) - (p0_0*p0_1)*u(k-2) ...
%      + Kc*[ e(k) - (z0_0+z0_1)*e(k-1) + (z0_0*z0_1)*e(k-2) ]
% e(k) = refd(k) - yd(k)

if nargin < 12 || isempty(n_per_seg), n_per_seg = 600; end
if ~isa(Gd,'tf'), error('Gd debe ser tf discreto.'); end

% --- Coefs planta (normalizados a a0=1) ---
[numD, denD] = tfdata(Gd, 'v');
numD = numD(:).'; denD = denD(:).';
if abs(denD(1) - 1) > 1e-12
numD = numD/denD(1);
denD = denD/denD(1);
```

```
end

% Asegurar longitud mínima del denominador [1 a1 a2]
if numel(denD) < 3, denD(end+1:3) = 0; end
a1 = denD(2);
a2 = denD(3);

% b0,b1,b2 sin romper si faltan términos en el numerador
b0 = 0; b1 = 0; b2 = 0;
if numel(numD) >= 2, b0 = numD(2); end
if numel(numD) >= 3, b1 = numD(3); end
if numel(numD) >= 4, b2 = numD(4); end
% Nota: esto respeta tu estilo previo (b0=numD(2), b1=numD(3), ...)

coefs = struct('b0',b0,'b1',b1,'b2',b2,'a1',a1,'a2',a2, ...
'z0_0',z0_0,'z0_1',z0_1,'p0_0',p0_0,'p0_1',p0_1,'Kc',Kc);

% --- Tiempo y referencia (refmin -> refmax -> refmin) ---
N  = 3*n_per_seg;
td = (0:N-1)' * T;
refd = [refmin*ones(n_per_seg,1);
refmax*ones(n_per_seg,1);
refmin*ones(n_per_seg,1)];

% --- Inicialización ---
yd = zeros(N,1);
ed = zeros(N,1);
ud = zeros(N,1);

% --- Precalculos del compensador ---
P1 = (p0_0 + p0_1);
P2 = (p0_0 * p0_1);
Z1 = (z0_0 + z0_1);
Z2 = (z0_0 * z0_1);

% --- Loop ---
for k = 3:N-1
ed(k) = refd(k) - yd(k);

% Compensador 2º orden
u_k = P1*ud(k-1) - P2*ud(k-2) + Kc*( ed(k) - Z1*ed(k-1) + Z2*ed(k-2) );

% Saturación
if u_k > umax
ud(k) = umax;
elseif u_k < umin
ud(k) = umin;
else
ud(k) = u_k;
end

% Planta discreta
yd(k+1) = b0*ud(k) + b1*ud(k-1) + b2*ud(k-2) - a1*yd(k) - a2*yd(k-1);
end
end

function [td, refd, yd, ud, ed, coefs] = sim_compensador_first_order( ...
Gd, z0, p0, Kc, T, umin, umax, refmin, refmax, n_per_seg)

% ===== Igual que tu PID: y(k+1)=b0*u(k)+b1*u(k-1)-a1*y(k)-a2*y(k-1) =====
% Controlador (posición): u(k) = p0*u(k-1) + Kc*ed(k) - Kc*z0*ed(k-1)
% ed(k) = refd(k) - yd(k)

if nargin < 10 || isempty(n_per_seg), n_per_seg = 600; end
if ~isa(Gd,'tf'), error('Gd debe ser tf discreto.'); end

% --- Coefs planta exactamente como en tu ejemplo ---
```

```
[numD, denD] = tfdata(Gd, 'v');
b0 = numD(2);
b1 = numD(3);
a1 = denD(2);
a2 = denD(3);
coefs = struct('b0',b0,'b1',b1,'a1',a1,'a2',a2);

% --- Tiempo y referencia (refmin -> refmax -> refmin) ---
N  = 3*n_per_seg;
td = (0:N-1)' * T;
refd = [refmin*ones(n_per_seg,1);
refmax*ones(n_per_seg,1);
refmin*ones(n_per_seg,1)];

% --- Inicialización idéntica a tu estilo ---
yd = zeros(N,1);
ed = zeros(N,1);
ud = zeros(N,1);

% --- Loop (misma estructura y orden que tu PID) ---
for k = 3:N-1
ed(k) = refd(k) - yd(k);

