

# PIDControl para Arduino

---

*Control PID, identificación de procesos, sintonización y filtrado*

Documentación de PIDControl v1.4.2

Ámbito. Este documento describe la API de la librería PIDControl v1.4.2 y, en capítulos separados, el sketch TCLab usado para validación en hardware. Los comandos de consola pertenecen al sketch de pruebas; la librería principal no depende de Serial.

19/09/2026

|                                                                  |    |
|------------------------------------------------------------------|----|
| 1. Introducción: qué resuelve PIDControl .....                   | 4  |
| 2. Arquitectura de la librería .....                             | 5  |
| Novedades de la versión 1.4.2 .....                              | 5  |
| 3. Terminología, magnitudes y unidades .....                     | 6  |
| 4. Construcción y ciclo básico del controlador .....             | 7  |
| 5. Estructuras PID, PI-D e I-PD .....                            | 8  |
| 6. Acción DIRECT y REVERSE .....                                 | 9  |
| 7. Modos MAN/AUTO, PV Tracking y transferencia bumpless .....    | 10 |
| Modo MAN .....                                                   | 10 |
| Modo AUTO .....                                                  | 10 |
| PV Tracking .....                                                | 10 |
| Transferencia bumpless .....                                     | 10 |
| 8. Saturación, anti-windup, límites y tiempo de muestreo .....   | 11 |
| Límites de salida .....                                          | 11 |
| Anti-windup .....                                                | 11 |
| Tiempo de muestreo .....                                         | 11 |
| 9. Acondicionamiento de señal y filtros .....                    | 12 |
| Media móvil .....                                                | 12 |
| Mediana .....                                                    | 12 |
| Paso bajo de primer orden .....                                  | 12 |
| Filtro complementario .....                                      | 12 |
| 10. Identificación en lazo abierto con StepTest .....            | 13 |
| Secuencia de trabajo .....                                       | 13 |
| Criterio robusto de estacionario .....                           | 13 |
| Respuesta direccional .....                                      | 13 |
| 11. Modelo FOPDT y ganancia normalizada .....                    | 14 |
| Ganancia en unidades de ingeniería .....                         | 14 |
| Estimación de $T_0$ y $T_p$ .....                                | 14 |
| Ganancia normalizada .....                                       | 14 |
| Objeto FOPDTModel .....                                          | 14 |
| 12. Sintonización IMC/Lambda y Tyreus-Luyben con PIDTuning ..... | 15 |
| IMC / Lambda PI .....                                            | 15 |
| Lambda PID .....                                                 | 15 |
| 13. Ensayo de relé con RelayTest .....                           | 16 |
| Configuración compacta en la API .....                           | 16 |
| Funcionamiento .....                                             | 16 |
| Cálculo de $K_u$ y $T_u$ .....                                   | 16 |

|                                                                       |    |
|-----------------------------------------------------------------------|----|
| Persistencia del resultado en memoria.....                            | 17 |
| Ejemplo real de validación en TCLab.....                              | 17 |
| Tyreus-Luyben integrado en PIDControl 1.4.2.....                      | 17 |
| 14. Estados, errores y comportamiento seguro de los ensayos.....      | 17 |
| StepTestState .....                                                   | 17 |
| StepTestError .....                                                   | 18 |
| RelayTestState .....                                                  | 18 |
| RelayTestError.....                                                   | 18 |
| 15. Sketch TCLab: adquisición, filtrado, seguridad y validación ..... | 20 |
| Cadena de adquisición .....                                           | 20 |
| Rango operativo y protección térmica.....                             | 20 |
| 16. Interfaz de comandos serie del sketch TCLab .....                 | 21 |
| Comandos generales.....                                               | 21 |
| Comandos StepTest.....                                                | 21 |
| Comandos RelayTest .....                                              | 22 |
| Formato compacto RelayTest .....                                      | 22 |
| Recuperación de resultados con IDENT .....                            | 23 |
| Trama para Arduino COM Plotter .....                                  | 23 |
| 17. Flujos de trabajo recomendados.....                               | 24 |
| Operación normal .....                                                | 24 |
| Identificación FOPDT + IMC/Lambda .....                               | 24 |
| Ensayo de relé .....                                                  | 24 |
| 18. Referencia completa de la API.....                                | 25 |
| PIDControl .....                                                      | 25 |
| StepTest.....                                                         | 25 |
| RelayTest.....                                                        | 26 |
| PIDTuning.....                                                        | 26 |
| PIDControlFilters .....                                               | 27 |
| 19. Limitaciones y buenas prácticas .....                             | 28 |
| 20. Estructura del paquete y preparación para repositorio.....        | 29 |
| Instalación.....                                                      | 29 |
| Apéndice A. Ejemplo mínimo completo .....                             | 30 |
| Apéndice B. Resumen rápido .....                                      | 31 |

# 1. Introducción: qué resuelve PIDControl

PIDControl es una librería Arduino orientada al control de procesos. Su objetivo no es limitarse a calcular una salida PID, sino cubrir una cadena completa: acondicionamiento de señal, control MAN/AUTO, identificación del proceso, sintonización y validación.

