Práctica de Laboratorio 5: Ubicación Arbitraria de Polos.

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

En esta práctica se aborda el diseño de controladores mediante la técnica de **ubicación arbitraria de polos**, un método fundamental en el análisis y síntesis de sistemas de control en el espacio de estados. El propósito principal es modificar la dinámica de una planta determinada para que el sistema en lazo cerrado cumpla con *especificaciones deseadas de respuesta transitoria*, tales como el tiempo de establecimiento y el coeficiente de amortiguamiento.

A través del cálculo de las matrices del sistema, su discretización e implementación en un **controlador digital basado en el PSoC**, se busca comprender cómo la realimentación de estados permite alterar el comportamiento dinámico y mejorar el desempeño del sistema.

Finalmente, se comparan los resultados obtenidos mediante simulación en `MATLAB` con los resultados experimentales, analizando la efectividad del diseño y las posibles diferencias entre el modelo teórico y la práctica.

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.
- Asegurar que el sistema regulado sea estable.
- Observar y analizar los efectos del controlador en el comportamiento dinámico del sistema.
- Considerar distintos métodos para el ajuste de los parámetros del controlador y analizar los resultados obtenidos.
- Diseñar el sistema de control en `MATLAB` e implementar la ecuación en diferencia correspondiente en el `PSoC`.

II. Desarrollo

II-A. Modelado del Sistema

II-A1. Obtención de las matrices del sistema F, G, H y J: Para el modelado del sistema se parte del circuito mostrado en la Figura 1, a partir del cual se determinan las ecuaciones de estado mediante el análisis de los lazos de realimentación y las relaciones de tensión en los componentes.

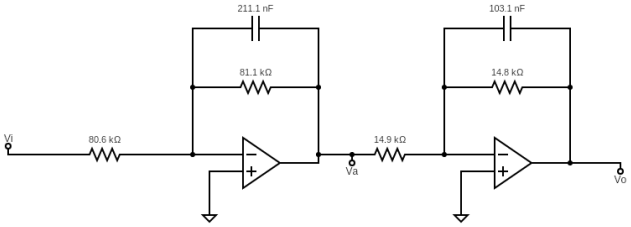


Figura 1. Circuito de la planta.

El sistema se describe mediante las siguientes ecuaciones en espacio de estados:

$$\dot{x}(t) = F\,x(t) + G\,V_i(t) \tag{1}$$

$$y(t) = H\,x(t) + J\,V_i(t) \tag{2}$$

Para obtener las expresiones de las variables de estado, se parte del equivalente del paralelo entre un resistor y un capacitor:

$$R \parallel \frac{1}{sC} = \frac{R}{1+sRC}$$

Considerando que ambos amplificadores operacionales se encuentran en configuración no inversora, se obtienen las siguientes relaciones:

$$V_a = \frac{-R_2}{1+sR_2C_1}\frac{V_i}{R_1} \quad \Rightarrow \quad sV_a = -\frac{1}{R_1C_1}V_i - \frac{1}{R_2C_1}V_a$$

$$V_o = -\frac{R_4}{1+sR_4C_2}\frac{V_a}{R_3} \quad \Rightarrow \quad sV_o = -\frac{1}{R_3C_2}V_a - \frac{1}{R_4C_2}V_o$$

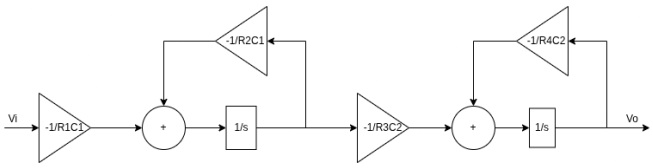


Figura 2. Diagrama de bloques del sistema continuo.

Definiendo como variables de estado $x_1(t) = V_a$ y $x_2(t) = V_o$, las ecuaciones anteriores se expresan en forma matricial como:

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}-\dfrac{1}{R_2C_1} & 0\\ -\dfrac{1}{R_3C_2} & -\dfrac{1}{R_4C_2}\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}-\dfrac{1}{R_1C_1}\\ 0\end{bmatrix} V_i(t)$$

y la ecuación de salida queda definida como:

$$y(t) = \begin{bmatrix}0 & 1\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + 0 \cdot V_i(t)$$

Sustituyendo los valores de los componentes $C_1 = 211.1 \times 10^{-9}\,\text{F}$, $R_1 = 80.55 \times 10^3\,\Omega$, $R_2 = 81.09 \times 10^3\,\Omega$, $C_2 = 103.07 \times 10^{-9}\,\text{F}$, $R_3 = 14.878 \times 10^3\,\Omega$ y $R_4 = 14.76 \times 10^3\,\Omega$, se obtienen las siguientes matrices numéricas:

$$F = \begin{bmatrix}-58.42 & 0\\ -652.11 & -657.37\end{bmatrix}, \quad G = \begin{bmatrix}-58.81\\ 0\end{bmatrix}, \quad H = \begin{bmatrix}0 & 1\end{bmatrix}$$
$$y \quad J = 0 \tag{3}$$

II-A2. Mostrar el diagrama de bloques del sistema:

II-B. Discretización del Sistema

II-B1. Elección del tiempo de muestreo $T_s = 1$ ms: Para la discretización del sistema continuo descrito por las ecuaciones (1) y (2), se busca obtener un modelo equivalente en tiempo discreto que relacione las variables de estado y la señal de entrada en instantes de muestreo definidos. Las ecuaciones del sistema discreto se expresan como:

$$X(k+1) = A\,X(k) + B\,u(k) \tag{4}$$

$$Y(k) = C\,X(k) + D\,u(k) \tag{5}$$

Usando las matrices continuas F, G, H y J obtenidas previamente, las matrices discretas se determinan mediante las siguientes expresiones:

$$A = \mathrm{e}^{FT_s}, \qquad B = F^{-1}\big(\mathrm{e}^{FT_s} - I\big)\,G, \qquad C = H$$

$$\&\quad D = J$$

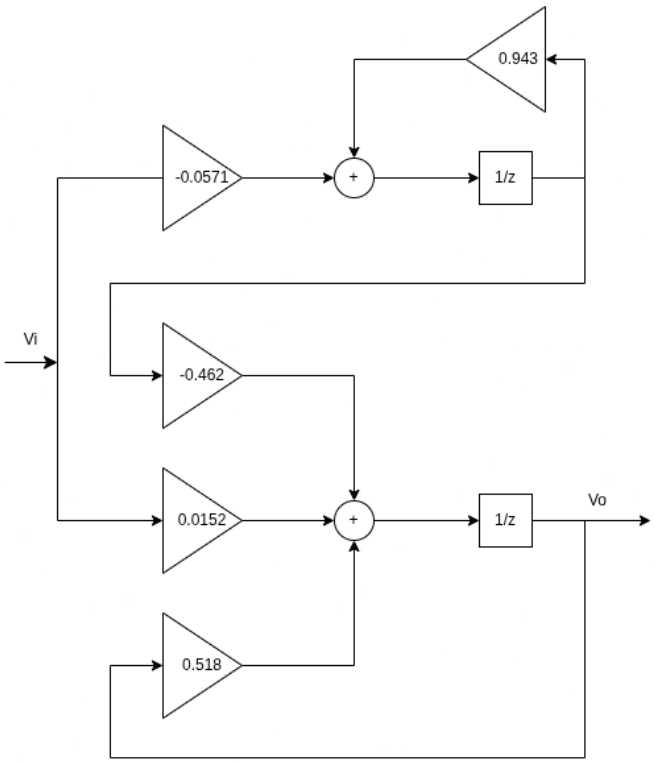


Figura 3. Diagrama de bloques del sistema discretizado.

II-B2. Obtención de las matrices discretas A, B, C y D: Con un tiempo de muestreo $T_s = 1$ ms, se obtienen las siguientes matrices discretizadas:

$$A = \begin{bmatrix}0.943 & 0\\ -0.462 & 0.518\end{bmatrix}, \qquad B = \begin{bmatrix}-0.0571\\ 0.0152\end{bmatrix},$$
$$C = \begin{bmatrix}0 & 1\end{bmatrix}, \qquad \& \qquad D = 0 \tag{6}$$

Estas matrices representan el modelo digital equivalente del sistema continuo, y serán utilizadas posteriormente para el diseño del controlador e implementación en el `PSoC`.

II-B3. Diagrama de bloques del sistema discretizado: En la Figura 3 se presenta el diagrama de bloques correspondiente al sistema discretizado, donde se observa la relación entre las variables de estado, la entrada $u(k)$ y la salida $Y(k)$.

II-B4. Verificación de la controlabilidad y observabilidad del sistema: La matriz de controlabilidad se obtiene a partir de la siguiente relación general:

$$x(n) - A^n x(0) = \sum_{i=0}^{n-1} A^{n-i-1} B\,u(i)$$

lo que lleva a la siguiente forma matricial:

$$x(n) - A^n x(0) = \begin{bmatrix}A^{n-1}B & A^{n-2}B & \cdots & AB & B\end{bmatrix}\begin{bmatrix}u(0)\\ u(1)\\ \vdots\\ u(n-1)\end{bmatrix}$$

De esta expresión, se define la matriz de controlabilidad como:

$$\mathcal{C} = \begin{bmatrix} B & AB \end{bmatrix}$$

Para el sistema analizado, la matriz resultante es:

$$\mathcal{C} = \begin{bmatrix} -0.05344 & -0.05713 \\ 0.03435 & 0.01527 \end{bmatrix}$$

El rango de esta matriz es $n = 2$, lo que indica que el sistema es completamente controlable.

