

Práctica de Laboratorio 6: Utilización y evaluación de estimadores de estado de predicción y actual.

Elías Álvarez

Carrera de Ing. Electrónica

Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay

Email: elias.alvarez@universidadcatolica.edu.py

Docente: Lic. Montserrat González

Facultad de Ingeniería

Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay

Tania Romero

Carrera de Ing. Electrónica

Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay

Email: tania.romero@universidadcatolica.edu.py

Docente: PhD. Enrique Vargas

Facultad de Ingeniería

Universidad Católica Nuestra Señora de la Asunción
Asunción, Paraguay

I. INTRODUCCIÓN

II. OBJETIVOS

- Diseñar controladores digitales para una planta analógica utilizando estados estimados.
- Aplicar el diseño de realimentación de estados con estimadores de predicción y de actualización.
- Presentar y discutir los resultados experimentales en comparación con los obtenidos por simulación, dentro de un informe técnico razonado.

III. DESARROLLO

Considerando la planta analógica esquematizada en la Fig. 1, se procedió a diseñar compensadores digitales utilizando en forma conjunta la técnica de *ubicación arbitraria de polos* y el empleo de *estimadores de estado* en tiempo discreto, tal como lo exige la consigna de la Práctica 6. En esta sección se documentan las decisiones de diseño, los cálculos realizados en MATLAB (código completo en el Apéndice, ver Código ??) y la comparación entre las dos arquitecturas de estimador: el **estimador de predicción** y el **estimador actual**.

III-A. Selección del tiempo de muestreo

El primer paso fue fijar un tiempo de muestreo T_s que permita capturar la dinámica más rápida de la planta. Para ello se partió del modelo continuo obtenido a partir de los valores medidos de resistencias y capacitores:

$$\begin{aligned} C_1 &= 211,100 \times 10^{-9} \text{ F}, & C_2 &= 103,070 \times 10^{-9} \text{ F}, \\ R_1 &= 47,000 \times 10^3 \Omega, & R_2 &= 81,090 \times 10^3 \Omega, \\ R_3 &= 6,800 \times 10^3 \Omega, & R_4 &= 14,760 \times 10^3 \Omega. \end{aligned}$$

Con estos valores se definieron las constantes de tiempo y las ganancias

$$\tau_1 = R_2 C_1, \quad \tau_2 = R_4 C_2, \quad k_1 = -\frac{R_2}{R_1}, \quad k_2 = -\frac{R_4}{R_3},$$

y se armó el modelo continuo en espacio de estados:

$$F = \begin{bmatrix} -\frac{1}{\tau_1} & 0 \\ \frac{k_2}{\tau_2} & -\frac{1}{\tau_2} \end{bmatrix}, \quad G = \begin{bmatrix} \frac{k_1}{\tau_1} \\ 0 \end{bmatrix}, \quad H = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad J = 0,$$

que coincide con la planta analógica real.

El cálculo en MATLAB se hizo con el siguiente fragmento (el código completo está en el Apéndice, Código ??):

```
1 clear all; clc; close all;
2
3 % --- Planta continua (modelo medido) ---
4 C2 = 103.07e-9; C1 = 211.1e-9;
5 R3 = 6.8e3; R4 = 14.760e3;
6 R1 = 47e3; R2 = 81.09e3;
7
8 tau1 = R2*C1; k1 = -R2/R1;
9 tau2 = R4*C2; k2 = -R4/R3;
10
11 F = [ -1/tau1, 0;
12 k2/tau2, -1/tau2 ];
13 G = [ k1/tau1; 0 ];
14 H = [ 0 1 ]; J = 0;
15
16 sysC = ss(F,G,H,J);
17 Gc = tf(sysC);
18
19 % Ts base vía polo más rápido continuo
20 fn = max(abs(zpk(Gc).P{1})) / pi;
21 Tn = 1/fn;
22 Ts = Tn/4; % 4 muestras por la dinámica más rápida
23
24 % Discretización
25 sysD = c2d(sysC, Ts, 'zoh');
26 [A,B,C,D] = ssdata(sysD);
27
28 % Controlabilidad / observabilidad
29 Co = ctrb(A,B);
30 Ob = obsv(A,C);
31 rCo = rank(Co); rOb = rank(Ob);
```

Listing 1. Cálculo de T_s a partir del polo continuo más rápido.

La lógica fue:

1. obtener los polos continuos de la planta;
2. tomar el polo de mayor módulo (el más rápido);
3. definir el período asociado $T_n = 1/f_n$;
4. fijar el tiempo de muestreo como

$$T_s = \frac{T_n}{4},$$

es decir, cuatro muestras sobre la parte más rápida de la dinámica.

Con ese T_s se discretizó la planta y se verificó que

$$\text{rango}(\text{ctrb}(A, B)) = \text{rango}(\text{obsv}(A, C)) = 2,$$

por lo que el diseño por ubicación de polos y el diseño de observadores son viables en discreto.

III-B. Diseño del controlador por realimentación de estados

Con el modelo discreto (A, B, C, D, T_s) se diseñó primero el lazo de control. Se eligió la pareja compleja

$$p_{\text{ctrl}} = 0,8 \pm j 0,2$$

porque:

- queda más lenta que un diseño agresivo, por lo que el esfuerzo de control no se dispara;
- está dentro del círculo unidad y asegura un régimen aceptable;
- está en una zona intermedia respecto de los polos discretizados de la planta.

La ganancia de realimentación se obtuvo con Ackermann:

```
1 % Polos de control deseados
2 p_ctrl = [0.8 + 1j*0.2; 0.8 - 1j*0.2];
3
4 % Ganancia de realimentación de estados
5 K = acker(A, B, p_ctrl);
6
7 % Prefiltro para que y_ss = r (caso SISO
8 y D=0)
9 [~, ~, Nbar] = refi(A, B, C, K);
```

Listing 2. Cálculo de K y del prefiltro N_{bar} .

La ley de control que se implementó fue entonces

$$u[k] = N_{\text{bar}} r[k] - K \hat{x}[k],$$

donde $\hat{x}[k]$ es el *estado estimado*. Esto es así porque en esta planta sólo se mide la salida

$$y[k] = C x[k] = x_2[k],$$

y con esa única medición se debe reconstruir el vector completo de estado $x[k]$; por lo tanto, el estimador no es opcional.

III-C. Diseño de los dos estimadores de estado

A partir de la única medición de la salida $y[k]$ se debe llegar a reconstruir el vector de estado completo $x[k] = [x_1 \ x_2]^T$; por eso el estimador es una parte fundamental del diseño. Se implementaron las dos variantes que se piden en el laboratorio:

1. Estimador de predicción:

$$\hat{x}[k+1] = A \hat{x}[k] + B u[k] + L_{\text{pred}} (y[k] - C \hat{x}[k]).$$

2. Estimador actual (con la predicción $z[k+1]$ que usás en tu código):

$$z[k+1] = A \hat{x}[k] + B u[k],$$

$$\hat{x}[k+1] = z[k+1] + L_{\text{act}} (y[k+1] - C z[k+1]).$$

Para comparar el efecto de la dinámica del observador se usaron dos ubicaciones en el plano- z :

- caso 1 (moderado): $p_{\text{obs}} = 0,4 \pm j 0,4$;
- caso 2 (rápido): $p_{\text{obs}} = 0,2 \pm j 0,2$.

Las ganancias se obtuvieron por dualidad:

```
1 % Polos del observador (cambiar 0.4+0.4j
2 <-> 0.2+0.2j según el caso)
3 p_obs = [0.2+1j*0.2, 0.2-1j*0.2];
4
5 % Estimador de PREDICCIÓN: corrige con y
6 [k]
7 L_pred = acker(A', C', p_obs).';
8
9 % Estimador ACTUAL: corrige con y[k+1],
10 usa C*A
11 L_act = acker(A', (C*A)', p_obs).';
```

Listing 3. Ganancias de los dos estimadores.

La condición buscada fue que los polos del observador queden más cerca del origen que los polos del lazo cerrado $A - BK$, de modo que el error de estimación desaparezca en pocas muestras y la dinámica dominante siga siendo la de los polos que se ubicaron para la planta.

III-D. Simulación comparativa

Con K , N_{bar} , L_{pred} y L_{act} se corrieron dos simulaciones sobre el mismo modelo discreto (A, B, C, D, T_s) : una con `sim_predictor` y otra con `sim_current`. En ambas se usó:

- la misma referencia alternada;
- las mismas condiciones iniciales no nulas para $x[0]$ y $\hat{x}[0]$;
- y el mismo horizonte de tiempo.

```
1 simPred = sim_predictor(A, B, C, D, K,
2 L_pred, Nbar, Ts, Tsim, x0, xh0, r);
3 simPred.title = "Estimador de prediccion
4 ";
5
6 simCurr = sim_current(A, B, C, D, K, L_act,
7 Nbar, Ts, Tsim, x0, xh0, r);
8 simCurr.title = "Estimador actual";
```

Listing 4. Simulación de ambas arquitecturas.