% Compensador: u(k) = p0*u(k-1) + Kc*ed(k) - Kc*z0*ed(k-1)
u_k = p0*ud(k-1) + Kc*ed(k) - Kc*z0*ed(k-1);


if u_k>umax
ud(k) = umax;
elseif u_k<umin
ud(k) = umin;
else
ud(k) = u_k;
end


% Planta discreta (dividido por a0 implícito como en tu ejemplo)
yd(k+1) = b0*ud(k) + b1*ud(k-1) - a1*yd(k) - a2*yd(k-1);
end
end

function [tr, ts, wn] = plot_step_info(G)
info = stepinfo(G);
tr = info.RiseTime;
ts = info.SettlingTime;
wn = 1.8 / tr;

t_end = 1.1 * ts;
if ~isfinite(t_end) || t_end <= 0, t_end = 5 * max(tr, 1e-3); end
t = linspace(0, t_end, 2*pi/wn);

[y, tout] = step(G, t);
figure; plot(tout*1000, y, 'LineWidth', 1.4); grid on;grid minor; hold on;
xlabel('Tiempo [ms]'); ylabel('Salida');
title(sprintf('Escalón: tr=%.4gms, ts=%.4gms, \\omega_n=%.4g rad/s', tr*1000, ts*1000, wn));
xline(tr*1000, '--', sprintf('  t_r=%.3g ms', tr*1000), 'LabelOrientation','horizontal');
xline(ts*1000,  '--', sprintf('  t_s=%.3g ms', ts*1000), 'LabelOrientation','horizontal');
legend('G(t)', 'Location', 'best');
end
```

Listing 3: Funciones reutilizadas de laboratorios pasados.

```
close all; clear; clc;

%% ====================== Parámetros nominales ======================
R_1 = 15e3;  R_2 = 82e3;  R_3 = 15e3;  R_4 = 82e3;
C_1 = 0.22e-6;  C_2 = 100e-9;
```

```
6
7  % Tolerancias de catálogo (especificación)
8  tolR = 0.05;    % ±5% resistencias
9  tolC = 0.20;    % ±20% capacitores
10
11 % Para Monte Carlo (3  tolerancia) ->  = tol/3
12 sigR = tolR/3;
13 sigC = tolC/3;
14
15 % Correlación entre resistores "pareja" del mismo valor (más realista)
16 rhoR = 0;   % entre R1-R3 (15k/15k) y R2-R4 (82k/82k)
17
18 % Nominales (alias)
19 R1n = R_1; R2n = R_2; R3n = R_3; R4n = R_4;
20 C1n = C_1; C2n = C_2;
21
22 %% ====================== Helper de planta continua ======================
23 % G(s) = [(R3/R1)*(R4/R2)] / [(R3*C2 s + 1)(R4*C1 s + 1)]
24 makeG = @(R1,R2,R3,R4,C1,C2) tf( (R3/R1)*(R4/R2), conv([R3*C2 1],[R4*C1 1]) );
25
26 %% ====================== Cargar datos medidos ======================
27 data = readmatrix('./imagenes/OpenLoop/mediciones.csv');
28
29 % u = ENTRADA medida; y = SALIDA medida
30 u = data(:, 11);    % ajustá si cambia columna
31 y = data(:, 17);
32
33 Ts = 0.000099999997474;  % tiempo de muestreo (100 µs)
34
35 % Vector de tiempo alineado a la medición
36 Ndata = min(length(y), length(u));
37 u = u(1:Ndata); y = y(1:Ndata);
38 u = u(:); y = y(:);
39 tdata = (0:Ndata-1).' * Ts;
40
41 %% ====================== Modelo estimado (opcional, referencia) ======================
42 try
43 data_id = iddata(y, u, Ts);
44 sys0 = tfest(data_id, 2,0);              % 2 polos por defecto
45 if sys0.Ts==0, sys0d = c2d(sys0, Ts, 'zoh'); else, sys0d = d2d(sys0, Ts); end
46 catch
47 sys0d = [];  % si no está el toolbox, seguimos sin el estimado
48 end
49
50 %% ====================== Nominal (discreto) ======================
51 Gc_nom = makeG(R1n,R2n,R3n,R4n,C1n,C2n);
52 Gd_nom = c2d(Gc_nom, Ts, 'zoh');
53
54 %% ====================== MONTE CARLO realista ======================
55 Nmc = 1000;            % # de muestras (subí si querés más suavidad)
56 rng(123);              % reproducible
57
58 % Ruido normal correlacionado para (R1,R3) y (R2,R4)
59 Zc13 = randn(Nmc,1);  % común par (R1,R3)
60 Z1   = randn(Nmc,1);  % propio R1
61 Z3   = randn(Nmc,1);  % propio R3
62
63 Zc24 = randn(Nmc,1);  % común par (R2,R4)
64 Z2   = randn(Nmc,1);  % propio R2
65 Z4   = randn(Nmc,1);  % propio R4
66
67 % Construir deltas relativos con correlación rhoR
68 a = rhoR; b = sqrt(1 - rhoR^2);
69 dR1 = sigR*(a*Zc13 + b*Z1);
70 dR3 = sigR*(a*Zc13 + b*Z3);
71 dR2 = sigR*(a*Zc24 + b*Z2);
72 dR4 = sigR*(a*Zc24 + b*Z4);
```