La librería está pensada con una filosofía próxima a la instrumentación industrial: separa PV, SP y OP; distingue acción directa e inversa; implementa seguimiento de PV, transferencia sin salto apreciable, anti-windup y varios formatos de controlador. Los ensayos de identificación son no bloqueantes y trabajan sobre variables ya escaladas, por lo que el hardware queda fuera del algoritmo.

- Controlador PID con estructuras PID, PI-D e I-PD.
- Modos MAN y AUTO.
- Acción DIRECT y REVERSE.
- PV Tracking y transferencia bumpless.
- Anti-windup por integración condicional.
- Límites de salida y tiempo de muestreo configurable.
- Filtros de media móvil, mediana, paso bajo y complementario.
- StepTest para identificación FOPDT en lazo abierto.
- Ganancia de proceso en unidades de ingeniería y ganancia normalizada.
- PIDTuning para sintonización IMC/Lambda PI, Lambda PID y Tyreus-Luyben.
- RelayTest para obtener  $K_u$  y  $T_u$  mediante ensayo de relé.

**Idea principal.** PIDControl intenta que el código de aplicación se limite a leer la PV, llamar a las clases de control/identificación y escribir la OP. La lógica de control no debe mezclarse con pines, ADC, PWM o comunicaciones.

## 2. Arquitectura de la librería

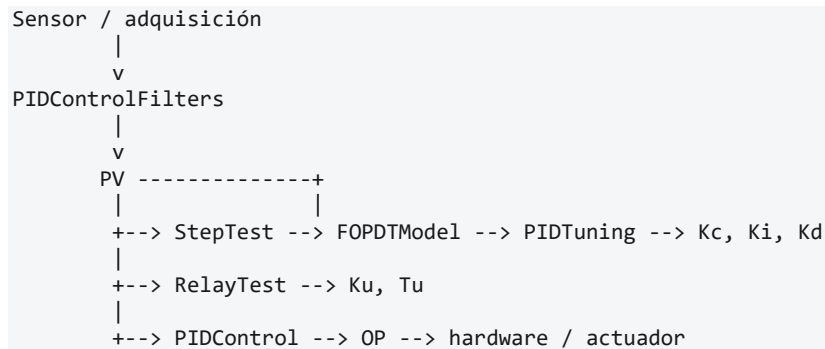
PIDControl v1.4.2 está dividido en cinco bloques principales. Esta separación permite utilizar únicamente los módulos necesarios en cada proyecto.

### Novedades de la versión 1.4.2

La versión 1.4.2 completa la cadena de identificación y sintonización. StepTest ofrece una sintonía PI IMC/Lambda y una PID Lambda, mientras que RelayTest puede transformar Ku/Tu en propuestas Tyreus-Luyben PI y PID. El sketch TCLab añade RELAY(...) compacto y el comando IDENT para recuperar resultados almacenados en memoria.

| Módulo            | Responsabilidad                                                                               | Cabecera            |
|-------------------|-----------------------------------------------------------------------------------------------|---------------------|
| PIDControl        | Control en lazo cerrado: PID, PI-D, I-PD, MAN/AUTO, acción, tracking, bumpless y anti-windup. | PIDControl.h        |
| StepTest          | Ensayo no bloqueante de respuesta a escalón e identificación FOPDT.                           | StepTest.h          |
| RelayTest         | Ensayo no bloqueante de relé para estimar Ku y Tu.                                            | RelayTest.h         |
| PIDTuning         | Sintonización IMC/Lambda PI y Lambda PID desde FOPDT; Tyreus-Luyben PI/PID desde Ku/Tu.       | PIDTuning.h         |
| PIDControlFilters | Filtros reutilizables de señal.                                                               | PIDControlFilters.h |

La relación típica entre los módulos puede representarse así:



StepTest y RelayTest reciben punteros a PV y OP. No conocen qué pin se utiliza ni cómo se escala una lectura. Esto permite usar la misma identificación con ADC/PWM, I2C, SPI, Modbus, un simulador o cualquier otra capa de hardware.