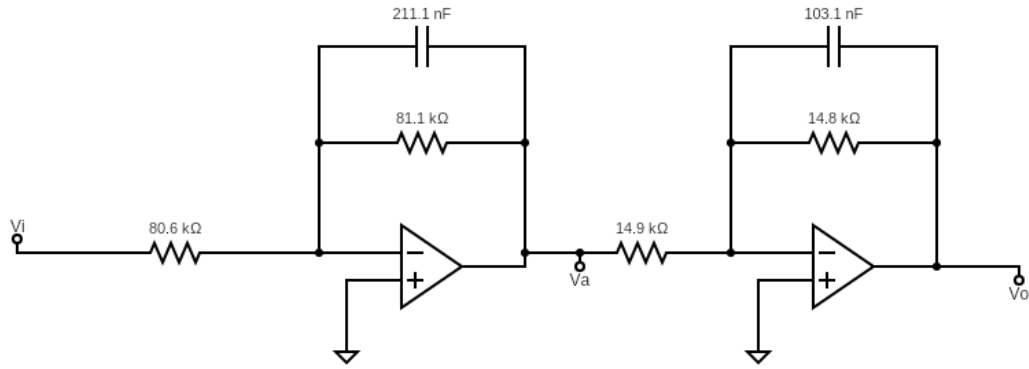


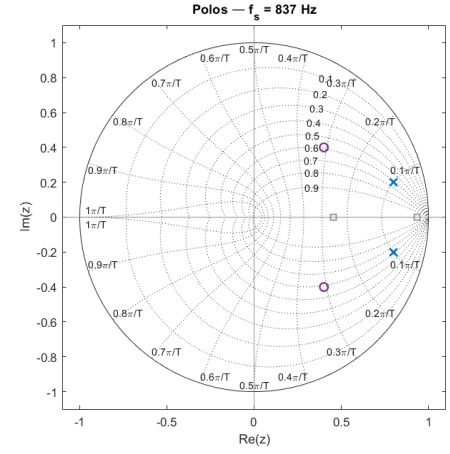
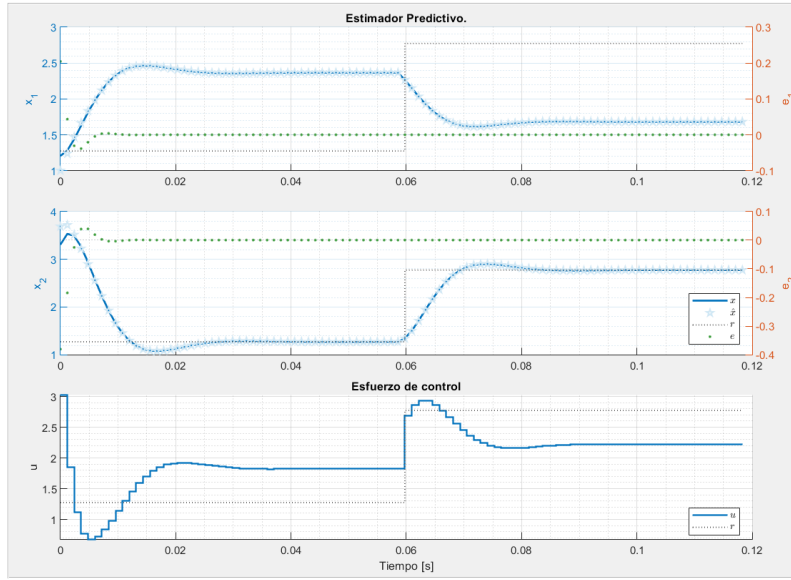
Figura 1. Planta analógica discretizada para el laboratorio.

III-E. Discusión de resultados

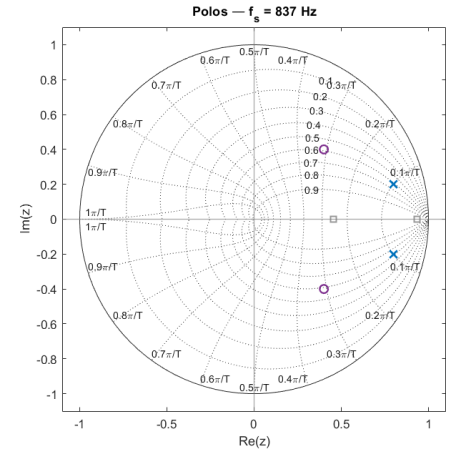
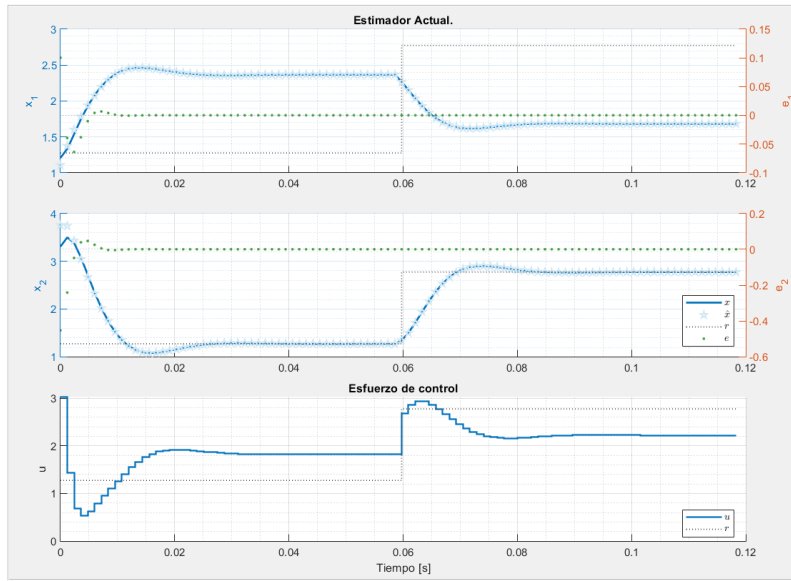
De las cuatro simulaciones se observa que:

1. Para una misma ubicación de polos del observador, el estimador actual converge en menos muestras que el estimador de predicción.
2. Al acelerar los polos del observador (de $0,4 \pm j0,4$ a $0,2 \pm j0,2$), ambos estimadores se vuelven muy rápidos y la diferencia entre sus trayectorias de estado se reduce.
3. Como la ley de control es la misma en todos los casos,

$$u[k] = N_{\text{bar}}r[k] - K\hat{x}[k],$$
 la salida en lazo cerrado termina siendo prácticamente igual una vez que el estimador alcanza al estado real.
4. La elección de polos del controlador en $0,8 \pm j0,2$ permitió que las diferencias que se ven en las figuras se deban al observador y no a una saturación del actuador.

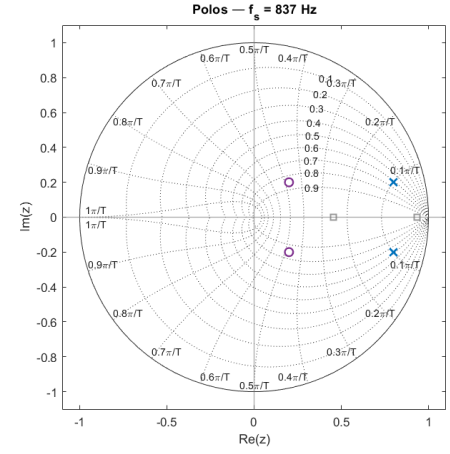
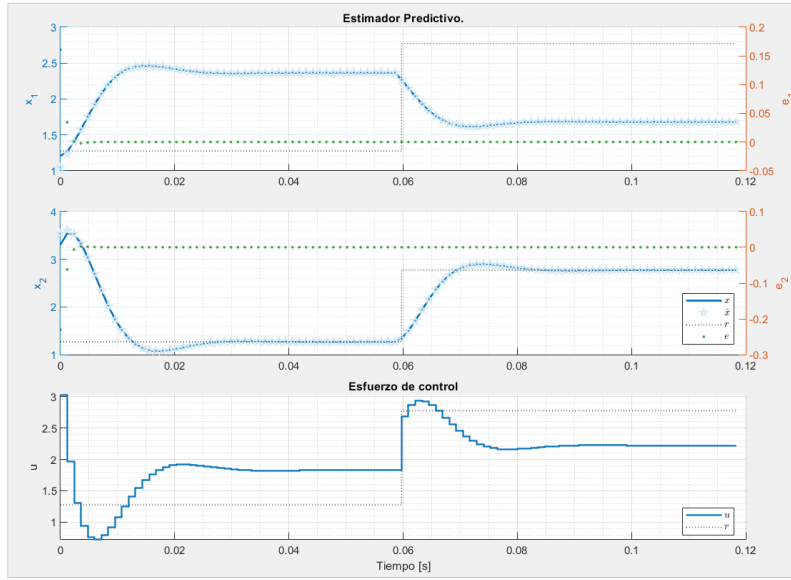


(a) Predicción, $p_{\text{obs}} = 0,4 \pm j0,4$

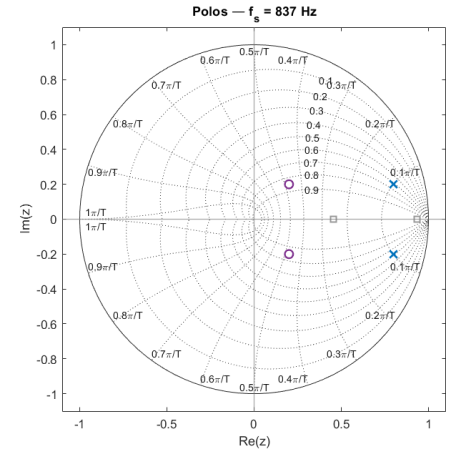
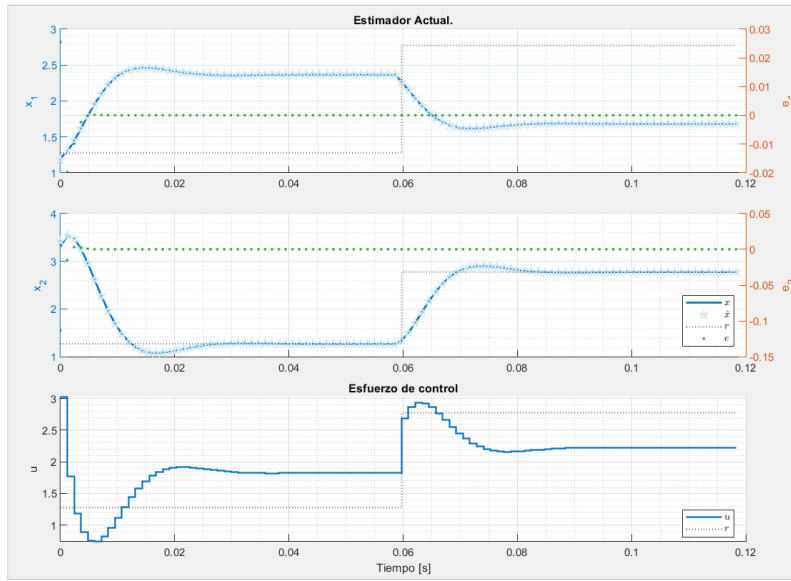


(b) Actual, $p_{\text{obs}} = 0,4 \pm j0,4$

Figura 2. Comparación de respuestas para estimador de predicción y actual con polos de observador moderados.

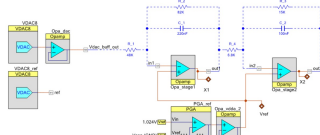


(a) Predicción, $p_{\text{obs}} = 0.2 \pm j0.2$

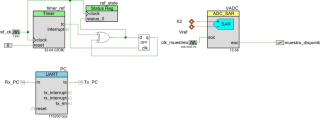


(b) Actual, $p_{\text{obs}} = 0.2 \pm j0.2$

Figura 3. Comparación de respuestas para estimador de predicción y actual con polos de observador rápidos.



(a) PSoC: planta y DAC (pág. 1).



(b) PSoC: ref, UART, ADC e ISR (pág. 2).

Figura 4. Implementación en PSoC del controlador con estimador de predicción.

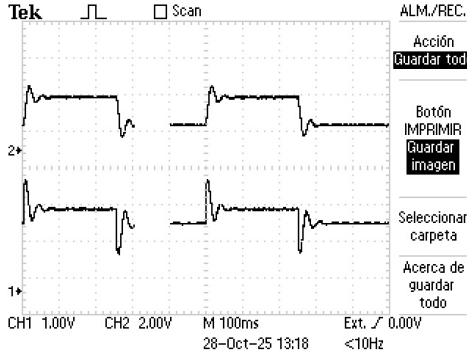


Figura 5. Respuesta experimental del esquema con estimador de predicción con errores.