```
73
74 % Capacitores: independientes (distinto valor/naturaleza)
75 dC1 = sigC*randn(Nmc,1);
76 dC2 = sigC*randn(Nmc,1);
77
78 % Clip a las tolerancias físicas (no exceder ±tol)
79 clip = @(x,tol) min(max(x, -tol), tol);
80 dR1 = clip(dR1, tolR); dR3 = clip(dR3, tolR);
81 dR2 = clip(dR2, tolR); dR4 = clip(dR4, tolR);
82 dC1 = clip(dC1, tolC); dC2 = clip(dC2, tolC);
83
84 % Vectores absolutos
85 R1s = R1n*(1 + dR1);  R3s = R3n*(1 + dR3);
86 R2s = R2n*(1 + dR2);  R4s = R4n*(1 + dR4);
87 C1s = C1n*(1 + dC1);  C2s = C2n*(1 + dC2);
88
89 % Simular todas las muestras con la MISMA entrada u
90 Ymc = zeros(Ndata, Nmc);
91 for k = 1:Nmc
92 Gc  = makeG(R1s(k),R2s(k),R3s(k),R4s(k),C1s(k),C2s(k));
93 Gdz = c2d(Gc, Ts, 'zoh');
94 yk  = lsim(Gdz, u, tdata);
95 if isrow(yk), yk = yk.'; end
96 Ymc(:,k) = yk;
97 end
98
99 % Envolvente percentil (más realista) y también min/max por si querés ver extremos
100 yP05 = prctile(Ymc, 5,  2);
101 yP95 = prctile(Ymc, 95, 2);
102 yMin = min(Ymc, [], 2);
103 yMax = max(Ymc, [], 2);
104
105 %% ====================== (Opcional) WORST-CASE 64 corners ======================
106 plot_worstcase = false;    % ponelo true si querés superponerlo
107 if plot_worstcase
108 R1_lo = (1 - tolR)*R1n;   R1_hi = (1 + tolR)*R1n;
109 R2_lo = (1 - tolR)*R2n;   R2_hi = (1 + tolR)*R2n;
110 R3_lo = (1 - tolR)*R3n;   R3_hi = (1 + tolR)*R3n;
111 R4_lo = (1 - tolR)*R4n;   R4_hi = (1 + tolR)*R4n;
112 C1_lo = (1 - tolC)*C1n;   C1_hi = (1 + tolC)*C1n;
113 C2_lo = (1 - tolC)*C2n;   C2_hi = (1 + tolC)*C2n;
114
115 pairs = [ R1_lo R1_hi; R2_lo R2_hi; R3_lo R3_hi; R4_lo R4_hi; C1_lo C1_hi; C2_lo C2_hi ];
116 comb = dec2bin(0:(2^6-1)) - '0';    % 64x6
117
118 Ywc = zeros(Ndata, size(comb,1));
119 for k = 1:size(comb,1)
120 sel = comb(k,:)+1;
121 R1 = pairs(1,sel(1)); R2 = pairs(2,sel(2));
122 R3 = pairs(3,sel(3)); R4 = pairs(4,sel(4));
123 C1 = pairs(5,sel(5)); C2 = pairs(6,sel(6));
124 Gc  = makeG(R1,R2,R3,R4,C1,C2);
125 Gdz = c2d(Gc, Ts, 'zoh');
126 Ywc(:,k) = lsim(Gdz, u, tdata);
127 end
128 yWmin = min(Ywc, [], 2);
129 yWmax = max(Ywc, [], 2);
130 else
131 yWmin = []; yWmax = [];
132 end
133
134 %% ====================== Simulación nominal y estimado con misma u ======================
135 y_nom = lsim(Gd_nom, u, tdata);
136 if ~isempty(sysOd), y_est = lsim(sysOd, u, tdata); else, y_est = []; end
137
138 %% ====================== Plot comparativo ======================
139 figure; hold on; box on;
```