La matriz de observabilidad se obtiene a partir de la expresión general:

$$Y(n-1) = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix} X(0)$$

Por lo tanto, la matriz de observabilidad queda definida como:

$$\mathcal{O} = \begin{bmatrix} C \\ CA \end{bmatrix}$$

Sustituyendo los valores del sistema:

$$\mathcal{O} = \begin{bmatrix} 0 & 1 \\ -0.462 & 0.518 \end{bmatrix}$$

El rango de la matriz de observabilidad también resulta ser $n = 2$.
Por lo tanto, se concluye que el sistema es **completamente controlable y observable**, cumpliendo con las condiciones necesarias para el diseño de control mediante realimentación de estados.

II-B5. Comparar los resultados obtenidos con las simulaciones realizadas en MATLAB: **Resultados de las matrices continuas:**

$$F = \begin{bmatrix} -58.42 & 0 \\ -652.10 & -657.30 \end{bmatrix}, \quad G = \begin{bmatrix} -58.81 \\ 0 \end{bmatrix}, \quad H = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$\& \quad J = [\,0\,]$$

Resultados de las matrices discretas:

$$A = \begin{bmatrix} 0.9433 & 0 \\ -0.4628 & 0.5182 \end{bmatrix}, \quad B = \begin{bmatrix} -0.05712 \\ 0.01527 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$\& \quad D = [\,0\,]$$

Se observa que los resultados de las matrices obtenidas son congruentes con los valores calculados en las ecuaciones (3) y (6), verificando la coherencia entre el modelo teórico y los resultados obtenidos mediante MATLAB.

En las figuras 4 y 5, del MATLAB se pueden observar que no es implementable.

III. Implementación del Sistema

Durante la etapa de implementación no se intentó replicar directamente los controladores diseñados en *Matlab* con las especificaciones originales, ya que se determinó que dichos parámetros no eran implementables con el hardware disponible. Los valores de ganancia requeridos generaban esfuerzos imposibles de aplicar sin saturar el DAC, por lo que incluso antes de probar en el PSoC se optó por **relajar las especificaciones** y diseñar controladores implementables que preservaran el comportamiento cualitativo del sistema. En otras palabras, se mantuvo el mismo enfoque de control por realimentación de estados, pero con polos menos agresivos y ganancias adaptadas al rango físico del actuador. Aun así, los controladores "relajados" solamente pudieron trabajar con referencias muy pequeñas ($0{,}4$ V para el Caso 1 y $0{,}3$ V para el Caso 2); valores mayores producían saturación inmediata y lo único observable en el osciloscopio era ruido amplificado. El sistema digital completo se muestra en la Figura 6. Allí se incluyen los convertidores ADC SAR, el circuito de sincronización con dos flip-flops tipo D para detectar el fin de conversión de ambos canales, y un módulo UART utilizado para enviar comandos al PSoC. A través de este canal se podían modificar en tiempo real la frecuencia del temporizador, abrir o cerrar el lazo, y cambiar la amplitud de referencia, lo cual facilitó enormemente las pruebas experimentales.

La planta analógica se muestra en la Figura 10. Junto a ella se encuentra el DAC principal, que actúa como actuador aplicando los esfuerzos de control calculados. Además, se incorporó un segundo DAC destinado a generar la referencia en voltios; si bien esta referencia se define digitalmente, resultó útil disponer de su equivalente analógico para visualizarla en el osciloscopio durante las pruebas.

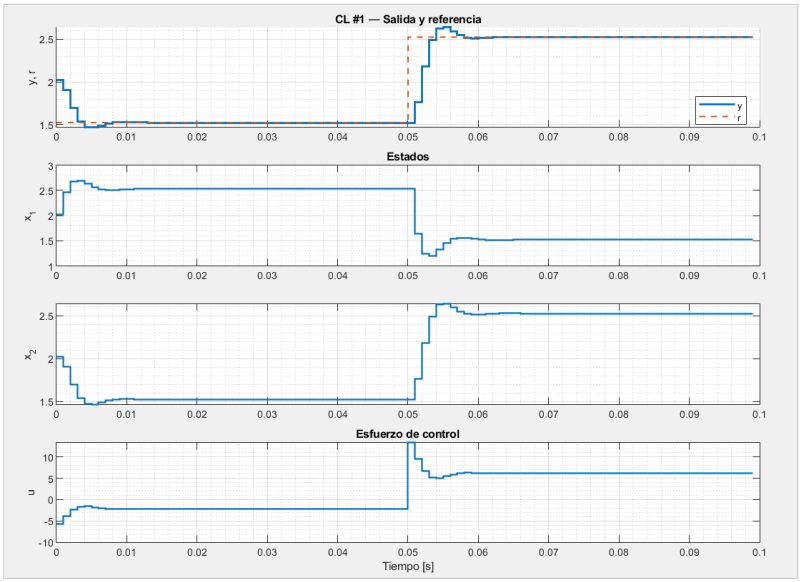
La referencia digital se generó mediante un temporizador configurable y un registro tipo T,implementado mediante uno de tipo D, definiendo el valor de referencia como $\mathrm{refA}/2$ cuando el bit lógico leído era 1 y $-\mathrm{refA}/2$ cuando era 0. De esta forma se obtuvo una señal cuadrada de amplitud controlada, útil para evaluar las respuestas transitorias del sistema.

Es importante destacar que todos los cálculos del controlador se realizaron desacoplando la componente de DC, ya que al no hacerlo se amplifica la componente continua en la etapa de salida y se produce saturación mucho más rápida.

En el firmware, para facilitar el traslado del diseño desde *Matlab* a lenguaje C, se implementó la librería `mat.c`. En ella se definieron los tipos de datos y funciones necesarios para realizar las operaciones matriciales requeridas en el cálculo del esfuerzo de control, utilizando únicamente las ganancias K y N_{var}.

Una vez calculado el esfuerzo de control (en valor alterno), se le sumó $V_{\mathrm{DD}}/2$ para montarlo sobre el nivel medio del DAC y mantener compatibilidad con el circuito analógico. De esta manera se logró que las mediciones en el osciloscopio fuesen consistentes con las simulaciones en *Matlab*.

Las frecuencias de muestreo efectivas fueron de aproximadamente $418{,}5$ Hz para el Caso 1 y $313{,}9$ Hz para el



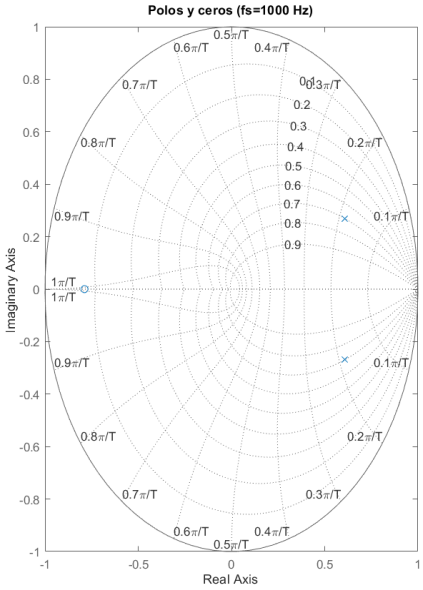
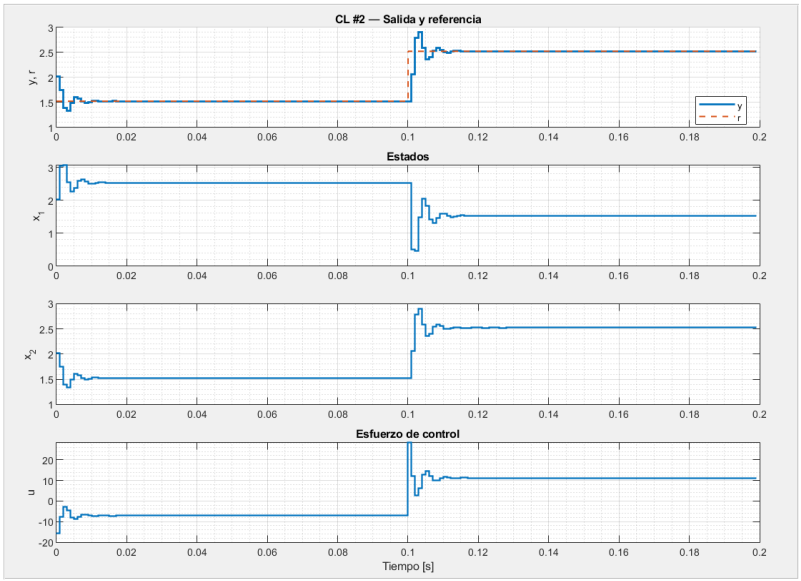


Figura 4. Respuesta simulada en MATLAB correspondiente al Caso 2($f_s = 1000Hz$).