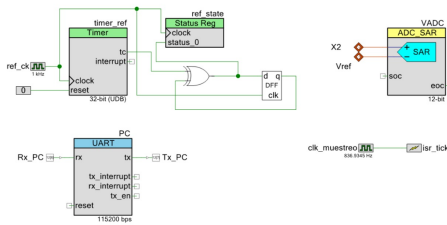


Figura 6. PSoC: esquema modificado para el estimador actual (pág. 2).

IV. IMPLEMENTACIÓN

En esta sección se documenta la implementación en el PSoC de los dos esquemas diseñados en MATLAB: el controlador con *estimador de predicción* y el controlador con *estimador actual*, ambos usando los polos del observador en

$$p_{obs} = 0,2 \pm j 0,2.$$

Los códigos fuente completos están en el Apéndice:

- controlador con estimador de predicción: Código ??;
- controlador con estimador actual: Código ??;
- versión reducida para depurar tiempos: Código ??.

IV-A. Implementación del estimador de predicción

La primera implementación que se probó en hardware fue la del estimador que corrige en el mismo instante k . El proyecto de PSoC quedó armado así:

- **Página 1:** planta + DAC para inyectar el esfuerzo, como en los labs anteriores.
- **Página 2:** toda la “circuitería de servicio”: generación/carga de la referencia por UART (amplitud y período del tren de pulsos), ADC para medir la salida y el pin de la ISR conectado al *End of Conversion* del ADC.

El código correspondiente es el Código ??. La lógica es la misma que en la simulación: 1) leer $y[k]$ del ADC, 2) calcular $u[k] = N_{bar}r[k] - K\hat{x}[k]$, 3) actualizar $\hat{x}[k+1] = A\hat{x}[k] + Bu[k] + L_{pred}(y[k] - C\hat{x}[k])$, 4) escribir $u[k]$ en el DAC.

En el prototipado funcionó, pero la forma de la respuesta no coincidía con la de la simulación: el esfuerzo tenía una forma diferente y la salida tenía sobreimpulso y un pequeño rizado. Eso se ve en la Fig. 5.

La causa fue el **retardo real** del lazo de interrupción: el cálculo del estimador + el cálculo del control estaban demasiado cerca del siguiente muestreo, así que en vez de aplicar el control en T_s se estaba aplicando “un poquito después”, y eso la simulación no lo contemplaba.

IV-B. Implementación del estimador actual

Como el predictor no quedó igual al modelo, se intentó el esquema con *estimador actual*. En este caso solo cambió la segunda página del PSoC (la de la lógica), porque la planta y el DAC quedaron igual.

El código usado es ??. A diferencia del predictor, la secuencia en la ISR es: 1) formar la predicción $z[k+1] = A\hat{x}[k] + Bu[k]$, 2) aplicar el esfuerzo al DAC, 3) medir la nueva salida, 4) corregir $\hat{x}[k+1] = z[k+1] + L_{act}(y[k+1] - Cz[k+1])$.

Esta versión, al principio, **no funcionó siquiera**: La salida oscilaba entre los valores extremos del DAC. Eso nos llevó a revisar tiempos de ejecución con el depurador.

IV-C. Retardo medido y reloj

Las pausas en la ISR y los mensajes por serie mostraron que el tiempo desde el *End of Conversion* hasta que el DAC actualizaba era comparable con T_s . En esas condiciones, la suposición del diseño (“el control se aplica en el mismo instante de muestreo”) deja de ser válida.

Para reducir ese retardo se decidió **subir la frecuencia del master clock** del PSoC. Con más clock, el mismo código C ocupó menos porcentaje del período y las dos versiones (predictor y actual) empezaron a parecerse más al caso ideal.

Se ve que, con el clock más rápido, el sobreimpulso baja y el esfuerzo de control se acerca más al de la simulación. O sea: el problema principal no era “qué estimador usar” sino **cuánto tarda el micro en hacer las cuentas**.

Cuadro I
COMPARACIÓN EXPERIMENTAL Y SIMULADA DEL DESEMPEÑO.

Caso	t_s [ms]	OS [%]
Predictor (Sim)	7,000	4,600
Predictor (clock bajo)	3,580	<10,000
Predictor (clock alto)	6,520	18,600
Actual (Sim)	7,000	4,600
Actual (clock alto)	5,940	\approx 10,000

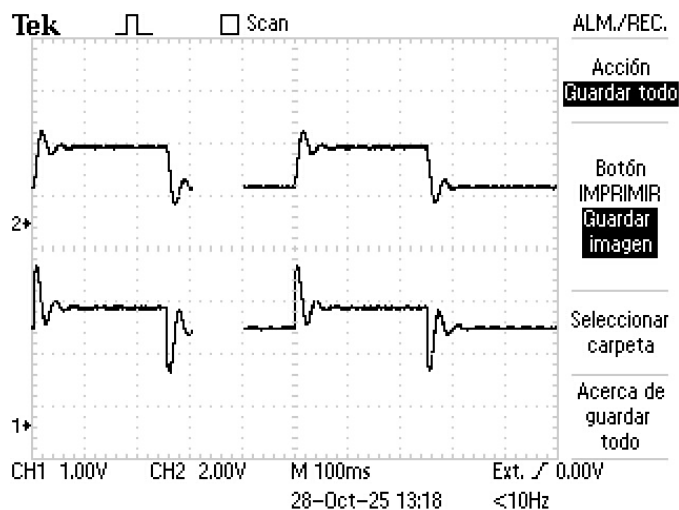
V. RESULTADOS

En las figuras anteriores se aprecia que la implementación con un tiempo de muestreo insuficiente difiere notablemente de la respuesta esperada, mientras que tanto el observador *predictivo* como el *actual* logran un desempeño similar al de la simulación, aunque con un sobreimpulso mayor al teórico. Las discrepancias se explican principalmente por las tolerancias de los componentes pasivos, pequeñas desviaciones en las

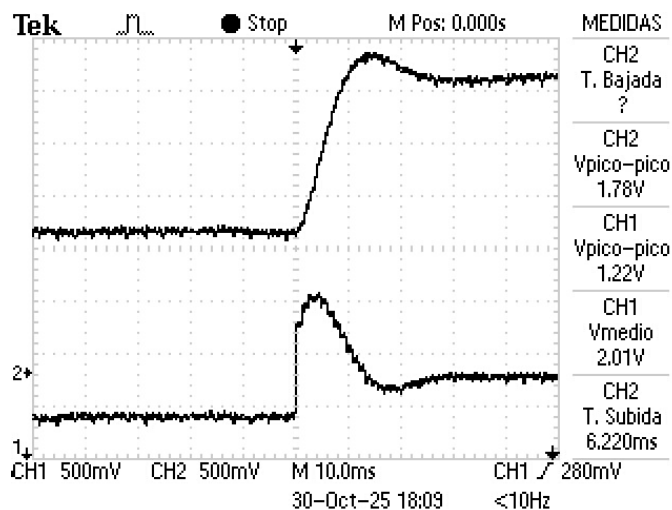
ganancias reales del sistema y efectos de cuantización de los conversores DAC/ADC.

La tabla I resume los valores medidos y simulados. Se observa que con el *clock* bajo la respuesta dista mucho de la esperada, evidenciando la importancia del período de que se cumplan los supuestos en el diseño. En cambio, con el *clock* alto, ambas variantes del observador mantienen una dinámica coherente con lo diseñado, aunque con un sobreimpulso mayor al simulado. El incremento del OS puede atribuirse a las tolerancias de resistencias y capacitores, a la dispersión de los polos reales respecto a los teóricos y a las no idealidades del hardware.

En conjunto, los resultados confirman que ambos esquemas de estimador son implementables y que **la elección adecuada del T_s y del reloj interno del PSoC es tan determinante como la ubicación de polos** para lograr un control estable y preciso.

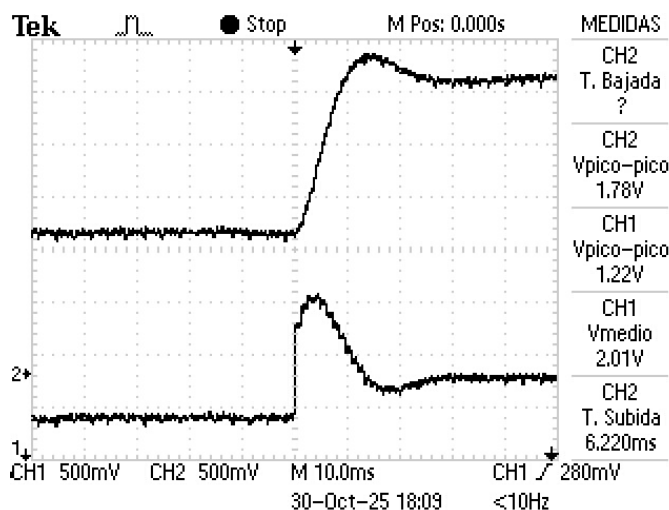


(a) Estimador de predicción (clock original).

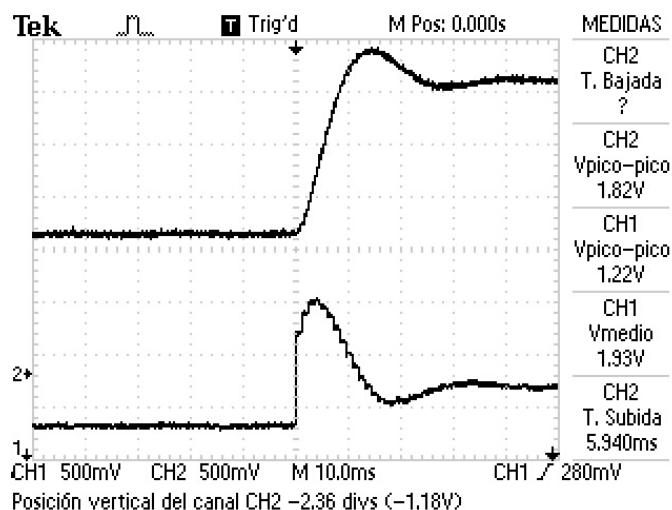


(b) Estimador predicción (clock aumentado).

Figura 8. Respuestas experimentales cambiando el clock.