```
% Banda percentil (más realista)
fill([tdata; flipud(tdata)], [yP05; flipud(yP95)], [0.80 0.92 1.00], ...
'EdgeColor','none','FaceAlpha',0.55);

% (opcional) extremos worst-case en líneas finas
if ~isempty(yWmin)
plot(tdata, yWmin, 'Color',[0.2 0.4 0.8], 'LineStyle','--', 'LineWidth',0.8);
plot(tdata, yWmax, 'Color',[0.2 0.4 0.8], 'LineStyle','--', 'LineWidth',0.8);
end

% Medición y modelos
stairs(tdata, y, 'k', 'LineWidth', 1.3);          % medición
plot(tdata, y_nom, 'LineWidth', 1.2);             % nominal
if ~isempty(y_est), plot(tdata, y_est, 'LineWidth', 1.2); end   % estimado

xlabel('t [s]'); ylabel('y');
leg = {'Banda MC (-p5p95)'};
if ~isempty(yWmin), leg{end+1} = 'Worst-case min/max'; end
leg = [leg, {'Medición','Nominal'}];
if ~isempty(y_est), leg{end+1} = 'Estimado (tfest)'; end
legend(leg, 'Location','best');
title('Comparación temporal con la MISMA entrada u[k] - banda de tolerancia realista');
grid on; grid minor;

%% ====================== (Opcional) corner MC más parecido por RMSE ======================
RMSE = sqrt(mean( (Ymc - y).^2, 1 ));
[rmseBest, kBest] = min(RMSE);
fprintf('MC con menor RMSE: #%d  RMSE = %.4g\n', kBest, rmseBest);
best =
    struct('R1',R1s(kBest),'R2',R2s(kBest),'R3',R3s(kBest),'R4',R4s(kBest),'C1',C1s(kBest),'C2',C2s(kBest));
disp(best);
```

Listing 4: Script para la comparación de la planta en lazo abierto nominal vs mediciones.

```
close all
clear all

addpath('..\Lab1\')

%% Definicion de parametros
R_1 = 15e3;
R_3 = 15e3;
C_2 = 100e-9;

R_2 = 82e3;
R_4 = 82e3;
C_1 = 0.22e-6;

%% Generar funcion de transferencia d
numStage = [-R_3/R_1 -R_4/R_2];
denStage = { [C_2*R_3 1], [C_1*R_4 1] };

% Usamos celdas para guardar los tf de cada stage
Gstage = cell(1,2);
G = 1;
for i = 1:2
Gstage{i} = tf(numStage(i), denStage{i});
G = G*Gstage{i};
end

%% Analizamos en el tiempo
[tr, ts, wn] = plot_step_info(G);

disp('Planta continua G(s):');
%% Paso 1: Definir periodo de muestreo
% Se busca una mejora de 2 en el tiempo de rising =, o sea tr = 10 ms
```

```
N = 4;
T = tr/(8*N);
%Gcl = feedback(G,1);
%figure;
%step(Gcl)
%T = stepinfo(Gcl).SettlingTime/8;

%% Paso 2: Digitalizar con ZOH
Gd = c2d(G, T, 'zoh');
disp('Planta digital G(z):')
Gd




%% C1_K

data = readmatrix('.\C1_K\tablas.csv');  % lee todo el CSV en una matriz

t_meas   = data(:,4);    % columna 1 = tiempo
ref_meas = data(:,5);    % columna 2 = referencia
u_meas   = data(:,11);    % columna 3 = esfuerzo
y_meas   = data(:,17);    % columna 4 = salida