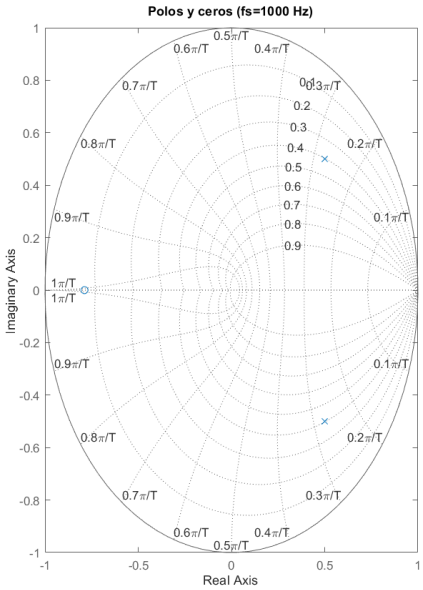


Figura 5. Respuesta simulada en MATLAB correspondiente al Caso 2($f_s = 1000Hz$).

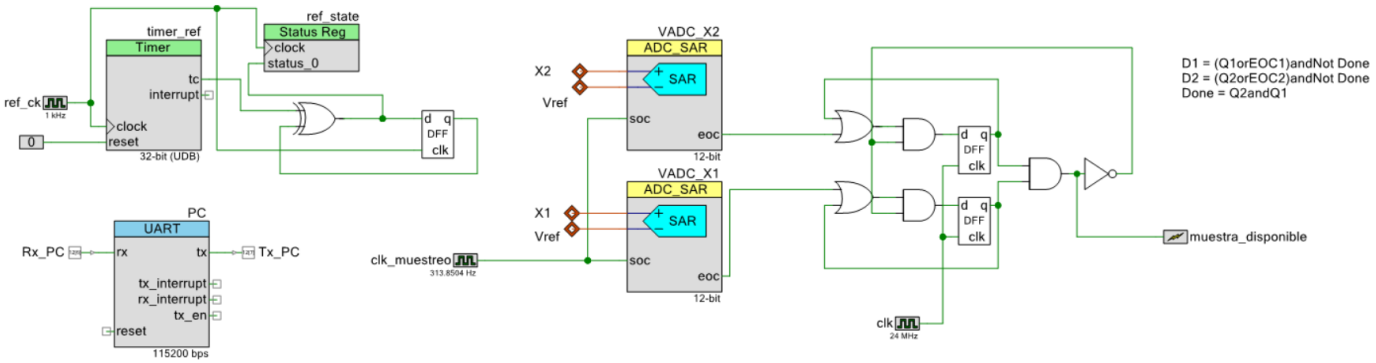


Figura 6. Circuito digital implementado en el PSoC, con los módulos SAR, flip-flops de sincronización y UART para control.

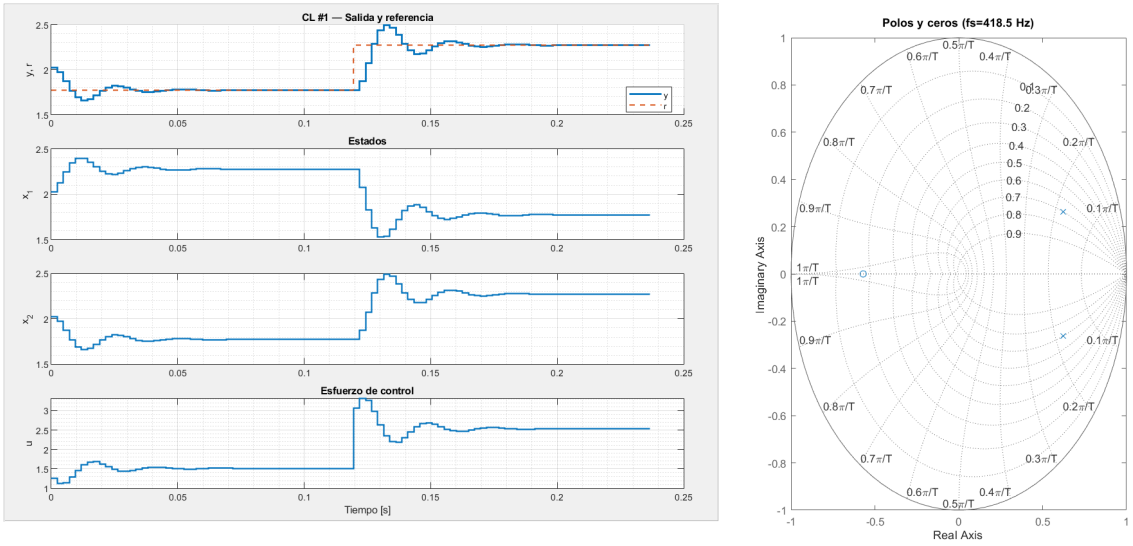


Figura 7. Respuesta simulada correspondiente al Caso 1 ($f_s \approx 418{,}5$ Hz).

Caso 2. Las Figuras 12 y 13 muestran las respuestas observadas experimentalmente.

Aunque las señales obtenidas presentan cierto nivel de ruido —algo esperable dada su baja amplitud—, se lograron implementar controladores funcionales y estables dentro de las limitaciones del hardware disponible. Las respuestas observadas mantienen la forma y tendencia previstas en las simulaciones, aunque restringidas a un rango reducido de amplitudes.

IV. Resultados

IV-A. *Comparación simulación vs. experimento*

La comparación entre la simulación y las mediciones experimentales se presenta en las Figuras 14 y 15. En ambos casos se grafican, para las señales x_1, $x_2 (= y)$ y u, la traza experimental frente a la simulada y, en paneles adyacentes, el error correspondiente. Las figuras fueron generadas directamente desde el script de *Matlab* que remuestrea (por ZOH) las señales simuladas a la grilla temporal del osciloscopio, asegurando una comparación punto a punto usando la misma referencia. De esta forma se garantiza que los resultados reflejen únicamente las diferencias reales entre el modelo teórico y la implementación física.

IV-B. *Métricas cuantitativas*

A continuación se resumen las métricas principales para cada señal: **RMSE absoluto**, **RMSE porcentual**, **error máximo absoluto** y **error máximo porcentual**. La Tabla I corresponde

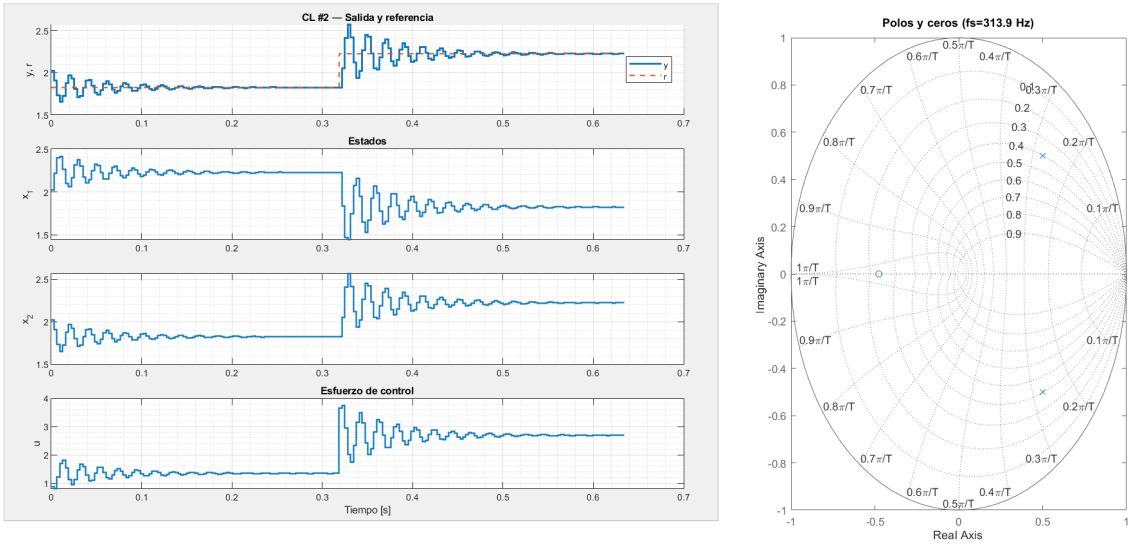


Figura 8. Respuesta simulada correspondiente al Caso 2 ($f_s \approx 313{,}9$ Hz).

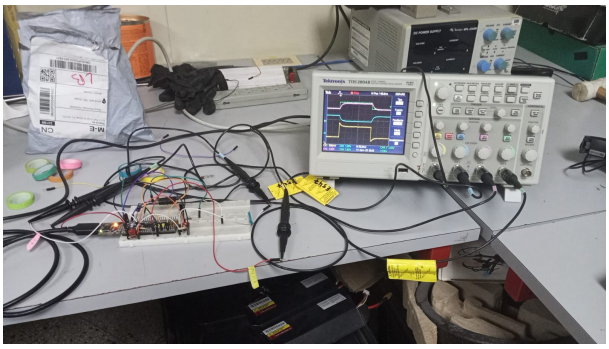


Figura 9. Fotografía de la práctica.

al *Caso 1 (Exp1)* y la Tabla II queda preparada para el *Caso 2 (Exp2)*.