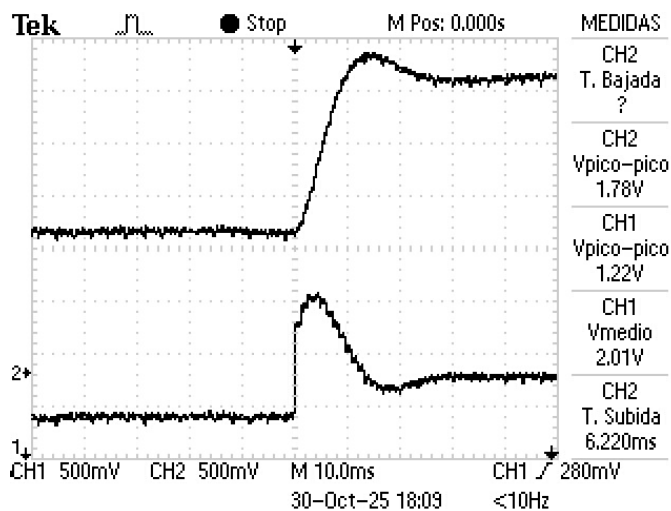


(a) Estimador de predicción (clock aumentado).

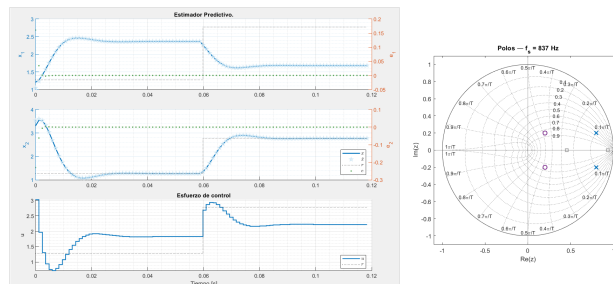


(b) Estimador actual (clock aumentado).

Figura 9. Respuestas experimentales con el clock aumentado.

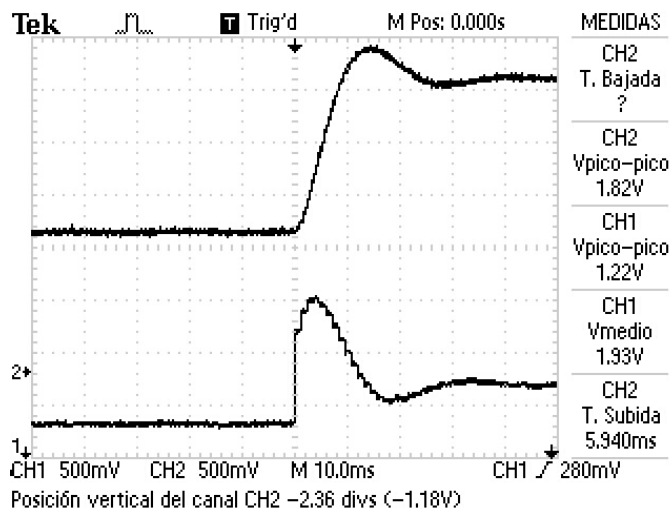


(a) Estimador de predicción (clock aumentado).

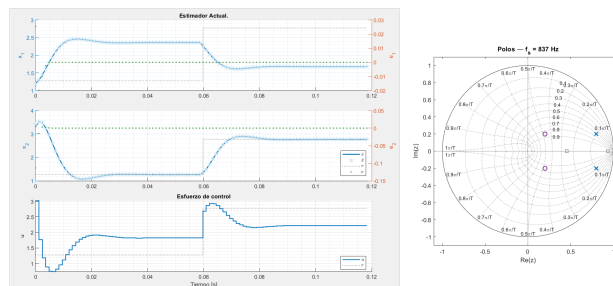


(b) Estimador de predicción (simulación).

Figura 10. Respuesta experimental con el clock aumentado vs simulación.



(a) Estimador de actual (clock aumentado).



(b) Estimador de actual (simulación).

Figura 11. Respuesta experimental con el clock aumentado vs simulación.

APÉNDICE

En este anexo se incluyen los archivos en C utilizados en el PSoC para implementar los dos esquemas de control (estimador de predicción y estimador actual), así como las librerías de apoyo y los scripts de MATLAB.

A. Código en C — Estimador predictivo

```
1  #include "project.h"
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdlib.h>
5  #include <ctype.h>
6
7  #include "config.h"
8  #include "mat.h"          /* tu librería */
9  #include "io.h"
10 #include "serial.h"
11 #include "control.h"
12
13 /* ===== Estado global ===== */
14 volatile char   flag = '0';
15 volatile double ref_max = (float) (A_AMPL/2);
16 volatile double ref_min = (float) (-A_AMPL/2);
17 volatile double ref      = 0.0;          /* referencia en AC */
18 volatile double y_last_volts = 0.0;      /* última muestra del ADC (tu hardware ya entrega AC)
19                                          */
20
21 volatile uint8 mode = MODE_CLOSED;
22 static control_state_t ctrl_st;
23
24 /* ===== PEGAR EN main.c ===== */
25 static const control_model_t M = {
26     .Ad = {.filas=2,.columnas=2,.d={{0.932581, 0.000000},
27                                     {-1.145618, 0.455938}}},
28     .Bd = {.filas=2,.columnas=1,.d={{-0.117824},
29                                     {0.080093}}},
30     .Cd = {.filas=1,.columnas=2,.d={{0.000000, 1.000000}}},
31     .K  = {.filas=1,.columnas=2,.d={{2.907022, 1.636048}}},
32     .K0 = {.filas=1,.columnas=1,.d={{0.569878}}},
33     .L  = {.filas=2,.columnas=1,.d={{-0.503374},
34                                     {0.988519}}}
35 };
36
37 /* poles(A):   0.455938  0.932581 */
38 /* poles(A-BK): 0.800000  0.800000 */
39 /* poles(A-LC): 0.200000  0.200000 */
40 /* Ts = 1.19483659e-03 s */
41 /* Podés cambiar a CL_CURRENT por UART ('law:curr') si luego querés */
42 static control_law_t g_law = CL_PREDICTOR;
43
44 /* ===== ISR de muestreo ===== */
45 CY_ISR(adquirirMuestra)
46 {
47     /* NO limpiar pending aquí; el EOC siguiente genera un nuevo pulso */
48     /* NO deshabilitar la interrupción aquí; la dejamos armada permanentemente */
49
50     /* Conmutar referencia (en AC) por pin/switch */
51     ref = (ref_state_Read()) ? ref_max : ref_min;
52
53     /* Mostrar la referencia en el DAC auxiliar (para el osciloscopio) */
54     io_ref_write((float) (ref + VDDA2));
55
56     /* Leer ADC -> en tu hardware ya está en AC (centrado) */
57     y_last_volts = io_adc_read_volts();
58
59     flag = '1';
60 }
```

```

60
61 /* ===== Utilidades ===== */
62 static long my_lroundf(float x) { return (x >= 0.0f) ? (long)(x + 0.5f) : (long)(x - 0.5f); }
63 /* ===== helpers de parsing y print ===== */
64 static float parse_float_local(const char *s) {
65     char buf[32]; size_t i=0;
66     while (*s && i<sizeof(buf)-1) {
67         char c = *s++;
68         if (c==',' || c=='.') continue;
69         if (!(c>='0'&&c<='9' || c=='+' || c=='-' || c=='.' || c=='e' || c=='E')) break;
70         buf[i++] = c;
71     }
72     buf[i]='\0';
73     return (float)atof(buf);
74 }
75
76 static long parse_uint_local(const char *s) {
77     while (*s==' ' || *s=='\t') ++s;
78     long v=0; while (*s>='0'&&*s<='9') { v = v*10 + (*s-'0'); ++s; }
79     return v;
80 }
81
82 /* Set de amplitud 'a' => ref_max = +a/2, ref_min = -a/2, imprime en mV */
83 static void set_amp(float a) {
84     if (a < 0.0f || a > 1.5f) { serial_puts("ERR: a out of range (0..1.5)\r\n"); return; }
85     uint8 intr = CyEnterCriticalSection();
86     ref_max = +a*0.5f; ref_min = -a*0.5f;
87     ref = (ref_state_Read()) ? ref_max : ref_min;
88     CyExitCriticalSection(intr);
89
90     /* imprimir sin %f */
91     int amp_mV = (int)(a*1000.0f + 0.5f);
92     serial_puts("OK: a="); char msg[32]; snprintf(msg, sizeof(msg), "%d mVpp\r\n", amp_mV);
93     serial_puts(msg);
94 }
95
96 /* STATUS reducido: m, a, p (sin %f) */
97 static void cmd_status(void) {
98     float a = (float)(ref_max - ref_min);
99     unsigned per = (unsigned)timer_ref_ReadPeriod();
100     int a_mV = (int)(a*1000.0f + 0.5f);
101     char msg[96];
102     snprintf(msg, sizeof(msg),
103         "STATUS:\r\n m=%s\r\n a=%d mVpp\r\n p=%u\r\n",
104         (mode==MODE_OPEN)?"open":"closed", a_mV, per);
105     serial_puts(msg);
106 }
107 static void on_serial_line(const char *line_in)
108 {
109     char line[RX_BUF_SZ];
110     strncpy(line, line_in, sizeof(line));
111     line[sizeof(line)-1]='\0';
112
113     /* trim/basura */
114     size_t n=strlen(line);
115     while (n && (line[n-1]=='\r' || line[n-1]=='\n' || isspace((unsigned char)line[n-1]))) line[--n]='\0';
116     char *p=line; while(*p && isspace((unsigned char)*p)) ++p;
117     while(*p && !isalnum((unsigned char)*p)) ++p;
118     for(char *q=p; *q; ++q) *q=(char)tolower((unsigned char)*q);
119
120     if (!strcmp(p, "s") || !strcmp(p, "status")) { cmd_status(); return; }
121     if (!strcmp(p, "help") || !strcmp(p, "?")) {
122         serial_puts("a:<0..1.5>, p:<uint16>, m:<open|closed|1|0>, s\r\n"); return;
123     }
124
125     char *sep = strchr(p, ':');