% Suponiendo que tenés vectores del osciloscopio:
% t_meas, ref_meas, u_meas, y_meas  (todos Nx1, en segundos y unidades reales)
opts = struct('Nini', 1, 'Nfin', 1, 'plot', true,'demean_errors',false);

S = sim_compensador_first_order_meas( ...
Gd, 0, 0, 7, T, 0, 4.08, ...
t_meas, ref_meas, u_meas, y_meas, ...
opts);

% Accesos:
S.errors.u     % métricas esfuerzo sim vs real
S.errors.y     % métricas salida  sim vs real

opts = struct('Nini', 1, 'Nfin', 1, 'plot', true);

S = sim_compensador_first_order_meas( ...
Gd, 0, 0, 7, T, 0, 4.08, ...
t_meas, ref_meas, u_meas, y_meas, ...
opts);

% Accesos:
S.errors.u     % métricas esfuerzo sim vs real
S.errors.y     % métricas salida  sim vs real


%% C1_Lead

data = readmatrix('.\C1_lead\tablas.csv');  % lee todo el CSV en una matriz

t_meas   = data(:,4);    % columna 1 = tiempo
ref_meas = data(:,5);    % columna 2 = referencia
u_meas   = data(:,11);    % columna 3 = esfuerzo
y_meas   = data(:,17);    % columna 4 = salida

% Suponiendo que tenés vectores del osciloscopio:
% t_meas, ref_meas, u_meas, y_meas  (todos Nx1, en segundos y unidades reales)
opts = struct('Nini', 1, 'Nfin', 1, 'plot', true);

S = sim_compensador_first_order_meas( ...
Gd, 0.622387144541200, 0.374311732736374,  1.184734400491065, T, 0, 4.08, ...
t_meas, ref_meas, u_meas, y_meas, ...
opts);
```

```
100
101 % Accesos:
102 S.errors.u     % métricas esfuerzo sim vs real
103 S.errors.y     % métricas salida  sim vs real
104
105 opts = struct('Nini', 1, 'Nfin', 1, 'plot', true,'demean_errors',true);
106
107 S = sim_compensador_first_order_meas( ...
108 Gd, 0.622387144541200, 0.374311732736374,  1.184734400491065, T, 0, 4.08, ...
109 t_meas, ref_meas, u_meas, y_meas, ...
110 opts);
111
112 % Accesos:
113 S.errors.u     % métricas esfuerzo sim vs real
114 S.errors.y     % métricas salida  sim vs real
115
116 %% C2
117
118 data = readmatrix('.\C2\tablas.csv');  % lee todo el CSV en una matriz
119
120 t_meas   = data(:,4);    % columna 1 = tiempo
121 ref_meas = data(:,5);    % columna 2 = referencia
122 u_meas   = data(:,11);    % columna 3 = esfuerzo
123 y_meas   = data(:,17);    % columna 4 = salida
124
125 % Suponiendo que tenés vectores del osciloscopio:
126 % t_meas, ref_meas, u_meas, y_meas  (todos Nx1, en segundos y unidades reales)
127 opts = struct('Nini', 1, 'Nfin', 1, 'plot', true,'demean_errors',false);
128
129 S = sim_compensador_second_order_meas( ...
130 Gd,-1,0.948680356607820,...
131 1,0.805044751405714,...
132 0.185620357009816,...
133 T, 0, 4.08, ...
134 t_meas, ref_meas, u_meas, y_meas, ...
135 opts);
136
137
138 opts = struct('Nini', 1, 'Nfin', 1, 'plot', true,'demean_errors',true);
139
140 S = sim_compensador_second_order_meas( ...
141 Gd,-1,0.948680356607820,...
142 1,0.805044751405714,...
143 0.185620357009816,...
144 T, 0, 4.08, ...
145 t_meas, ref_meas, u_meas, y_meas, ...
146 opts);
147
148 % Accesos:
149 S.errors.u     % métricas esfuerzo sim vs real
150 S.errors.y     % métricas salida  sim vs real
```

Listing 5: Script utilizado para la comparación de las simulaciones con las mediciones de los compensadores.

Apéndice B
Código desarrollado para el PSoC