### 3. Terminología, magnitudes y unidades

| Símbolo | Significado                                     | Ejemplo                     |
|---------|-------------------------------------------------|-----------------------------|
| PV      | Process Variable. Variable de proceso medida.   | Temperatura T1 [°C]         |
| SP      | Setpoint. Consigna deseada.                     | 50 °C                       |
| OP      | Output. Salida del controlador.                 | 0..100 %                    |
| Kc      | Ganancia proporcional del controlador.          | 6.40                        |
| Ki      | Ganancia integral usada internamente [1/s].     | 0.0426 1/s                  |
| Kd      | Ganancia derivativa usada internamente [s].     | 0                           |
| Ti      | Tiempo integral.                                | 150 s                       |
| Td      | Tiempo derivativo.                              | 0 s                         |
| Kp      | Ganancia del proceso: $\Delta PV / \Delta OP$ . | °C/%OP                      |
| T0      | Tiempo muerto del proceso.                      | s                           |
| Tp      | Constante de tiempo del proceso.                | s                           |
| Ku      | Ganancia última estimada por ensayo de relé.    | adimensional según escalado |
| Tu      | Periodo último.                                 | s                           |

#### Conversión de parámetros de sintonía

$$K_i = K_c / T_i$$

$$K_d = K_c * T_d$$

PIDControl usa segundos en las ecuaciones del controlador. Por ello Ti, Td y el tiempo asociado a Ki/Kd deben ser coherentes con segundos.

**Kc no es Kp.** En esta documentación se reserva Kc para la ganancia del controlador y Kp para la ganancia del proceso identificado. Esta distinción evita una ambigüedad habitual en documentación PID.

## 4. Construcción y ciclo básico del controlador

El constructor recibe punteros a PV, OP y SP junto con los parámetros Kc, Ki y Kd. La declaración más corta utiliza los valores por defecto de la librería y suele ser suficiente para una configuración habitual.

```
PIDControl pid(&PV, &OP, &SP, Kc, Ki, Kd);

// Declaración completa
const bool pvTracking = true;

PIDControl pidCompleto(
    &PV, &OP, &SP,
    Kc, Ki, Kd,
    PIDType::PI_D,
    PIDAction::REVERSE,
    pvTracking
);
```

Con la declaración corta el controlador se crea en modo MAN, con estructura PI\_D, acción REVERSE, PV Tracking activado, límites de salida entre 0 y 100 % y un tiempo de muestreo de 100 ms. La declaración completa permite dejar estos parámetros explícitos o sustituirlos por otros valores.

```
void loop()
{
    PV = LeerVariableDeProceso();

    pid.Compute();

    EscribirSalida(OP);
}
```

Compute() devuelve true cuando se ha realizado un nuevo cálculo automático. En AUTO respeta el tiempo de muestreo configurado. En MAN aplica la salida manual y, si PV Tracking está activo, actualiza el SP interno para que siga a la PV.

**Importante.** Conviene llamar a Compute() también en MAN. En PIDControl el modo manual no significa que la clase deje de participar: mantiene OP manual y, si procede, el seguimiento de PV.

## 5. Estructuras PID, PI-D e I-PD

PIDControl permite trabajar con tres estructuras de controlador mediante PIDType. La estructura utilizada por defecto es PI\_D.

| Estructura | P actúa sobre | I actúa sobre | D actúa sobre | Uso conceptual                                        |
|------------|---------------|---------------|---------------|-------------------------------------------------------|
| PID        | Error         | Error         | Error         | PID clásico.                                          |
| PI_D       | Error         | Error         | PV            | Evita derivative kick ante cambios de SP.             |
| I_PD       | PV            | Error         | PV            | Reduce el efecto inmediato de cambios de SP en P y D. |

Para acción REVERSE, con  $e = SP - PV$ , las ecuaciones implementadas son:

```
PID:
P = Kc * e
I(k) = I(k-1) + Ki * e * Ts
D = Kd * (e - eAnterior) / Ts

PI-D:
P = Kc * e
I(k) = I(k-1) + Ki * e * Ts
D = -Kd * (PV - PVAnterior) / Ts

I-PD:
P = -Kc * PV
I(k) = I(k-1) + Ki * e * Ts
D = -Kd * (PV - PVAnterior) / Ts
```

La estructura se selecciona en cualquier momento con:

```
pid.SetStructure(PIDType::PID);
pid.SetStructure(PIDType::PI_D);
pid.SetStructure(PIDType::I_PD);
```

Si la estructura se cambia mientras el controlador está en AUTO, PIDControl recalcula el término integral mediante InitializeBumpless() para minimizar el salto de OP.