```

```

125     if (!sep) { sep=strchr(p, ' '); if(sep) *sep=': '; }
126     if (!sep) { serial_puts("ERR: expected a:<val> | p:<val> | m:<val> | s\r\n"); return; }
127     *sep='\0'; const char *key=p, *val=sep+1; while(*val && isspace((unsigned char)*val)) ++
    val;
128
129     if (!strcmp(key, "a")) { set_amp(parse_float_local(val)); return; }
130
131     if (!strcmp(key, "p")) {
132         long per = parse_uint_local(val);
133         if (per<0 || per>65535) { serial_puts("ERR: p out of range (0..65535)\r\n"); return; }
134
135         timer_ref_Stop(); timer_ref_WritePeriod((uint16)per); timer_ref_WriteCounter((uint16)
    per); timer_ref_Start();
136         serial_puts("OK: p set\r\n"); return;
137     }
138
139     if (!strcmp(key, "m")) {
140         if (!strcmp(val, "open") || !strcmp(val, "1")) { mode=MODE_OPEN; serial_puts("OK: m=open\r
    n"); }
141         else if (!strcmp(val, "closed") || !strcmp(val, "0")) { mode=MODE_CLOSED; serial_puts("OK:
    m=closed\r\n"); }
142         else { serial_puts("ERR: m must be open|closed|1|0\r\n"); }
143         return;
144     }
145     serial_puts("ERR: unknown key (use a, p, m, s)\r\n");
146 }
147
148
149
150
151 /* ===== main ===== */
152 int main(void)
153 {
154     CyGlobalIntEnable;
155
156     io_init();
157     control_init(&ctrl_st);
158     serial_init(on_serial_line);
159
160     /* Timer/ISR de muestreo */
161     muestra_disponible_StartEx(adquirirMuestra);
162     muestra_disponible_ClearPending(); /* arranque limpio */
163     muestra_disponible_Enable();
164     timer_ref_Start();
165
166     for(;;) {
167         /* UART no bloqueante */
168         serial_poll();
169
170         /* procesar 1 muestra */
171         if (flag=='1') {
172             flag='0';
173
174             float u_phys = control_step(&ctrl_st, &M,
175                                     (float)ref, /* AC */
176                                     (float)y_last_volts, /* AC en tu hardware */
177                                     mode,
178                                     g_law);
179
180             io_dac_write(u_phys);
181
182             /* ya NO re-armamos la IRQ: queda habilitada siempre */
183         }
184     }
185 }

```

Listing 5. main.c: bucle principal, adquisición y lógica de control con estimador predictivo.

B. Código en C — Estimador actual

```

1  #include "project.h"
2  #include "config.h"    // SAT_MIN, SAT_MAX, VDDA2, A_AMPL
3
4  /* ===== Helpers ===== */
5  static inline uint8_t volt_to_dac(float v)
6  {
7      if (v < SAT_MIN) v = SAT_MIN;
8      if (v > SAT_MAX) v = SAT_MAX;
9      float n = v / SAT_MAX; if (n < 0) n = 0; if (n > 1) n = 1;
10     return (uint8_t)(255.0f * n);
11 }
12 static inline float sat_u_phys(float u_phys)
13 {
14     if (u_phys > SAT_MAX) u_phys = SAT_MAX;
15     if (u_phys < SAT_MIN) u_phys = SAT_MIN;
16     return u_phys;
17 }
18
19 /* ===== Parámetros del modelo (DISCRETO, en AC) =====
20 Copiados de tu export bueno (coherentes con la simulación):
21 A = [ 0.932581    0
22       -1.145618  0.455938 ]
23 B = [ -0.117824
24        0.080093 ]
25 C = [ 0  1 ]
26 K = [ 2.907022  1.636048 ]
27 K0 = 0.569878
28 Ke (L_current) = [ -0.503374
29                    0.988519 ]
30 */
31 static const float a11 = 0.932581f, a12 = 0.000000f;
32 static const float a21 = -1.145618f, a22 = 0.455938f;
33 static const float b1  = -0.117824f, b2  = 0.080093f;
34
35 static const float k1  = 2.907022f, k2  = 1.636048f;
36 static const float k0  = 0.569878f;
37
38 static const float l1  = -0.503374f, l2  = 0.988519f;
39
40 /* ===== Estado global ===== */
41 static volatile uint8_t tick_flag = 0;
42 static volatile float  ref_ac     = 0.0f;    // referencia en AC (+-A/2)
43 static float  xhat1 = 0.0f, xhat2 = 0.0f;    // estimador 2x1
44
45 /* Para probar sin UART: amplitud fija A_AMPL */
46 static const float ref_max = 0.5f * A_AMPL;
47 static const float ref_min = -0.5f * A_AMPL;
48
49 /* ===== ISR de tick (NO des/clear dentro de la ISR) ===== */
50 CY_ISR(adquirirMuestra)
51 {
52     // referencia por pin (AC)
53     ref_ac = ref_state_Read() ? ref_max : ref_min;
54     // DAC de referencia (para el osciloscopio)
55     VDAC8_ref_SetValue(volt_to_dac(ref_ac + VDDA2));
56
57     // marcar que hay que ejecutar un paso de control
58     tick_flag = 1u;
59
60     // NO llamar Disable/Enable/ClearPending aquí.
61 }

```

```

62
63  /* ===== main ===== */
64  int main(void)
65  {
66      CyGlobalIntEnable;
67
68      /* Analógico / DAC / ADC */
69      Opa_dac_Start();
70      Opa_stage1_Start();
71      Opa_vdda_2_Start();
72      Opa_stage2_Start();
73
74      VDAC8_Start();
75      VDAC8_ref_Start();
76
77      VADC_Start(); // Configurar el ADC en "Firmware Trigger" (no free run)
78
79      /* Timer + ISR */
80      isr_tick_StartEx(adquirirMuestra);
81      timer_ref_Start(); // fija tu Ts
82
83      for (;;)
84      {
85          if (tick_flag)
86          {
87              tick_flag = 0u;
88
89
90              float u_ac = k0 * ref_ac - (k1 * xhat1 + k2 * xhat2);
91
92              /* ===== 2) Aplicar esfuerzo (físico) y saturar ===== */
93              float u_phys = sat_u_phys(u_ac + VDDA2);
94              VDAC8_SetValue(volt_to_dac(u_phys));
95
96              /* u aplicado en AC (post-sat) */
97              float u_ac_apl = u_phys - VDDA2;
98
99              /* ===== 3) Predicción a k+1: z = A*xhat + B*u_aplicado ===== */
100             float z1 = a11 * xhat1 + a12 * xhat2 + b1 * u_ac_apl;
101             float z2 = a21 * xhat1 + a22 * xhat2 + b2 * u_ac_apl;
102
103             /* ===== 4) Medición y[k+1] (AC) ===== */
104             VADC_StartConvert();
105             while (VADC_IsEndConversion(VADC_RETURN_STATUS) == 0) { /* busy wait corto */ }
106             float y_ac = VADC_CountsTo_Volts(VADC_GetResult16()); // tu hardware ya entrega
AC
107
108
109             float innov = y_ac - z2;
110             xhat1 = z1 + l1 * innov;
111             xhat2 = z2 + l2 * innov;
112         }
113     }
114 }
115

```

Listing 6. main.c: bucle principal, adquisición y lógica de control con estimador actual.

C. Código en C — Versión simplificada (debug)

```

1  #include "project.h"
2  #include "config.h"
3
4  /* ----- Helpers ----- */
5  static inline uint8_t volt_to_dac(float v)
6  {
7      if (v < SAT_MIN) v = SAT_MIN;

```



```

8     if (v > SAT_MAX) v = SAT_MAX;
9     float n = v / SAT_MAX; if (n < 0) n = 0; if (n > 1) n = 1;
10    return (uint8_t)(255.0f * n);
11 }
12 static inline float sat_u_phys(float u)
13 {
14     if (u > SAT_MAX) u = SAT_MAX;
15     if (u < SAT_MIN) u = SAT_MIN;
16     return u;
17 }
18
19 /* ----- Modelo discreto ----- */
20 float a11 = 0.932581f, a12 = 0.000000f;
21 float a21 = -1.145618f, a22 = 0.455938f;
22 float b1 = -0.117824f, b2 = 0.080093f;
23 float c1 = 0.0f, c2 = 1.0f;
24
25 float k1 = 2.907022f, k2 = 1.636048f;
26 float k0 = 0.569878f;
27
28 float l1 = -0.503374f, l2 = 0.988519f;
29
30 /* ----- Estado ----- */
31 static volatile uint8_t tick_flag = 0;
32 static volatile float ref_ac = 0.0f;
33 static float xhat1 = 0.0f, xhat2 = 0.0f;
34
35 static const float ref_max = 0.5f * A_AMPL;
36 static const float ref_min = -0.5f * A_AMPL;
37
38 /* ----- ISR de muestreo ----- */
39 CY_ISR(adquirirMuestra)
40 {
41     /* referencia AC según el switch */
42     ref_ac = ref_state_Read() ? ref_max : ref_min;
43     /* DAC auxiliar para scope */
44     VDAC8_ref_SetValue(volt_to_dac(ref_ac + VDDA2));
45     tick_flag = 1u;
46 }
47
48 /* ----- MAIN ----- */
49 int main(void)
50 {
51     CyGlobalIntEnable;
52
53     /* Front-end analógico */
54     Opa_dac_Start();
55     Opa_stage1_Start();
56     Opa_vdda_2_Start();
57     Opa_stage2_Start();
58
59     VDAC8_Start();
60     VDAC8_ref_Start();
61
62     VADC_Start(); // Firmware trigger (NO free-run)
63
64     muestra_disponible_StartEx(adquirirMuestra);
65     timer_ref_Start(); // genera Ts
66
67     for (;;)
68     {
69         if (tick_flag)
70         {
71             tick_flag = 0u;
72
73             float u_ac = k0 * ref_ac - (k1 * xhat1 + k2 * xhat2);

```

```

75
76     /* 2) Leer y[k] (AC) */
77     VADC_StartConvert();
78     while (VADC_IsEndConversion(VADC_RETURN_STATUS) == 0) {}
79     float y_ac = VADC_CountsTo_Volts(VADC_GetResult16());
80
81
82     float y_hat = c1 * xhat1 + c2 * xhat2;
83     float err = y_ac - y_hat;
84     float x1n = a11 * xhat1 + a12 * xhat2 + b1 * u_ac + l1 * err;
85     float x2n = a21 * xhat1 + a22 * xhat2 + b2 * u_ac + l2 * err;
86     xhat1 = x1n;
87     xhat2 = x2n;
88
89     /* 4) DAC físico (con saturación y offset) */
90     float u_phys = sat_u_phys(u_ac + VDDA2);
91     VDAC8_SetValue(volt_to_dac(u_phys));
92 }
93 }
94 }

```

Listing 7. main.c: versión simplificada del estimador de predicción.