```
1
2 #include "project.h"
3 #include <stdio.h>
4 #include <string.h>
5 #include <stdlib.h>
6 #include <ctype.h>
7
8 /* ==== Prototipos ==== */
9 CY_ISR_PROTO(adquirirMuestra);
```

```
static void pc_process_rx(void);
static void handle_command(const char *line);
static void set_ref_value(float *target, float val, const char *name);
static void set_ref_period(uint16_t period);
static void ref_status(void);

/* ==== Config general / límites ==== */
#define SAT_MIN        (0.0f)
#define SAT_MAX        (4.08f)
#define MIDDLE_VOLTAGE ((SAT_MAX - SAT_MIN)/2.0f)

#define RX_BUF_SZ 64

/* ==== Estado global ==== */
volatile char  flag    = '0';
volatile float ref_max = (float)(MIDDLE_VOLTAGE + 0.5f);
volatile float ref_min = (float)(MIDDLE_VOLTAGE - 0.5f);
volatile float ref     = 0.0f;
volatile float muestra = 0.0f;
volatile float e       = 0.0f;

/* Modo de operación: 0 = closed loop (PID), 1 = open loop (DAC = ref) */
enum { MODE_CLOSED = 0, MODE_OPEN = 1 };
volatile uint8 mode = MODE_CLOSED;

/* ==== UART RX line buffer ==== */
static char   rx_buf[RX_BUF_SZ];
static uint8  rx_len = 0;

/* ==== Helpers sin libm ==== */
static long my_lroundf(float x) { return (x >= 0.0f) ? (long)(x + 0.5f) : (long)(x - 0.5f); }

/* Mapear voltaje a 0..255 con clamps a [SAT_MIN, SAT_MAX] */
static inline uint8 volt_to_dac(float v)
{
        if (v < SAT_MIN) v = SAT_MIN;
        if (v > SAT_MAX) v = SAT_MAX;
        float norm = v / SAT_MAX;               /* 0..1 */
        if (norm < 0.0f) norm = 0.0f;
        if (norm > 1.0f) norm = 1.0f;
        return (uint8)(norm * 255.0f);
}

/* ==== ISR de muestra ==== */
CY_ISR(adquirirMuestra)
{
        /* Limpiar fuente de interrupción de "muestra "disponible */
        muestra_disponible_ClearPending();

        /* Conmutar referencia según pin/switch (1 -> max, 0 -> min) */
        ref = (ref_state_Read()) ? ref_max : ref_min;

        VDAC8_ref_SetValue(volt_to_dac(ref));

        /* Leer ADC y calcular error */
        muestra = VADC_CountsTo_Volts(VADC_GetResult16());
        e = ref - muestra;

        flag = '1';

        /* Según tu diseño, deshabilitás y re-habilitás en el main */
        muestra_disponible_Disable();
}

/* ==== Controlador discreto (forma recursiva estándar) ==== */
/* Parámetros (ajustá a gusto) */
#define T   (double)(0.001245213061220)
```

```
#define Ts  T
//
//
//#define U0  -0.737363460561365/1.18473440049107
//#define U1  1
//#define U2  0
//
//#define E0  -0.737363460561365
//#define E1  1.18473440049107
//#define E2  0


#define z0  -1//0//0.6224
#define z1  0.948680356607820//0
#define p0  1.00000000000000//0//0.3743
#define p1  0.805044751405714//0
#define K  0.185620357009816 //7//1.1847


#define U0  1
#define U1  -(p0+p1)
#define U2  p0*p1


#define E0  1
#define E1  -(z0+z1)
#define E2 z0*z1



/* Hook del esfuerzo: respeta el modo (open/closed) */
static inline void actualizarEsfuerzo(void)
{
        if (mode == MODE_OPEN) {
                /* === LAZO ABIERTO: el DAC sigue la referencia (ref_min / ref_max) === */
                VDAC8_SetValue(volt_to_dac(ref));
                return;
        }

        /* === LAZO CERRADO: controlador PID discreto === */
        static float u = 0.0f, u_1 = 0.0f, u_2 = 0.0f;
        static float e_1 = 0.0f, e_2 = 0.0f;