Cuadro I
MÉTRICAS DE COMPARACIÓN — CASO 1 (EXP1).

Señal	RMSE_abs	RMSE_ %	ErrMax_abs	ErrMax_ %
X1	0,055	15,542	0,259	73,451
X2	0,057	15,466	0,308	83,691
U	0,298	28,241	1,369	129,720

Cuadro II
MÉTRICAS DE COMPARACIÓN — CASO 2 (EXP2).

Señal	RMSE_abs	RMSE_ %	ErrMax_abs	ErrMax_ %
X1	0,097	23,315	0,400	96,301
X2	0,085	20,500	0,333	80,455
U	0,564	38,199	1,938	131,318

Otros/circuitoAnalogico.png

Figura 10. Circuito analógico correspondiente a la planta física utilizada en el experimento.

IV-C. Discusión y conclusiones

Se observa que la respuesta de la planta sigue la forma esperada: el *rising time* es muy similar al de la simulación y la señal permanece dentro de los valores previstos. El error medio se concentra alrededor de cero salvo en las zonas próximas a la saturación del actuador, donde aumentan las discrepancias, atribuibles a no linealidades de la planta y a las limitaciones del DAC. Además, el esfuerzo medido se aprecia como una *versión achatada* del esfuerzo simulado, consistente con la compresión que introduce la saturación y el rango dinámico

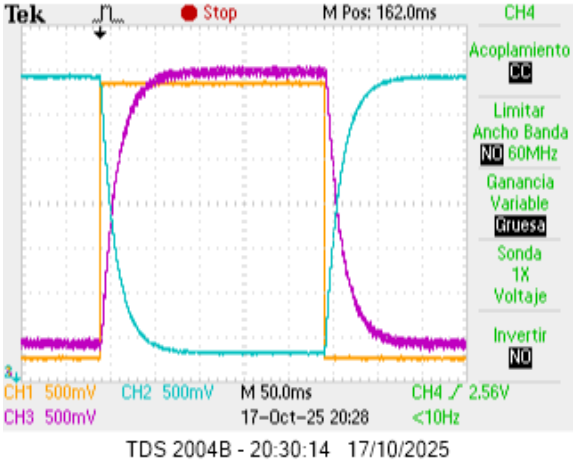


Figura 11. Respuesta experimental del sistema en lazo abierto.

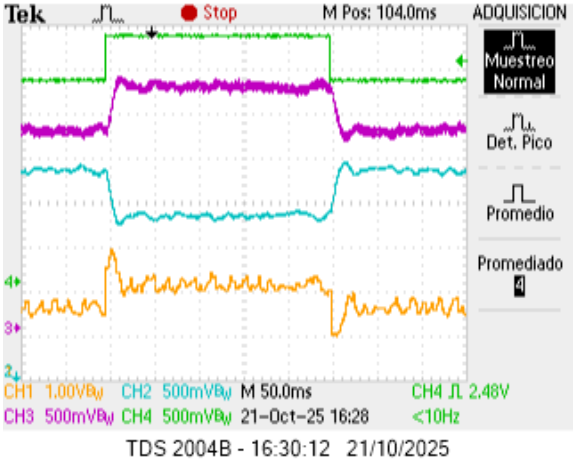


Figura 12. Respuesta experimental correspondiente al Caso 1 ($f_s \approx 418{,}5$ Hz).

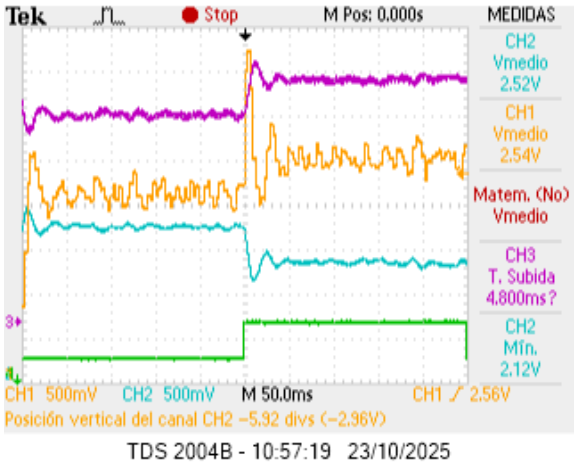


Figura 13. Respuesta experimental correspondiente al Caso 2 ($f_s \approx 313{,}9$ Hz).

reducido del hardware.

Es importante notar que las simulaciones no consideran efectos de muestreo imperfecto, ruido en las mediciones ni perturbaciones eléctricas presentes en la práctica. Dado que las señales medidas tienen amplitudes muy pequeñas, incluso un nivel bajo de ruido produce diferencias visibles, lo cual explica parte de las discrepancias observadas entre el modelo y el comportamiento real. En conjunto, los resultados confirman que los controladores *implementables* reproducen la dinámica cualitativa diseñada, con un nivel de error razonable para el contexto experimental y dentro de los márgenes esperados para este tipo de implementación física.

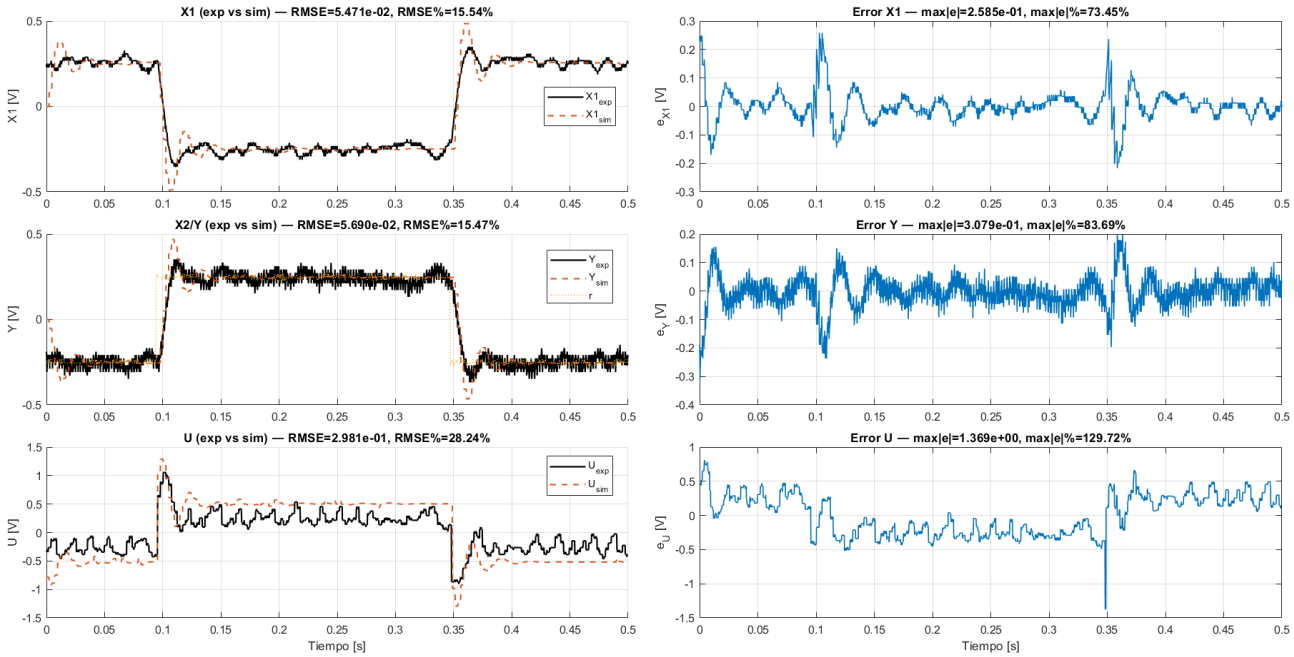


Figura 14. Comparación simulación vs. experimento — Caso 1. Métricas detalladas (RMSE y errores máximos) indicadas en cada panel.