D. Librerías del proyecto

```

1  #ifndef CONFIG_H
2  #define CONFIG_H
3
4  /* === Límites y referencias físicas === */
5  #define SAT_MIN      (0.0f)
6  #define SAT_MAX      (4.048f)
7  #define MIDDLE_VOLTAGE ((SAT_MAX - SAT_MIN)/2.0f)
8  #define VDDA2        SAT_MAX/2
9
10 /* UART */
11 #define RX_BUF_SZ 64
12
13 /* Amplitud para referencia cuadrada (renombrado: NO usar 'A') */
14 #define A_AMPL 0.5f
15
16 /* Modo de operación */
17 enum { MODE_CLOSED = 0, MODE_OPEN = 1 };
18
19 #endif

```

Listing 8. config.h: parámetros globales del sistema.

```

1  #include "control.h"
2  #include "config.h" /* SAT_MIN, SAT_MAX, VDDA2 */
3  #include <string.h>
4
5  /* Saturación de esfuerzo físico */
6  static inline float sat_u_phys(float u)
7  {
8      if (u > SAT_MAX) u = SAT_MAX;
9      if (u < SAT_MIN) u = SAT_MIN;
10     return u;
11 }
12
13 void control_init(control_state_t *st)
14 {
15     /* xhat = 0 (2x1) */
16     st->xhat.filas = 2;
17     st->xhat.columnas = 1;
18     st->xhat.d[0][0] = 0.0;
19     st->xhat.d[1][0] = 0.0;
20 }

```

```

21
22  /* u = K0*r - K*xhat (AC); observador según 'law' */
23  float control_step(control_state_t *st,
24                      const control_model_t *m,
25                      float ref_ac,
26                      float y_volts,
27                      uint8_t mode,
28                      control_law_t law)
29  {
30      /* Lazo abierto: el DAC debe seguir la referencia en físico */
31      if (mode == 1 /* MODE_OPEN */) {
32          float u_phys_ol = ref_ac + VDDA2;
33          return sat_u_phys(u_phys_ol);
34      }
35
36
37      double y_ac = (double)y_volts; /* ya AC según tu circuito */
38
39      /* u_ac = K0*r - K*xhat */
40      Mat R = (Mat){ .filas=1,.columnas=1,.d={{ ref_ac }} };
41      Mat KX = (Mat){ .filas=1,.columnas=1,.d={{ 0.0 }} };
42      Mat K0R = (Mat){ .filas=1,.columnas=1,.d={{ 0.0 }} };
43
44      mat_mul(&m->K, &st->xhat, &KX);
45      mat_mul(&m->K0, &R, &K0R);
46
47      double u_ac = K0R.d[0][0] - KX.d[0][0];
48
49      /* Llevar a físico y saturar SOLO al salir */
50      float u_phys = sat_u_phys((float)(u_ac + (double)VDDA2));
51
52      /* u realmente aplicado en AC (post-sat) */
53      double u_ac_apl = (double)u_phys - (double)VDDA2;
54
55      /* Observador */
56      Mat Cx = (Mat){ .filas=1,.columnas=1,.d={{ 0.0 }} };
57      mat_mul(&m->Cd, &st->xhat, &Cx);
58      double e_y = y_ac - Cx.d[0][0];
59
60      Mat Ax = (Mat){ .filas=2,.columnas=1,.d={{ 0.0 },{ 0.0 }} };
61      Mat Bu = (Mat){ .filas=2,.columnas=1,.d={{ 0.0 },{ 0.0 }} };
62      Mat Le = (Mat){ .filas=2,.columnas=1,.d={{ 0.0 },{ 0.0 }} };
63
64      Mat U1 = (Mat){ .filas=1,.columnas=1,.d={{ 0.0 }} };
65      Mat E1 = (Mat){ .filas=1,.columnas=1,.d={{ e_y }} };
66
67      switch (law) {
68
69          case CL_PREDICTOR:
70
71              U1.d[0][0] = u_ac_apl;
72              mat_mul(&m->Ad, &st->xhat, &Ax);
73              mat_mul(&m->Bd, &U1, &Bu);
74              mat_mul(&m->L, &E1, &Le);
75
76              st->xhat.d[0][0] = Ax.d[0][0] + Bu.d[0][0] + Le.d[0][0];
77              st->xhat.d[1][0] = Ax.d[1][0] + Bu.d[1][0] + Le.d[1][0];
78              break;
79
80          case CL_CURRENT:
81
82          default:
83
84              U1.d[0][0] = u_ac_apl;
85              mat_mul(&m->Ad, &st->xhat, &Ax);
86              mat_mul(&m->Bd, &U1, &Bu);
87

```

```

88
89     Ax.d[0][0] += Bu.d[0][0];
90     Ax.d[1][0] += Bu.d[1][0];
91
92
93     Mat Cx2 = (Mat){ .filas=1, .columnas=1, .d={{ 0.0 }} };
94     mat_mul(&m->Cd, &Ax, &Cx2);
95     {
96         double e2 = y_ac - Cx2.d[0][0];
97         Mat E2 = (Mat){ .filas=1, .columnas=1, .d={{ e2 }} };
98         mat_mul(&m->L, &E2, &Le);
99     }
100
101     st->xhat.d[0][0] = Ax.d[0][0] + Le.d[0][0];
102     st->xhat.d[1][0] = Ax.d[1][0] + Le.d[1][0];
103     break;
104 }
105
106 return u_phys;
107 }

```

Listing 9. control.c: cálculo de la ley de control.

```

1  #ifndef CONTROL_H
2  #define CONTROL_H
3
4  #include <stdint.h>
5  #include "mat.h"
6
7
8  typedef enum {
9      CL_PREDICTOR = 0,
10     CL_CURRENT   = 1
11 } control_law_t;
12
13
14 typedef struct {
15     Mat Ad;    /* 2x2 */
16     Mat Bd;    /* 2x1 */
17     Mat Cd;    /* 1x2 */
18     Mat K;     /* 1x2 */
19     Mat K0;    /* 1x1 (prefiltro / Nbar / K0) */
20     Mat L;     /* 2x1 (ganancias del observador) */
21 } control_model_t;
22
23 /* Estado interno del estimador */
24 typedef struct {
25     Mat xhat; /* 2x1 */
26 } control_state_t;
27
28 /* Inicializa el estado del estimador en cero */
29 void control_init(control_state_t *st);
30
31
32 float control_step(control_state_t *st,
33                   const control_model_t *m,
34                   float ref_ac,
35                   float y_volts,
36                   uint8_t mode,
37                   control_law_t law);
38
39 #endif /* CONTROL_H */

```

Listing 10. control.h: prototipos y estructuras del módulo de control.

```

1  #include "project.h"
2  #include "config.h"

```

```

3  #include "io.h"
4
5  uint8_t io_volt_to_dac(float v)
6  {
7      if (v < SAT_MIN) v = SAT_MIN;
8      if (v > SAT_MAX) v = SAT_MAX;
9      float norm = v / SAT_MAX; /* 0..1 */
10     if (norm < 0.0f) norm = 0.0f;
11     if (norm > 1.0f) norm = 1.0f;
12     return (uint8_t)(norm * 255.0f);
13 }
14
15 void io_init(void)
16 {
17     Opa_dac_Start();
18     Opa_stage1_Start();
19     Opa_vdda_2_Start();
20     Opa_stage2_Start();
21     PGA_ref_Start();
22
23     VDAC8_Start();
24     VDAC8_ref_Start();
25
26     VADC_Start();
27 }
28
29 float io_adc_read_volts(void)
30 {
31     /* Bloqueante corto (ya está convertido por el timer/ISR) */
32     return (float)VADC_CountsTo_Volts(VADC_GetResult16());
33 }
34
35 void io_dac_write(float volts)
36 {
37     VDAC8_SetValue(io_volt_to_dac(volts));
38 }
39
40 void io_ref_write(float volts)
41 {
42     VDAC8_ref_SetValue(io_volt_to_dac(volts));
43 }

```

Listing 11. io.c: funciones de entrada/salida (ADC, DAC, UART).

```

1  #ifndef IO_H
2  #define IO_H
3  #include <stdint.h>
4
5  /* Init de periféricos HW */
6  void io_init(void);
7
8  /* ADC -> voltios */
9  float io_adc_read_volts(void);
10
11 /* DAC principal: esfuerzo */
12 void io_dac_write(float volts);
13
14 /* DAC auxiliar: referencia visual al scope */
15 void io_ref_write(float volts);
16
17 /* Map 0..SAT_MAX -> 0..255 con clamps */
18 uint8_t io_volt_to_dac(float v);
19
20 #endif

```

Listing 12. io.h: prototipos de E/S.

```

1  #include "mat.h"
2
3
4  void mat_zero_volatile(volatile Mat *M, int r, int c) {
5      M->filas = r; M->columnas = c;
6      for (int i = 0; i < r; i++)
7          for (int j = 0; j < c; j++)
8              M->d[i][j] = 0.0;
9  }
10
11
12  // ===== Implementaciones =====
13  void mat_zero(Mat *M, int r, int c) {
14      M->filas = r; M->columnas = c;
15      for (int i = 0; i < r; i++)
16          for (int j = 0; j < c; j++)
17              M->d[i][j] = 0.0;
18  }
19
20  void mat_copy(Mat *dst, const Mat *src) {
21      dst->filas = src->filas; dst->columnas = src->columnas;
22      for (int i = 0; i < src->filas; i++)
23          for (int j = 0; j < src->columnas; j++)
24              dst->d[i][j] = src->d[i][j];
25  }
26
27
28  void mat_copy_volatile(Mat *dst, volatile Mat *src) {
29      dst->filas = src->filas; dst->columnas = src->columnas;
30      for (int i = 0; i < src->filas; i++)
31          for (int j = 0; j < src->columnas; j++)
32              dst->d[i][j] = src->d[i][j];
33  }
34
35  void mat_add(const Mat *A, const Mat *B, Mat *C) {
36      C->filas = A->filas; C->columnas = A->columnas;
37      for (int i = 0; i < A->filas; i++)
38          for (int j = 0; j < A->columnas; j++)
39              C->d[i][j] = A->d[i][j] + B->d[i][j];
40  }
41
42  void mat_sut(const Mat *A, const Mat *B, Mat *C) {
43      C->filas = A->filas; C->columnas = A->columnas;
44      for (int i = 0; i < A->filas; i++)
45          for (int j = 0; j < A->columnas; j++)
46              C->d[i][j] = A->d[i][j] - B->d[i][j];
47  }
48
49  void mat_mul(const Mat *A, const Mat *B, Mat *C) {
50      C->filas = A->filas; C->columnas = B->columnas;
51      for (int i = 0; i < A->filas; i++) {
52          for (int j = 0; j < B->columnas; j++) {
53              double acc = 0.0;
54              for (int k = 0; k < A->columnas; k++)
55                  acc += A->d[i][k] * B->d[k][j];
56              C->d[i][j] = acc;
57          }
58      }
59  }
60
61  void ss_step(const Mat *A, const Mat *B, const Mat *C, const Mat *D,
62              const Mat *xk, const Mat *uk, Mat *xk1, Mat *yk)
63  {
64      Mat Ax, Bu, Cx, Du;
65      mat_mul(A, xk, &Ax);
66      mat_mul(B, uk, &Bu);