        /* Salida no saturada */
        float u_unsat = (float)((-U1*u_1 + -U2*u_2 + K*(E0*e + E1*e_1 + E2*e_2))/U0);

        /* Saturación correcta (sobre u_unsat) */
        float u_sat;
        if (u_unsat > SAT_MAX)      u_sat = SAT_MAX;
        else if (u_unsat < SAT_MIN) u_sat = SAT_MIN;
        else                        u_sat = u_unsat;

        /* DAC: escribir esfuerzo saturado */
        VDAC8_SetValue(volt_to_dac(u_sat));

        /* Correr estados (orden correcto) */
        u_2 = u_1; u_1 = u_sat;
        e_2 = e_1; e_1 = e;
}

/* ==== Helpers de parsing ==== */

/* Recorta espacios a la derecha (CR/LF/espacios) */
static void rstrip(char *s)
{
        size_t n = strlen(s);
        while (n && (s[n-1]=='\r' || s[n-1]=='\n' || isspace((unsigned char)s[n-1])))
```

```
144         s[--n] = '\0';
145 }
146
147 static void to_lower_str(char *s)
148 {
149         for (; *s; ++s) *s = (char)tolower((unsigned char)*s);
150 }
151
152 /* Procesa bytes RX por líneas (eco por línea) */
153 static void pc_process_rx(void)
154 {
155         int ch;
156         while ((ch = PC_GetChar()) != 0) {
157                 if (ch == '\n' || ch == '\r') {
158                         if (rx_len > 0) {
159                                 rx_buf[rx_len] = '\0';
160                                 PC_PutString(rx_buf); PC_PutString("\r\n"); /* eco de línea */
161                                 handle_command(rx_buf);
162                                 rx_len = 0;
163                         }
164                 } else {
165                         if (rx_len < (RX_BUF_SZ-1)) {
166                                 rx_buf[rx_len++] = (char)ch;
167                         } else {
168                                 rx_len = 0;
169                                 PC_PutString("ERR: line too long\r\n");
170                         }
171                 }
172         }
173 }
174
175 /* ==== Status command (sin floats en printf; usa mV) ==== */
176 static void ref_status(void)
177 {
178         long ref_min_mV = my_lroundf(ref_min * 1000.0f);
179         long ref_max_mV = my_lroundf(ref_max * 1000.0f);
180         long ref_mV     = my_lroundf(ref * 1000.0f);
181         unsigned per    = (unsigned)timer_ref_ReadPeriod();
182
183         char msg[160];
184         snprintf(msg, sizeof(msg),
185         "STATUS:\r\n"
186         "  mode=%s\r\n"
187         "  ref_min=%ld mV\r\n"
188         "  ref_max=%ld mV\r\n"
189         "  ref=%ld mV\r\n"
190         "  ref_period=%u\r\n",
191         (mode == MODE_OPEN) ? "open" : "closed",
192         ref_min_mV, ref_max_mV, ref_mV, per);
193         PC_PutString(msg);
194 }
195
196 /* ==== Parser de comandos ==== */
197 static void handle_command(const char *line_in)
198 {
199         char line[RX_BUF_SZ];
200         strncpy(line, line_in, sizeof(line));
201         line[sizeof(line)-1] = '\0';
202         rstrip(line);
203         to_lower_str(line);
204
205         /* Comandos sin valor */
206         if (strcmp(line, "ref_status") == 0 || strcmp(line, "status") == 0) {
207                 ref_status(); return;
208         }
209         if (strcmp(line, "help") == 0 || strcmp(line, "?") == 0) {
210                 PC_PutString("Commands:\r\n"
```

```
                "  ref_max:<float 0..4.08>\r\n"
                "  ref_min:<float 0..4.08>\r\n"
                "  ref_period:<uint16 0..65535>\r\n"
                "  mode:<open|closed|1|0>\r\n"
                "  ref_status\r\n");
                return;
        }