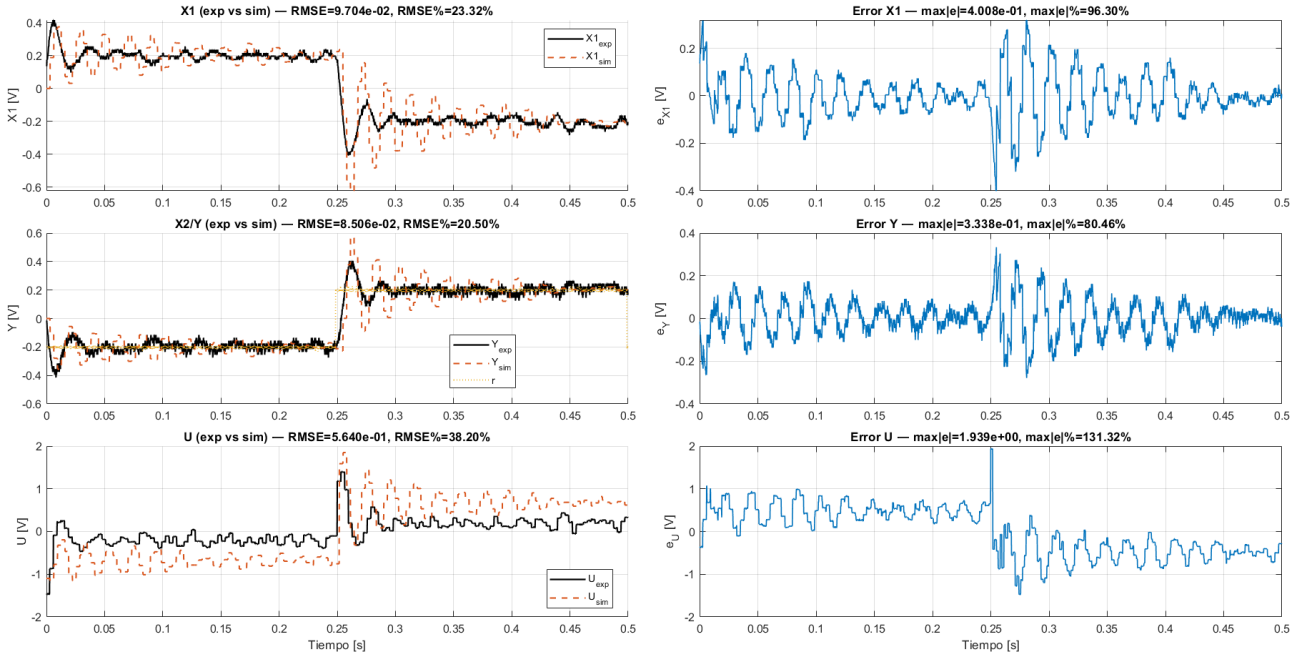


Figura 15. Comparación simulación vs. experimento — Caso 2. Métricas detalladas (RMSE y errores máximos) indicadas en cada panel.

APÉNDICE

A. *Código en C*

```c
#include "project.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include "mat.h"       // <-- tu librería de matrices

/* ==== Prototipos ==== */
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        (4.048f)
#define MIDDLE_VOLTAGE ((SAT_MAX - SAT_MIN)/2.0f)
#define VDDA2          (2.5f)
#define RX_BUF_SZ 64
#define A 0.4f
/* ==== Estado global ==== */
volatile char  flag    = '0';
volatile double ref_max = (float)(A/2);//(MIDDLE_VOLTAGE + 0.1f);
volatile double ref_min = (float)(-1*A/2);//(MIDDLE_VOLTAGE - 0.1f);
volatile double ref     = 0.0f;
volatile Mat X;
volatile double 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+VDDA2));

```

```
    /* Leer ADC y calcular error */
    float x1 = VADC_X1_CountsTo_Volts(VADC_X1_GetResult16());
    float x2 = VADC_X2_CountsTo_Volts(VADC_X2_GetResult16());
    X.d[0][0] = x1;
    X.d[1][0] = x2;
    //e = ref - muestra;

    flag = '1';

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


/* Hook del esfuerzo: respeta el modo (open/closed) */
static inline void actualizarEsfuerzo(void)
{
    const Mat K = {.filas = 1, .columnas = 2,
             .d = {{4.0979, 6.4912}}};//.d = {{2.6952,6.5662}}};//
    Mat X_copy  = {.filas = 2, .columnas = 1,
                    .d = {{0.0},
                          {0.0}}};
    Mat U_unsat = {.filas = 1, .columnas = 1,
             .d = {{0.0}}};

    Mat K0 = {.filas = 1, .columnas = 1,
            .d = {{3.3618}}};//.d = {{4.8505}}};//
    Mat R = {.filas = 1, .columnas = 1,
             .d = {{0.0}}};
    Mat KX= {.filas = 1, .columnas = 1,
             .d = {{0.0}}};
    Mat K0R = {.filas = 1, .columnas = 1,
             .d = {{0.0}}};;
    R.d[0][0] = ref; //Calculamos todo en AC
    if (mode == MODE_OPEN) {
        /* === LAZO ABIERTO: el DAC sigue la referencia (ref_min / ref_max) === */
        VDAC8_SetValue(volt_to_dac(ref));
        return;
    }


    mat_copy_volatile(&X_copy,&X);
    mat_mul(&K,&X_copy,&KX);
    mat_mul(&K0,&R,&K0R);
    mat_sut(&K0R,&KX,&U_unsat);
    U_unsat.d[0][0]+=VDDA2;
    float u_sat;
    if (U_unsat.d[0][0] > SAT_MAX)      u_sat = SAT_MAX;
    else if (U_unsat.d[0][0] < SAT_MIN) u_sat = SAT_MIN;
    else                          u_sat = U_unsat.d[0][0];
    VDAC8_SetValue(volt_to_dac(u_sat));
//    /* === LAZO CERRADO: controlador PID discreto === */
//    static float u_1 = 0.0f, u_2 = 0.0f, u_3 = 0.0f;
//    static float e_1 = 0.0f, e_2 = 0.0f, e_3 = 0.0f;
//
//    /* Salida no saturada */
//    float u_unsat = (float)(-U1*u_1 -U2*u_2 -U3*u_3+ E0*e + E1*e_1 + E2*e_2+E3*e_3)/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_3 = u_2; u_2 = u_1; u_1 = u_sat;
//    e_3 = e_2; 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])))
        s[--n] = '\0';
}

static void to_lower_str(char *s)
{
    for (; *s; ++s) *s = (char)tolower((unsigned char)*s);
}

/* Procesa bytes RX por líneas (eco por línea) */
static void pc_process_rx(void)
{
    int ch;
    while ((ch = PC_GetChar()) != 0) {
        if (ch == '\n' || ch == '\r') {
            if (rx_len > 0) {
                rx_buf[rx_len] = '\0';
                PC_PutString(rx_buf); PC_PutString("\r\n"); /* eco de línea */
                handle_command(rx_buf);
                rx_len = 0;
            }
        } else {
            if (rx_len < (RX_BUF_SZ-1)) {
                rx_buf[rx_len++] = (char)ch;
            } else {
                rx_len = 0;
                PC_PutString("ERR: line too long\r\n");
            }
        }
    }
}

/* ==== Status command (sin floats en printf; usa mV) ==== */
static void ref_status(void)
{
    long ref_min_mV = my_lroundf(ref_min * 1000.0f);
    long ref_max_mV = my_lroundf(ref_max * 1000.0f);
    long ref_mV     = my_lroundf(ref * 1000.0f);
    unsigned per    = (unsigned)timer_ref_ReadPeriod();

    char msg[160];
    snprintf(msg, sizeof(msg),
             "STATUS:\r\n"
             "  mode=%s\r\n"
             "  ref_min=%ld mV\r\n"
             "  ref_max=%ld mV\r\n"
             "  ref=%ld mV\r\n"
             "  ref_period=%u\r\n",
             (mode == MODE_OPEN) ? "open" : "closed",
             ref_min_mV, ref_max_mV, ref_mV, per);
    PC_PutString(msg);
}

/* ==== Parser de comandos ==== */
static void handle_command(const char *line_in)
{
```

```
    char line[RX_BUF_SZ];
    strncpy(line, line_in, sizeof(line));
    line[sizeof(line)-1] = '\0';
    rstrip(line);
    to_lower_str(line);

    /* Comandos sin valor */
    if (strcmp(line, "ref_status") == 0 || strcmp(line, "status") == 0) {
        ref_status(); return;
    }
    if (strcmp(line, "help") == 0 || strcmp(line, "?") == 0) {
        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);
    timer_ref_WriteCounter(period);
    timer_ref_Start();

    char msg[64];
    snprintf(msg, sizeof(msg), "OK: ref_period=%u\r\n", (unsigned)period);
    PC_PutString(msg);
}

/* ==== main ==== */
int main(void)
{
    flag = '0';

    CyGlobalIntEnable;

    mat_zero_volatile(&X,2,1);
    /* HW init */
    Opa_dac_Start();
    Opa_stage1_Start();
    Opa_vdda_2_Start();
    Opa_stage2_Start();
    VADC_X1_Start();
    VADC_X2_Start();
    VDAC8_Start();
    VDAC8_ref_Start();

    /* UART PC */
    PC_Start();
    PC_PutString("\r\nReady. Commands:\r\n");
    PC_PutString("  ref_max:<float 0..4.08>\r\n");
    PC_PutString("  ref_min:<float 0..4.08>\r\n");
    PC_PutString("  ref_period:<uint16 0..65535>\r\n");
    PC_PutString("  mode:<open|closed|1|0>\r\n");
    PC_PutString("  ref_status\r\n");

    /* Timer / ISR de muestra */
    muestra_disponible_StartEx(adquirirMuestra);
    muestra_disponible_Enable();
    timer_ref_Start();


    for (;;)
    {
        /* Procesar UART sin bloquear */
        pc_process_rx();

        /* Procesar muestra si disponible */
        if (flag == '1') {
            flag = '0';
            actualizarEsfuerzo();           /* respeta el modo */
            muestra_disponible_Enable();    /* re-habilitar ISR */
        }
    }
}
```