```

```

67     mat_add(&Ax, &Bu, xk1);
68
69     mat_mul(C, xk, &Cx);
70     mat_mul(D, uk, &Du);
71     mat_add(&Cx, &Du, yk);
72 }

```

Listing 13. mat.c: operaciones matriciales.

```

1  #ifndef MAT_H
2  #define MAT_H
3
4  #define MAX 10
5
6  // ===== Estructura de Matriz =====
7  typedef struct {
8      int filas;
9      int columns;
10     double d[MAX][MAX];
11 } Mat;
12
13 // ===== Prototipos =====
14 void mat_zero_volatile(volatile Mat *M, int r, int c);
15 void mat_zero(Mat *M, int r, int c);
16 void mat_copy(Mat *dst, const Mat *src);
17 void mat_copy_volatile(Mat *dst, volatile Mat *src);
18 void mat_add(const Mat *A, const Mat *B, Mat *C);
19 void mat_sut(const Mat *A, const Mat *B, Mat *C);
20 void mat_mul(const Mat *A, const Mat *B, Mat *C);
21 void ss_step(const Mat *A, const Mat *B, const Mat *C, const Mat *D,
22             const Mat *xk, const Mat *uk, Mat *xk1, Mat *yk);
23
24 #endif // MAT_H

```

Listing 14. mat.h: definición de Mat y prototipos.

```

1  #include "project.h"
2  #include "config.h"
3  #include "serial.h"
4  #include <string.h>
5  #include <ctype.h>
6  #include <stdio.h>
7
8  static serial_line_cb_t g_on_line = 0;
9  static char rx_buf[RX_BUF_SZ];
10 static uint8_t rx_len = 0;
11
12 void serial_init(serial_line_cb_t on_line)
13 {
14     g_on_line = on_line;
15     PC_Start();
16     PC_PutString(
17         "\r\nReady. Commands:\r\n"
18         "  a:<float 0..1.5>      amplitud AC (admite 0,6 o 0.6)\r\n"
19         "  p:<uint16>            periodo timer\r\n"
20         "  m:<open|closed|1|0>   modo\r\n"
21         "  s                    status\r\n"
22     );
23 }
24
25
26
27 void serial_puts(const char *s) { PC_PutString(s); }
28
29 /* sane \r\n handling + overflow protection */
30 void serial_poll(void)
31 {

```

```

32     int ch;
33     while ((ch = PC_GetChar()) != 0) {
34         if (ch == '\r' || ch == '\n') {
35             if (rx_len > 0) {
36                 rx_buf[rx_len] = '\0';
37                 if (g_on_line) g_on_line(rx_buf);
38                 rx_len = 0;
39             }
40             /* colapsar múltiples \r\n consecutivos */
41         } else {
42             if (rx_len < (RX_BUF_SZ - 1)) {
43                 rx_buf[rx_len++] = (char)ch;
44             } else {
45                 rx_len = 0;
46                 PC_PutString("ERR: line too long\r\n");
47             }
48         }
49     }
50 }

```

Listing 15. serial.c: funciones de comunicación serie.

```

1  #ifndef SERIAL_H
2  #define SERIAL_H
3  #include <stdint.h>
4  #include <stddef.h>
5
6  typedef void (*serial_line_cb_t)(const char *line);
7
8  /* Inicializa UART PC y registra callback por línea */
9  void serial_init(serial_line_cb_t on_line);
10
11 /* Poll no-bloqueante: drena RX, arma líneas y llama callback */
12 void serial_poll(void);
13
14 /* Envío utilitario */
15 void serial_puts(const char *s);
16
17 #endif

```

Listing 16. serial.h: prototipos del módulo serie.

E. Código en MATLAB

```

1
2 clear all; clc; close all;
3 addpath('..\');
4 %% --- Planta continua (tu modelo) ---
5 C2 = 103.07e-9; C1 = 211.1e-9;
6 R3 = 6.8e3; R4 = 14.760e3;
7 R1 = 47e3; R2 = 81.09e3;
8
9 tau1 = R2*C1; k1 = -R2/R1;
10 tau2 = R4*C2; k2 = -R4/R3;
11
12 F = [ -1/tau1, 0;
13       k2/tau2, -1/tau2 ];
14 G = [ k1/tau1; 0 ];
15 H = [ 0 1];
16 J = 0;
17
18 sysC = ss(F,G,H,J);
19 Gc = tf(sysC);
20
21 % Ts base vía polo más rápido continuo (heurística)
22 fn = max(abs(zpk(Gc).P{1}))/pi;
23 Tn = 1/fn;

```



```

24 Ts = Tn/4; % un único Ts; ajustá si querés
25
26 sysD = c2d(sysC, Ts, 'zoh');
27 [A,B,C,D] = ssdata(sysD);
28
29 %% --- Chequeo de controlabilidad/observabilidad, como pediste ---
30 Co = ctrb(A,B);
31 Ob = obsv(A,C);
32 rCo = rank(Co);
33 rOb = rank(Ob);
34 n = size(A,1);
35
36 fprintf('rank(ctrb) = %d, n = %d\n', rCo, n);
37 fprintf('rank(observ) = %d, n = %d\n', rOb, n);
38
39 %% --- Polos objetivo en z (únicos y fijos) ---
40 p_ctrl = [0.8 + 1j*0.2; 0.8 - 1j*0.2]; % TU pedido
41
42 % Ganancia de realimentación K por Ackermann
43 K = acker(A,B,p_ctrl);
44
45 % Prefiltro Nbar (para y_ss = r con D=0, SISO)
46
47 [~,~,Nbar] = refi(A,B,C,K);
48
49 %% --- "N veces más rápido" para el observador en discreto ---
50 p_obs = 0.2+1j*0.2; % eleva cada polo
51
52
53 p_obs = [p_obs(1), conj(p_obs(1))];
54
55 % Ganancia del observador por dualidad (esta es la forma correcta):
56 L_pred = acker(A', C', p_obs)'; % para estimador de PREDICCIÓN
57 L_pred_a = acker(A', (C*A)', p_obs)'; % para estimador ACTUAL
58
59
60
61 %% ===== SIMULACIÓN Y PLOTS (loop + función) =====
62 % Parámetros de simulación
63 N = 50;
64 M = N-10;
65 Amp = 1.5;
66 Tsim = 0:Ts:(2*N-1)*Ts;
67 r = Amp/2*[-ones(1,N), ones(1,N)];
68 x0 = Amp/2*[-1; 1];
69 xh0 = Amp/2*[-1.5; 1.5];
70
71 % Sistemas para polos
72 sys_open = ss(A,B,C,D,Ts);
73 Acl = A - B*K;
74 sys_cl = ss(Acl, B, C, D, Ts);
75
76 % --- después de simular ambos:
77 simPred = sim_predictor(A,B,C,D,K,L_pred, Nbar,Ts,Tsim,x0,xh0,r);
78 simPred.title = "Estimador Predictivo.";
79 simCurr = sim_current(A,B,C,D,K,L_pred_a, Nbar,Ts,Tsim,x0,xh0,r);
80 simCurr.title = "Estimador Actual.";
81 P_open = eig(A);
82 P_cl = eig(A - B*K);
83 r = r + 2.024;
84 sim = {simPred, simCurr};
85 for i = 1:numel(sim)
86     fancy_plot_states_control( ...
87         Tsim, ...
88         sim{i}.X, sim{i}.Xhat, sim{i}.U, r, ...
89         P_open, P_cl, sim{i}.P, sim{i}.title);

```

```

90 end
91
92
93 fancy_plot_states_control_combo( ...
94     Tsim, ...
95     simPred.X, simCurr.X, ...
96     simPred.Xhat, simCurr.Xhat, ...
97     simPred.U, simCurr.U, r, ...
98     P_open, P_cl, simPred.P, simCurr.P, ...
99     'Predicción vs Actual');
100
101 %% ===== FUNCIONES AUXILIARES =====
102 function out = sim_predictor(A,B,C,D,K,L,Nbar,Ts,T, x0,xh0,r)
103     nx = size(A,1); N = numel(T);
104     x = x0(:);      xh = xh0(:);
105     X = zeros(nx,N); XH = zeros(nx,N); U = zeros(1,N);
106
107     for k = 1:N
108         rk = r(k);
109
110         % 1) Control con estado estimado a k: u[k]
111         u = Nbar*rk - K*xh;
112
113         % 2) Medición a k (ANTES de propagar x): y[k]
114         y = C*x + D*u;      % si D=0, no cambia nada
115
116         % 3) Observador de PREDICCIÓN: usa y[k]
117         xh = A*xh + B*u + L*(y - C*xh);
118
119         % 4) Planta real a k+1
120         x = A*x + B*u;
121
122         % 5) Log
123         X(:,k) = x;      % estado real en k+1 (está bien si eres consistente)
124         XH(:,k) = xh;    % estimado en k+1
125         U(k) = u;
126     end
127
128     out.t = T;
129     out.X = X+2.024; out.Xhat = XH+2.024; out.U = U+2.024;
130     out.P = eig(A - L*C); % dinámicas del error
131 end
132
133
134
135 function out = sim_current(A,B,C,D,K,Ke,Nbar,Ts,T, x0,xh0,r)
136 % Estimador ACTUAL (current): corrige con y[k+1] sobre la predicción z[k+1]
137     nx = size(A,1);
138     N = numel(T);
139
140     x = x0(:);      % estado real
141     xh = xh0(:);    % estado estimado
142
143     X = zeros(nx,N);
144     XH = zeros(nx,N);
145     U = zeros(1, N);
146
147     for k = 1:N
148         rk = r(k);
149
150         % 1) control con ESTADO ACTUAL (corregido a k): u[k]
151         u = Nbar*rk - K*xh;
152
153         % 2) planta a k+1
154         x = A*x + B*u;
155