        /* Clave:valor */
        char *colon = strchr(line, ':');
        if (!colon) { PC_PutString("ERR: expected key:value or ref_status\r\n"); return; }
        *colon = '\0';
        const char *key = line;
        const char *val_str = colon + 1;
        while (*val_str && isspace((unsigned char)*val_str)) val_str++;

        if (strcmp(key, "ref_max") == 0) {
                float v = (float)atof(val_str);
                set_ref_value((float*)&ref_max, v, "ref_max");
        } else if (strcmp(key, "ref_min") == 0) {
                float v = (float)atof(val_str);
                set_ref_value((float*)&ref_min, v, "ref_min");
        } else if (strcmp(key, "ref_period") == 0) {
                long v = strtol(val_str, NULL, 0);
                if (v < 0 || v > 65535) PC_PutString("ERR: ref_period out of range (0..65535)\r\n");
                else set_ref_period((uint16_t)v);
        } else if (strcmp(key, "mode") == 0) {
                if (strcmp(val_str, "open") == 0 || strcmp(val_str, "1") == 0) {
                        mode = MODE_OPEN;  PC_PutString("OK: mode=open\r\n");
                } else if (strcmp(val_str, "closed") == 0 || strcmp(val_str, "0") == 0) {
                        mode = MODE_CLOSED; PC_PutString("OK: mode=closed\r\n");
                } else {
                        PC_PutString("ERR: mode must be open|closed|1|0\r\n");
                }
        } else {
                PC_PutString("ERR: unknown key. Use ref_max, ref_min, ref_period, mode, ref_status\r\n");
        }
}

/* Set de ref_* con validación y sección crítica */
static void set_ref_value(float *target, float val, const char *name)
{
        if (val < SAT_MIN || val > SAT_MAX) {
                char msg[64];
                long lo = my_lroundf(SAT_MIN*1000.0f), hi = my_lroundf(SAT_MAX*1000.0f);
                snprintf(msg, sizeof(msg), "ERR: %s out of range (%ld..%ld mV)\r\n", name, lo, hi);
                PC_PutString(msg);
                return;
        }

        uint8 intr = CyEnterCriticalSection();
        *target = val;
        /* Si está activa esa referencia, actualizá ref inmediatamente */
        ref = (ref_state_Read()) ? ref_max : ref_min;
        CyExitCriticalSection(intr);

        char msg[64];
        long v_mV = my_lroundf(val*1000.0f);
        snprintf(msg, sizeof(msg), "OK: %s=%ld mV\r\n", name, v_mV);
        PC_PutString(msg);
}

/* Cambia el periodo del timer de referencia de forma segura */
static void set_ref_period(uint16_t period)
{
        timer_ref_Stop();
        timer_ref_WritePeriod(period);
```

```
278         timer_ref_WriteCounter(period);
279         timer_ref_Start();
280 
281         char msg[64];
282         snprintf(msg, sizeof(msg), "OK: ref_period=%u\r\n", (unsigned)period);
283         PC_PutString(msg);
284 }
285 
286 /* ==== main ==== */
287 int main(void)
288 {
289         CyGlobalIntEnable;
290 
291         /* HW init */
292         Opa_stage1_Start();
293         Opa_vdda_2_Start();
294         Opa_stage2_Start();
295         VADC_Start();
296         VDAC8_Start();
297         VDAC8_ref_Start();
298 
299         /* UART PC */
300         PC_Start();
301         PC_PutString("\r\nReady. Commands:\r\n");
302         PC_PutString("  ref_max:<float 0..4.08>\r\n");
303         PC_PutString("  ref_min:<float 0..4.08>\r\n");
304         PC_PutString("  ref_period:<uint16 0..65535>\r\n");
305         PC_PutString("  mode:<open|closed|1|0>\r\n");
306         PC_PutString("  ref_status\r\n");
307 
308         /* Timer / ISR de muestra */
309         muestra_disponible_StartEx(adquirirMuestra);
310         muestra_disponible_Enable();
311         timer_ref_Start();
312 
313         flag = '0';
314 
315         for (;;)
316         {
317                 /* Procesar UART sin bloquear */
318                 pc_process_rx();
319 
320                 /* Procesar muestra si disponible */
321                 if (flag == '1') {
322                         flag = '0';
323                         actualizarEsfuerzo();           /* respeta el modo */
324                         muestra_disponible_Enable();    /* re-habilitar ISR */
325                 }
326         }
327 }
```

Listing 6: Código desarrollado para la implementación de los compensadores con el PSoC en lenguaje C.