Listing 1. `main.c`: bucle principal, adquisición y lógica de control.

```c
#include "mat.h"


void mat_zero_volatile(volatile Mat *M, int r, int c) {
    M->filas = r; M->columnas = c;
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            M->d[i][j] = 0.0;
}


// ===== Implementaciones =====
void mat_zero(Mat *M, int r, int c) {
    M->filas = r; M->columnas = c;
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            M->d[i][j] = 0.0;
}

void mat_copy(Mat *dst, const Mat *src) {
    dst->filas = src->filas; dst->columnas = src->columnas;
    for (int i = 0; i < src->filas; i++)
        for (int j = 0; j < src->columnas; j++)
            dst->d[i][j] = src->d[i][j];
}


void mat_copy_volatile(Mat *dst, volatile Mat *src) {
    dst->filas = src->filas; dst->columnas = src->columnas;
    for (int i = 0; i < src->filas; i++)
        for (int j = 0; j < src->columnas; j++)
            dst->d[i][j] = src->d[i][j];
}

void mat_add(const Mat *A, const Mat *B, Mat *C) {
    C->filas = A->filas; C->columnas = A->columnas;
    for (int i = 0; i < A->filas; i++)
        for (int j = 0; j < A->columnas; j++)
            C->d[i][j] = A->d[i][j] + B->d[i][j];
}

void mat_sut(const Mat *A, const Mat *B, Mat *C) {
    C->filas = A->filas; C->columnas = A->columnas;
    for (int i = 0; i < A->filas; i++)
        for (int j = 0; j < A->columnas; j++)
            C->d[i][j] = A->d[i][j] - B->d[i][j];
}

void mat_mul(const Mat *A, const Mat *B, Mat *C) {
    C->filas = A->filas; C->columnas = B->columnas;
    for (int i = 0; i < A->filas; i++) {
        for (int j = 0; j < B->columnas; j++) {
            double acc = 0.0;
            for (int k = 0; k < A->columnas; k++)
                acc += A->d[i][k] * B->d[k][j];
            C->d[i][j] = acc;
        }
    }
}

void ss_step(const Mat *A, const Mat *B, const Mat *C, const Mat *D,
             const Mat *xk, const Mat *uk, Mat *xk1, Mat *yk)
{
```

```
    Mat Ax, Bu, Cx, Du;
    mat_mul(A, xk, &Ax);
    mat_mul(B, uk, &Bu);
    mat_add(&Ax, &Bu, xk1);

    mat_mul(C, xk, &Cx);
    mat_mul(D, uk, &Du);
    mat_add(&Cx, &Du, yk);
}
```

Listing 2. `mat.c`: operaciones matriciales usadas por el controlador digital.

```
#ifndef MAT_H
#define MAT_H

#define MAX 10

// ===== Estructura de Matriz =====
typedef struct {
    int filas;
    int columnas;
    double d[MAX][MAX];
} Mat;

// ===== Prototipos =====
void mat_zero_volatile(volatile Mat *M, int r, int c);
void mat_zero(Mat *M, int r, int c);
void mat_copy(Mat *dst, const Mat *src);
void mat_copy_volatile(Mat *dst, volatile Mat *src);
void mat_add(const Mat *A, const Mat *B, Mat *C);
void mat_sut(const Mat *A, const Mat *B, Mat *C);
void mat_mul(const Mat *A, const Mat *B, Mat *C);
void ss_step(const Mat *A, const Mat *B, const Mat *C, const Mat *D,
             const Mat *xk, const Mat *uk, Mat *xk1, Mat *yk);

#endif // MAT_H
```

Listing 3. `mat.h`: estructuras y prototipos de funciones.

B. Código en Matlab

```
close all
clear all
clc;
%% modelo
% load('../modelo.mat');Gc = G;
% [F,G,H,J] = tf2ss(Gc.Numerator,Gc.Denominator); sysC = ss(F,G,H,J);
% fn = 2*abs(min(zpk(Gc).P{1})/pi);
% Tn = 1/fn;

% === Parametros fisicos ===
% C1 = 103.07e-9;  C2 = 211.1e-9;
% R1 = 14.878e3;    R2 = 14.760e3;
% R3 = 80.55e3;    R4 = 81.09e3;
C2 = 103.07e-9;  C1 = 211.1e-9;
R3 = 14.878e3;    R4 = 14.760e3;
R1 = 80.55e3;    R2 = 81.09e3;


tau1 = R2*C1;            k1 = -R2/R1;
tau2 = R4*C2;            k2 = -R4/R3;
%tau2 = 1/376.4;
%tau1 = 1/114.2;
F = [ -1/tau1,     0;
       k2/tau2, -1/tau2 ];
G = [ k1/tau1; 0 ];
H = [0 1];      % <- justo lo que querías
```

```
J = 0;

sysC = ss(F,G,H,J);      % modelo "práctico" coherente
Gc   = tf(sysC);         % si querés compararlo con el empírico

zpk(Gc)



fn = max(abs(zpk(Gc).P{1}))/pi;
Tn = 1/fn;


 %% Parametros
Ts = [Tn/2,Tn/1.5];  %[1e-3,1e-3];%
wn_obj = 4*abs(max(zpk(Gc).P{1}));
zita_obj = 0.7;
poloObj = @(zita,wn,Ts) exp(Ts*wn*(-zita+1j*sqrt(1-zita^2)));
p1 =[poloObj(zita_obj,wn_obj,Ts(1));conj(poloObj(zita_obj,wn_obj,Ts(1)))];
p2 = [0.5+1j*0.5;0.5-1j*0.5];
polos = cell(2);
polos = {p1,p2};
 %% Calculo


 %step(sysC)
 %save('../modeloSS.mat','sysC');
sysD = cell(2);
sysD = {c2d(sysC,Ts(1)),c2d(sysC,Ts(2))};
K = cell(length(polos));
K0 = cell(length(polos));
Kref = cell(length(polos));
sysCL = cell(length(polos));

addpath('..\');
for i = 1:length(polos)
    [A,B,C,D] =ssdata(sysD{i});
    K{i} = acker(A,B,polos{i});
    Ac = A - B*K{i};

    sysCL{i} = ss(Ac, B, C, D, sysD{i}.Ts);
    [~,~,K0{i}] = refi(Ac,B,C,K{i});
    Kref{i} = 1/dcgain(sysCL{i});
end



 %%
A = [0.4,0.3];
r = cell(2);
r = { A(1)/2.*[-ones(1,50), ones(1,50)], ...
      A(2)/2.*[-ones(1,100), ones(1,100)] };
VDDA2=2.5;
for i = 1:length(sysCL)
    sysi = sysCL{i};

    % === REFERENCIA de esta iteración, como columna ===
    ri = r{i}(:);

    % === Simulación ===
    [X, Y, U] = ss_sym_digital(sysi, ri, K{i}, K0{i}, [0;0],0);



    % === Tiempo consistente con la referencia usada ===
    N  = numel(ri);
    t  = (0:N-1)' * sysi.Ts;
```

```
    n  = size(X,2);

    % ----- Figura y paneles -----
    fig = figure('Name', sprintf('CL #%d -- señales + polos/ceros', i));
    left = uipanel(fig,'Position',[0.05 0.08 0.58 0.87]);
    rightAx = axes(fig,'Position',[0.68 0.10 0.28 0.82]);

    % ----- Columna izquierda -----
    tl = tiledlayout(left, n+2, 1, 'TileSpacing','compact','Padding','compact');

    % y & r
    ax = nexttile(tl); hold(ax,'on');
    stairs(ax, t, VDDA2+Y(:), 'LineWidth', 1.6);              % y como vector
    stairs(ax, t, VDDA2 + ri, '--', 'LineWidth', 1.2);  % r alineada a t
    grid(ax,'on'); grid(ax,'minor'); ylabel(ax,'y, r');
    legend(ax, {'y','r'}, 'Location','best');
    title(ax, sprintf('CL #%d -- Salida y referencia', i));

    % estados
    for k = 1:n
        axk = nexttile(tl);
        stairs(axk, t, VDDA2+X(:,k), 'LineWidth', 1.4);
        grid(axk,'on'); grid(axk,'minor'); ylabel(axk, sprintf('x_%d',k));
        if k==1, title(axk, 'Estados'); end
    end

    % u
    axu = nexttile(tl);
    stairs(axu, t, VDDA2+U(:), 'LineWidth', 1.3);
    grid(axu,'on'); grid(axu,'minor'); ylabel(axu,'u'); xlabel(axu,'Tiempo [s]');
    title(axu,'Esfuerzo de control');

    % ----- Columna derecha: polos y ceros -----
    axes(rightAx);  %#ok<LAXES>
    cla(rightAx);
    try
        pzplot(sysi); zgrid;
        title(sprintf('Polos y ceros (fs=%.4g Hz)', 1/sysi.Ts));
    catch
        [Z,P,~] = zpkdata(sysi,'v'); hold(rightAx,'on'); grid(rightAx,'on'); axis(rightAx,'
    equal');
        plot(real(P), imag(P), 'x', 'LineWidth', 1.6, 'Parent', rightAx);
        plot(real(Z), imag(Z), 'o', 'LineWidth', 1.6, 'Parent', rightAx);
        if sysi.Ts>0
            th = linspace(0,2*pi,400);
            plot(cos(th), sin(th), '--', 'Parent', rightAx);
            xlabel(rightAx,'Re(z)'); ylabel(rightAx,'Im(z)');
        else
            xline(rightAx,0,'--'); xlabel(rightAx,'Re(s)'); ylabel(rightAx,'Im(s)');
        end
        legend(rightAx,{'Polos','Ceros','Círc. unidad'},'Location','best');
        title(rightAx, sprintf('Polos/Ceros (manual) -- fs=%.4g Hz', 1/sysi.Ts));
    end
end


%% === COMPARACIÓN CON RESULTADOS EXPERIMENTALES ===
exp_files = {'./Exp1.csv', './Exp2.csv'};
Ts_exp = 199.99999495E-6;
VDDA2 = 2.5;                          % para coherencia con simulación
% Tabla vacía robusta
varNames = {'Caso','Ts_sim','Ts_exp','RMSE_y','RMSE_u','MeanErr_y','MeanErr_u'};
varTypes = repmat({'double'}, 1, numel(varNames));
comparacion = table('Size',[0 numel(varNames)], ...
    'VariableTypes',varTypes, 'VariableNames',varNames);