```

```

156     % 3) predicción del estimador a k+1
157     z = A*xh + B*u;
158     y = C*x + D*u; % (en tu sistema D=0)
159
160     % corrección actual (usa y[k+1])
161     xh = z + Ke*(y - C*z);
162
163     % log
164     X(:,k) = x;
165     XH(:,k) = xh;
166     U(k) = u;
167 end
168
169 out.t = T; out.X = X+2.024; out.Xhat = XH+2.024; out.U = U+2.024; out.P = eig(A - Ke*C*A);
170
171 end
172
173 function fancy_plot_states_control( ...
174     t, X, Xhat, U, r, ...
175     Popen, Pcl, Pobs, titulo)
176
177     n = size(X,1);
178     Ts = t(2) - t(1);
179
180     % Colores
181     c_x = [0 0.4470 0.7410]; % azul
182     c_xhat = [0.82 0.91 0.96]; % celeste claro
183     c_ref = [0.3 0.3 0.3]; % gris
184     c_err = [0.2 0.6 0.2]; % verde para error
185
186     fig = figure('Name', titulo, 'Color','w');
187     left = uipanel(fig,'Position',[0.05 0.08 0.58 0.87]);
188     right = axes (fig,'Position',[0.68 0.10 0.28 0.82]); %#ok<LAXES>
189
190     tl = tiledlayout(left, n+1, 1, 'TileSpacing','compact','Padding','compact');
191
192     % ===== Estados (x, xhat) + referencia y error e = x - xhat (eje derecho) =====
193     for k = 1:n
194         ax = nexttile(tl); hold(ax,'on');
195
196         % Izquierda: señales principales
197         yyaxis(ax,'left');
198         hX = plot(ax, t, X(k,:), '-', 'LineWidth', 1.6, 'Color', c_x, 'DisplayName',
199             '$x$');
200         hXhat = stairs(ax, t, Xhat(k,:), 'pentagram', 'LineWidth', 1.2, 'Color', c_xhat, '
201             DisplayName', '$\hat{x}$');
202         hR = stairs(ax, t, r, ':', 'LineWidth', 1.0, 'Color', c_ref, '
203             DisplayName', '$r$');
204         ylabel(ax, sprintf('x_%d',k));
205         grid(ax,'on'); grid(ax,'minor');
206
207         % Derecha: error
208         yyaxis(ax,'right');
209         e = X(k,:) - Xhat(k,:);
210         hE = stairs(ax, t, e, '.', 'LineWidth', 1.2, 'Color', c_err, 'DisplayName', '$e$');
211         yline(ax, 0, ':', 'Color',[0.5 0.5 0.5], 'HandleVisibility','off');
212         ylabel(ax, sprintf('e_%d',k));
213
214         if k==1
215             title(ax, titulo);
216         end
217         if k==n
218             % Creamos leyenda con handles explícitos y la bloqueamos
219             yyaxis(ax,'left');
220             lg = legend(ax, [hX, hXhat, hR, hE], {'$x$','$\hat{x}$','$r$','$e$'}, 'Location',
221                 'best');

```

```

218         set(lg, 'Interpreter', 'latex', 'AutoUpdate', 'off');
219     end
220 end
221
222 % ===== Control U + r =====
223 axu = nexttile(tl); hold(axu, 'on');
224 hU = stairs(axu, t, U, '-', 'LineWidth', 1.4, 'Color', c_x, 'DisplayName', '$u$');
225 hR = stairs(axu, t, r, ':', 'LineWidth', 1.0, 'Color', c_ref, 'DisplayName', '$r$');
226 grid(axu, 'on'); grid(axu, 'minor');
227 ylabel(axu, 'u'); xlabel(axu, 'Tiempo [s]');
228 title(axu, 'Esfuerzo de control');
229 lg2 = legend(axu, [hU, hR], {'$u$', '$r$'}, 'Location', 'best');
230 set(lg2, 'Interpreter', 'latex', 'AutoUpdate', 'off');
231
232 % ===== Polos (zgrid, sin legend) =====
233 axes(right); cla(right); hold(right, 'on');
234 zgrid; axis(right, 'equal'); axis(right, [-1.1 1.1 -1.1 1.1]);
235 xlabel(right, 'Re(z)'); ylabel(right, 'Im(z)');
236 title(right, sprintf('Polos -- f_s = %.0f Hz', 1/Ts));
237
238 % Ejes en Re=0 e Im=0 (NO entran al legend)
239 xline(right, 0, '-', 'Color', [0.5 0.5 0.5], 'LineWidth', 0.8, 'HandleVisibility', 'off');
240 yline(right, 0, '-', 'Color', [0.5 0.5 0.5], 'LineWidth', 0.8, 'HandleVisibility', 'off');
241
242 plot(right, real(Popen), imag(Popen), 's', 'MarkerSize', 7, 'LineWidth', 1.2, 'Color', [0.6 0.6 0.6]);
243 plot(right, real(Pcl), imag(Pcl), 'x', 'MarkerSize', 9, 'LineWidth', 1.6, 'Color', [0 0.4470 0.7410]);
244 plot(right, real(Pobs), imag(Pobs), 'o', 'MarkerSize', 7, 'LineWidth', 1.4, 'Color', [0.4940 0.1840 0.5560]);
245 end
246
247
248 function fancy_plot_states_control_combo( ...
249     t, X_pred, X_act, Xhat_pred, Xhat_act, U_pred, U_act, r, ...
250     Popen, Pcl, Pobs_pred, Pobs_act, titulo)
251
252     n = size(X_pred, 1);
253     Ts = t(2) - t(1);
254
255     % Colores consistentes
256     c_pred = [0 0.4470 0.7410]; % azul
257     c_act = [0.70 0.00 0.00]; % rojo oscuro
258     c_xhatpred = [0.8500 0.3250 0.0980]; % naranja
259     c_xhatact = [0.4940 0.1840 0.5560]; % púrpura
260     c_ref = [0.3 0.3 0.3];
261     c_open = [0.6 0.6 0.6];
262     c_errpred = [0.2 0.6 0.2]; % verde error pred
263     c_erract = [0.0 0.6 0.6]; % cian error act
264
265     fig = figure('Name', titulo, 'Color', 'w');
266     left = uipanel(fig, 'Position', [0.05 0.08 0.58 0.87]);
267     right = axes(fig, 'Position', [0.68 0.10 0.28 0.82]); % #ok<LAXES>
268     tl = tiledlayout(left, n+1, 1, 'TileSpacing', 'compact', 'Padding', 'compact');
269
270     % ===== Estados + errores (ejes duales) =====
271     for k = 1:n
272         ax = nexttile(tl); hold(ax, 'on');
273
274         % Izquierda: señales principales
275         yyaxis(ax, 'left');
276         hXp = plot(ax, t, X_pred(k,:), '-', 'LineWidth', 1.6, 'Color', c_pred, 'DisplayName', '$x_{\mathrm{pred}}$');
277         hXa = plot(ax, t, X_act(k,:), '-', 'LineWidth', 1.2, 'Color', c_act, 'DisplayName', '$x_{\mathrm{act}}$');
278         hXhp = stairs(ax, t, Xhat_pred(k,:), 'pentagram', 'LineWidth', 1.2, 'Color', c_xhatpred, 'DisplayName', '$\hat{x}_{\mathrm{pred}}$');

```

```

279     hXha = stairs(ax, t, Xhat_act(k,:), "pentagram", 'LineWidth', 1.2, 'Color',
c_xhatact, 'DisplayName', '$\hat{x}_{\mathrm{act}}$');
280     hR = stairs(ax, t, r, ':', 'LineWidth', 1.0, 'Color', c_ref, '
DisplayName', '$r$');
281     ylabel(ax, sprintf('x_%d',k));
282     grid(ax,'on'); grid(ax,'minor');
283
284     % Derecha: errores
285     yyaxis(ax,'right');
286     e_pred = X_pred(k,:) - Xhat_pred(k,:);
287     e_act = X_act(k,:) - Xhat_act(k,:);
288     hEp = stairs(ax, t, e_pred, '--', 'LineWidth', 1.0, 'Color', c_errpred, 'DisplayName',
'$e_{\mathrm{pred}}$');
289     hEa = stairs(ax, t, e_act, '--', 'LineWidth', 1.0, 'Color', c_erract, 'DisplayName',
'$e_{\mathrm{act}}$');
290     yline(ax, 0, ':', 'Color',[0.5 0.5 0.5], 'HandleVisibility','off');
291     ylabel(ax, sprintf('e_%d',k));
292
293     if k==1
294         title(ax, sprintf('%s -- Estados y estimaciones', titulo));
295     end
296     if k==n
297         % Leyenda con handles explícitos y bloqueada
298         yyaxis(ax,'left');
299         lg = legend(ax, [hXp, hXa, hXhp, hXha, hR, hEp, hEa], ...
300             {'$x_{\mathrm{pred}}$','$x_{\mathrm{act}}$', ...
301             '$\hat{x}_{\mathrm{pred}}$','$\hat{x}_{\mathrm{act}}$','$r$', ...
302             '$e_{\mathrm{pred}}$','$e_{\mathrm{act}}$'}, ...
303             'Location','best');
304         set(lg,'Interpreter','latex','AutoUpdate','off');
305     end
306 end
307
308 % ===== Control =====
309 axu = nexttile(tl); hold(axu,'on');
310 hUp = stairs(axu, t, U_pred, '-', 'LineWidth', 1.4, 'Color', c_pred, 'DisplayName', '$u_{\mathrm{pred}}$');
311 hUa = stairs(axu, t, U_act, '-', 'LineWidth', 1.2, 'Color', c_act, 'DisplayName', '$u_{\mathrm{act}}$');
312 hR = stairs(axu, t, r, ':', 'LineWidth', 1.0, 'Color', c_ref, 'DisplayName', '$r$');
313
314 grid(axu,'on'); grid(axu,'minor'); ylabel(axu,'u'); xlabel(axu,'Tiempo [s]');
315 title(axu,'Esfuerzo de control');
316 lg2 = legend(axu, [hUp, hUa, hR], {'$u_{\mathrm{pred}}$','$u_{\mathrm{act}}$','$r$'}, '
Location','best');
317 set(lg2,'Interpreter','latex','AutoUpdate','off');
318
319 % ===== Polos (sin legend) =====
320 axes(right); cla(right); hold(right,'on');
321 zgrid; axis(right,'equal'); axis(right,[-1.1 1.1 -1.1 1.1]); xlabel(right,'Re(z)');
322 ylabel(right,'Im(z)');
323 title(right, sprintf('Polos -- f_s = %.0f Hz', 1/Ts));
324
325 % Ejes en Re=0 e Im=0 (no ensucian leyenda)
326 xline(right, 0, '-', 'Color',[0.5 0.5 0.5],'LineWidth',0.8,'HandleVisibility','off');
327 yline(right, 0, '-', 'Color',[0.5 0.5 0.5],'LineWidth',0.8,'HandleVisibility','off');
328
329 plot(right, real(Popen), imag(Popen), 's', 'MarkerSize',7, 'LineWidth',1.2, '
Color', c_open);
330 plot(right, real(Pcl), imag(Pcl), 'x', 'MarkerSize',9, 'LineWidth',1.6, '
Color', c_pred);
331 plot(right, real(Pobs_pred), imag(Pobs_pred), 'o', 'MarkerSize',7, 'LineWidth',1.4, '
Color', c_xhatpred);
332 plot(right, real(Pobs_act), imag(Pobs_act), '^', 'MarkerSize',7, 'LineWidth',1.4, '
Color', c_xhatact);
333 end

```

Listing 17. HojaDeCalculos.m: cálculo de parámetros y generación de figuras.