```

```
for i = 1:length(exp_files)
    fprintf('=== Caso %d ===\n', i);

    % --- Cargar CSV (sin depender de nombres de columnas) ---
    % Si tus CSV tienen cabecera, readmatrix igual la ignora y traga los números.
    M = readmatrix(exp_files{i});

    % Columnas: E=5 (U), K=11 (X1), Q=17 (X2=Y), W=23 (ref)
    Uexp  = M(:,5);
    X1exp = M(:,11);          %#ok<NASGU> % lo cargas por si querés guardarlo
    X2exp = M(:,17);          % salida/estado 2
    Rexp  = M(:,23);

    % Limpiar NaN de final si los hay
    good  = all(~isnan([Uexp, X2exp, Rexp]), 2);
    Uexp  = Uexp(good);
    X1exp = X1exp(good);
    X2exp = X2exp(good);
    Rexp  = Rexp(good);
    Uexp = Uexp - mean(Uexp, 'omitnan');
    X1exp = X1exp- mean(X1exp, 'omitnan');
    X2exp = X2exp- mean(X2exp, 'omitnan');
    Rexp  = Rexp- mean(Rexp, 'omitnan');
         % --- Tiempo experimental (grilla original del osciloscopio) ---
    Nexp  = numel(Uexp);
    t_exp = (0:Nexp-1)' * Ts_exp;

    % --- Grilla de simulación (impuesta por el modelo) ---
    sysi   = sysCL{i};
    Tfin   = t_exp(end);
    Ns_sim = floor(Tfin / sysi.Ts) + 1;
    t_sim  = (0:Ns_sim-1)' * sysi.Ts;

        % --- Referencia para el modelo en grilla t_sim ---
    % Ya centrada: arriba hiciste Rexp = Rexp - mean(Rexp)
    r0    = Rexp;  % ya sin DC
    r_sim = interp1(t_exp(:), r0(:), t_sim(:), 'previous', 'extrap');

    % --- Simulación con la MISMA referencia (a Ts = sysi.Ts) ---
    [Xsim, Ysim, Usim] = ss_sym_digital(sysi, r_sim, K{i}, K0{i}, [0;0], 0);

    % === Normalizar formas y longitudes ===
    t_sim = t_sim(:);
    Xsim  = Xsim(:,:);          % Nx2
    Ysim  = Ysim(:);            % Nx1
    Usim  = Usim(:);            % Nx1
    Rsim  = r_sim(:);           % <- DEFINIR Rsim AQUÍ (misma longitud que t_sim)

    % Asegurar longitudes consistentes (recorte al mínimo común)
    Lsim  = min([numel(t_sim), size(Xsim,1), numel(Ysim), numel(Usim), numel(Rsim)]);
    t_sim = t_sim(1:Lsim);
    Xsim  = Xsim(1:Lsim,:);
    Ysim  = Ysim(1:Lsim);
    Usim  = Usim(1:Lsim);
    Rsim  = Rsim(1:Lsim);

    % --- Re-muestrear SIMULADAS a la grilla experimental t_exp ---
    % ZOH: 'previous' para no inventar energía
    X1sim_exp = interp1(t_sim, Xsim(:,1), t_exp, 'previous', 'extrap');
    Ysim_exp  = interp1(t_sim, Ysim,      t_exp, 'previous', 'extrap');
    Usim_exp  = interp1(t_sim, Usim,      t_exp, 'previous', 'extrap');
    Rsim_exp  = interp1(t_sim, Rsim,      t_exp, 'previous', 'extrap');


    % --- Alinear longitudes por seguridad (todo sobre t_exp) ---
    L = min([length(t_exp), length(X1exp), length(X2exp), length(Uexp), ...
```

```
            length(X1sim_exp), length(Ysim_exp), length(Usim_exp), length(Rsim_exp), length(Rexp)]);
    t_cmp     = t_exp(1:L);
    X1exp_cmp = X1exp(1:L);   Yexp_cmp = X2exp(1:L);   Uexp_cmp = Uexp(1:L);   Rexp_cmp = Rexp(1:L);
    X1sim_cmp = X1sim_exp(1:L); Ysim_cmp = Ysim_exp(1:L); Usim_cmp = Usim_exp(1:L); Rsim_cmp = Rsim_exp(1:L);

    % ------------ ERRORES (vectores) ------------
    % Absolutos
    err_x1_abs = X1exp_cmp - X1sim_cmp;
    err_y_abs  = Yexp_cmp  - Ysim_cmp;
    err_u_abs  = Uexp_cmp  - Usim_cmp;

    % Porcentuales (normalizamos por max|exp| para no explotar cerca de 0)
    den_x1 = max(abs(X1exp_cmp)); if den_x1 < 1e-9, den_x1 = 1e-9; end
    den_y  = max(abs(Yexp_cmp));  if den_y  < 1e-9, den_y  = 1e-9; end
    den_u  = max(abs(Uexp_cmp));  if den_u  < 1e-9, den_u  = 1e-9; end

    err_x1_pct = 100 * err_x1_abs / den_x1;
    err_y_pct  = 100 * err_y_abs  / den_y;
    err_u_pct  = 100 * err_u_abs  / den_u;

    % Máximos
    err_x1_abs_max = max(abs(err_x1_abs));
    err_y_abs_max  = max(abs(err_y_abs));
    err_u_abs_max  = max(abs(err_u_abs));

    err_x1_pct_max = max(abs(err_x1_pct));
    err_y_pct_max  = max(abs(err_y_pct));
    err_u_pct_max  = max(abs(err_u_pct));

        % ===== RMSE (abs y %) =====
    RMSE_x1_abs = sqrt(mean(err_x1_abs.^2));
    RMSE_y_abs  = sqrt(mean(err_y_abs.^2));
    RMSE_u_abs  = sqrt(mean(err_u_abs.^2));

    RMSE_x1_pct = sqrt(mean(err_x1_pct.^2));
    RMSE_y_pct  = sqrt(mean(err_y_pct.^2));
    RMSE_u_pct  = sqrt(mean(err_u_pct.^2));

    % ===== Tabla 3x4 (filas: X1,X2,U | cols: RMSE_abs, RMSE_pct, ErrMax_abs, ErrMax_pct) =====
    MetricsNames = {'RMSE_abs','RMSE_pct','ErrMax_abs','ErrMax_pct'};
    ResultCase = table( ...
        [RMSE_x1_abs; RMSE_y_abs; RMSE_u_abs], ...
        [RMSE_x1_pct; RMSE_y_pct; RMSE_u_pct], ...
        [err_x1_abs_max; err_y_abs_max; err_u_abs_max], ...
        [err_x1_pct_max; err_y_pct_max; err_u_pct_max], ...
        'VariableNames', MetricsNames, ...
        'RowNames', {'X1','X2','U'} );

    % Guardamos en struct por si querés usar luego:
    resultados_por_caso(i).tabla = ResultCase;  %#ok<SAGROW>
    resultados_por_caso(i).t    = t_cmp;
    resultados_por_caso(i).X1exp_cmp = X1exp_cmp; resultados_por_caso(i).X1sim_cmp = X1sim_cmp;
    resultados_por_caso(i).Yexp_cmp  = Yexp_cmp;  resultados_por_caso(i).Ysim_cmp  = Ysim_cmp;

    resultados_por_caso(i).Uexp_cmp  = Uexp_cmp;  resultados_por_caso(i).Usim_cmp  = Usim_cmp;

    resultados_por_caso(i).Rexp_cmp  = Rexp_cmp;  resultados_por_caso(i).Rsim_cmp  = Rsim_cmp;


    % (opcional) exportar la tabla del caso i a CSV con nombres de fila:
    writetable(ResultCase, sprintf('./Comparacion_Metricas_Caso%d.csv', i), 'WriteRowNames', true);
```

```
    disp(ResultCase);

        % ------------ FIGURA: 3 filas x 2 columnas (sobre t_cmp=t_exp recortado)
    ------------
    figure('Name', sprintf('Caso %d -- Señales y errores (grilla experimental)', i), 'Color',
     'w');
    tl = tiledlayout(3,2,'TileSpacing','compact','Padding','compact');

    % --- Fila 1: X1 ---
    nexttile; hold on;
    plot(t_cmp, X1exp_cmp, 'k', 'LineWidth', 1.3);
    plot(t_cmp, X1sim_cmp, '--', 'LineWidth', 1.2);
    legend('X1_{exp}','X1_{sim}','Location','best'); grid on; ylabel('X1 [V]');
    title(sprintf('X1 (exp vs sim) -- RMSE=%.3e, RMSE%%=%.2f%%', RMSE_x1_abs, RMSE_x1_pct));

    nexttile;
    plot(t_cmp, err_x1_abs, 'LineWidth', 1); grid on; ylabel('e_{X1} [V]');
    title(sprintf('Error X1 -- max|e|=%.3e, max|e|%%=%.2f%%', err_x1_abs_max, err_x1_pct_max
    ));

    % --- Fila 2: X2 = Y (con referencia) ---
    nexttile; hold on;
    plot(t_cmp, Yexp_cmp, 'k', 'LineWidth', 1.3);
    plot(t_cmp, Ysim_cmp, '--', 'LineWidth', 1.2);
    plot(t_cmp, Rexp_cmp, ':',  'LineWidth', 1.0);
    legend('Y_{exp}','Y_{sim}','r','Location','best'); grid on; ylabel('Y [V]');
    title(sprintf('X2/Y (exp vs sim) -- RMSE=%.3e, RMSE%%=%.2f%%', RMSE_y_abs, RMSE_y_pct));

    nexttile;
    plot(t_cmp, err_y_abs, 'LineWidth', 1); grid on; ylabel('e_{Y} [V]');
    title(sprintf('Error Y -- max|e|=%.3e, max|e|%%=%.2f%%', err_y_abs_max, err_y_pct_max));

    % --- Fila 3: U ---
    nexttile; hold on;
    plot(t_cmp, Uexp_cmp, 'k', 'LineWidth', 1.3);
    plot(t_cmp, Usim_cmp, '--', 'LineWidth', 1.2);
    legend('U_{exp}','U_{sim}','Location','best'); grid on; ylabel('U [V]'); xlabel('Tiempo [
    s]');
    title(sprintf('U (exp vs sim) -- RMSE=%.3e, RMSE%%=%.2f%%', RMSE_u_abs, RMSE_u_pct));

    nexttile;
    plot(t_cmp, err_u_abs, 'LineWidth', 1); grid on; ylabel('e_{U} [V]'); xlabel('Tiempo [s]')
    ;
    title(sprintf('Error U -- max|e|=%.3e, max|e|%%=%.2f%%', err_u_abs_max, err_u_pct_max));

end

writetable(comparacion, './Comparacion_Resultados.csv');
```

Listing 4. `HojaDeCalculos.m`: cálculo de parámetros y generación de figuras.

```
%% ====================== Cargar datos medidos ======================
data = readmatrix('mediciones.csv');

% u = ENTRADA medida; y = SALIDA medida
y = data(:, 11);    % ajustá si cambia columna
u = data(:, 17);

Ts = 0.000099999997474;  % tiempo de muestreo (≈100 µs)
data_id = iddata(y, u, Ts);
sysC = ssest(data_id, 2);
sysC = ss(sysC);            % A,B,C,D numéricos; sin K ni NoiseVariance
step(sysC)
```

Listing 5. `modeladoSS.m`: modelado en espacio de estados y discretización.

```
close all
```

```
clear all
% Verificar que las FT parciales salen exactas:
s = tf('s');
% === Parametros fisicos ===
C1 = 103.07e-9;  C2 = 211.1e-9;
R1 = 14.878e3;    R2 = 14.760e3;
R3 = 80.55e3;    R4 = 81.09e3;
tau1 = R2*C1;           k1 = -R2/R1;
tau2 = R4*C2;           k2 = -R4/R3;
F = [ -1/tau1,     0;
       k2/tau2, -1/tau2 ];
G = [ k1/tau1; 0 ];
H = [0 1];      % <- justo lo que querías
J = 0;

X1_R = (-R2/R1) / (tau1*s + 1);
X2_X1 = (-R4/R3) / (tau2*s + 1);

% Desde el ss:
[num1,den1] = tfdata(ss(F,[k1/tau1;0],[1 0],0),'v');   % X1/R
X1_R_ss = tf(num1,den1);

[num2,den2] = tfdata(ss([ -1/tau2 ], [k2/tau2], 1, 0),'v'); % X2/X1
X2_X1_ss = tf(num2,den2);

% Comparar (deberían ser iguales o numéricamente muy cercanos):
bode(X1_R, X1_R_ss); legend('X1/R (esperado)','X1/R (ss)');
figure; bode(X2_X1, X2_X1_ss); legend('X2/X1 (esperado)','X2/X1 (ss)');




X2_R   = X2_X1 * X1_R;          % cascada completa (verificación)

% Espacio de estados equivalente (x = [x1; x2], y = x2)
F = [ -1/tau1,    0;
       k2/tau2,  -1/tau2 ];
G = [ k1/tau1; 0 ];
H = [0 1];
J = 0;
sysC = ss(F,G,H,J);

%% === Excitación: escalón unitario en R ===
tf_end = 6*max(tau1,tau2);      % horizonte suficiente
t  = linspace(0, tf_end, 1500).';
r  = ones(size(t));             % escalón unitario

%% === Respuesta por espacio de estados (x1(t), x2(t)=y(t)) ===
[y_ss, t_ss, x_ss] = lsim(sysC, r, t);   % x_ss(:,1)=x1, x_ss(:,2)=x2

%% === Respuesta por T.F. en cascada como pediste ===
% 1) X1(t) respecto a R(t)
[x1_tf, t1] = step(X1_R, t);     % como R es escalón unitario, step sirve
% 2) Alimentar X1(t) como entrada de la segunda etapa para obtener X2(t)
x2_tf = lsim(X2_X1, x1_tf, t1);

% (opcional) chequeo directo de X2/R:
[y_tfdir, ~] = step(X2_R, t);    % debería coincidir con x2_tf

%% === Gráficas comparativas ===
figure('Name','Estados vs TF en cascada','Color','w');

% x1
subplot(3,1,1); hold on; grid on;
plot(t_ss, x_ss(:,1),'LineWidth',1.6);
plot(t1,   x1_tf,'--','LineWidth',1.4);
ylabel('x_1(t)');
```

```
title('x_1: Estado vs. TF X1/R');
legend('x_1 (estado-espacio)','x_1 (TF X1/R)','Location','best');

% x2
subplot(3,1,2); hold on; grid on;
plot(t_ss, x_ss(:,2),'LineWidth',1.6);
plot(t1,   x2_tf,'--','LineWidth',1.4);
ylabel('x_2(t)');
title('x_2: Estado vs. TF en cascada (X1/R → X2/X1)');
legend('x_2 (estado-espacio)','x_2 (TF cascada)','Location','best');

% y = x2, y verificación X2/R directo
subplot(3,1,3); hold on; grid on;
plot(t_ss, y_ss,'LineWidth',1.6);
plot(t,    y_tfdir,'--','LineWidth',1.4);
xlabel('Tiempo [s]'); ylabel('y(t)=x_2(t)');
title('Salida: Estado vs. TF directa (X2/R)');
legend('y (estado-espacio)','y (TF X2/R directa)','Location','best');

%% === Sanity checks cortos en consola ===
fprintf('Ganancia DC esperada X1/R = k1 = %.4f\n', k1);
fprintf('Ganancia DC esperada X2/X1 = k2 = %.4f\n', k2);
fprintf('Ganancia DC esperada X2/R = k1*k2 = %.4f\n', k1*k2);
```

Listing 6. `sanityCheck.m`: chequeos de consistencia.

```
function [X, Y, U] = ss_sym_digital(sys, ref, K, K0, x0,sigma)
% Discreto, simple:
%   u(k) = K0*ref(k) - K*x(k)
%   x+   = A*x + B*u
%   y(k) = C*x + D*u   (convención típica)
    [A,B,C,D] = ssdata(sys);
    n = size(A,1);
    N = numel(ref);

    if nargin < 5 || isempty(x0), x0 = zeros(n,1); end

    X = zeros(n,N);
    U = zeros(1,N);
    Y = zeros(1,N);
    ruido = sigma*randn(N);
    X(:,1) = x0;
    for k = 1:N-1
        U(k)     = K0*ref(k) - K*X(:,k)+ruido(k);
        X(:,k+1) = A*X(:,k) + B*U(k);
        Y(k)     = C*X(:,k) + D*U(k);
    end
    % último sample
    U(N) = K0*ref(N) - K*X(:,N);
    Y(N) = C*X(:,N) + D*U(N);

    % para plot cómodo (Nxn)
    X = X.'; Y = Y(:); U = U(:);
end
```

Listing 7. `ss_sym_digital.m`: simulación discreta con realimentación de estados.