## 6. Acción DIRECT y REVERSE

Antes de poner un controlador en automático debemos comprobar que la acción de control es correcta. La forma más sencilla de elegirla consiste en preguntarnos qué debe hacer la OP cuando aumenta la PV. Si al aumentar la PV la OP debe disminuir utilizaremos REVERSE. Si al aumentar la PV también debe aumentar la OP utilizaremos DIRECT. La acción utilizada por defecto en PIDControl es REVERSE.

| Acción  | Comportamiento típico        | Ejemplo                                                          |
|---------|------------------------------|------------------------------------------------------------------|
| REVERSE | Si aumenta PV, OP disminuye. | Calentamiento: si aumenta la temperatura, se reduce la potencia. |
| DIRECT  | Si aumenta PV, OP aumenta.   | Procesos en los que PV y OP deben variar en el mismo sentido.    |

```
pid.SetDirection(PIDAction::REVERSE);
pid.SetDirection(PIDAction::DIRECT);
```

TCLab. En el proceso térmico utilizado en los ejemplos, si aumenta la temperatura queremos reducir la potencia del calefactor. Por ello se utiliza acción REVERSE.

## 7. Modos MAN/AUTO, PV Tracking y transferencia bumpless

### Modo MAN

En MAN, la salida se fija mediante SetManualOutput().

```
pid.SetMode(PIDMode::MAN);
pid.SetManualOutput(40.0);
```

### Modo AUTO

En AUTO, la salida es calculada por el controlador.

```
pid.SetMode(PIDMode::AUTO);
```

### PV Tracking

El PV Tracking está activado por defecto. Mientras el controlador permanece en MAN, el SP interno sigue a la PV y la consigna solicitada por el operador se conserva por separado en operatorSetpoint. Esto facilita el paso de MAN a AUTO y permite preparar una transferencia sin un salto brusco de salida.

```
pid.SetPVTracking(true);
pid.SetSetpoint(50.0);    // se almacena como consigna del operador

// En MAN + tracking, el SP activo sigue a PV.
// La consigna 50.0 sigue disponible en GetOperatorSetpoint().
```

### Transferencia bumpless

Al pasar a AUTO, cambiar Kc/Ki/Kd, modificar la estructura, invertir la acción o cambiar límites de salida mientras se está en AUTO, PIDControl inicializa la integral de forma que P + I + D parta aproximadamente de la OP actual.

#### Inicialización conceptual

```
I = OP_actual - P
```

La contribución derivativa se inicializa a cero porque previousPV y previousError se actualizan con los valores presentes.

## 8. Saturación, anti-windup, límites y tiempo de muestreo

### Límites de salida

```
pid.SetOutputLimits(0.0, 100.0);
```

La OP calculada y la OP manual se limitan al intervalo configurado. Los límites por defecto son 0..100 %. Si se cambian los límites en AUTO, se realiza una nueva inicialización bumpless.

### Anti-windup

PIDControl utiliza integración condicional. Antes de aceptar el nuevo valor integral calcula una OP sin saturar. Si la salida está por encima del máximo y la integral pretende aumentarla todavía más, no integra. Lo mismo ocurre en el límite inferior.

```
integralIncrement = sign * Ki * error * Ts;
newIntegral       = integral + integralIncrement;
unsaturatedOutput = P + newIntegral + D;

si OP_sin_limitar > max y integralIncrement > 0 -> no integrar
si OP_sin_limitar < min y integralIncrement < 0 -> no integrar
```

### Tiempo de muestreo

```
pid.SetSampleTime(100); // ms
```

El tiempo de muestreo por defecto es 100 ms. Compute() solo ejecuta un nuevo cálculo automático cuando ha transcurrido al menos el periodo configurado. La ecuación utiliza el tiempo real transcurrido, convertido a segundos, y no una constante Ts fija asumida.

## 9. Acondicionamiento de señal y filtros

PIDControlFilters es independiente del controlador y permite tratar la señal antes de entregarla al PID o a los ensayos de identificación. El objetivo no es conseguir la curva más bonita posible, sino eliminar suficiente ruido sin ocultar la dinámica real del proceso.

### Media móvil

```
MovingAverageFilter<10> filtro;
double pvFiltrada = filtro.Update(pvRaw);
```

### Ecuación

```
y[k] = (1/N) * sum(x[k-i]), i=0..N-1
```

Reduce ruido aleatorio, a cambio de introducir retardo. Durante el arranque promedia únicamente las muestras realmente recibidas hasta completar N.

### Mediana

```
MedianFilter<5> filtro;
double pvFiltrada = filtro.Update(pvRaw);
```

Es especialmente eficaz contra picos aislados y valores espurios. La implementación admite ventanas pares e impares.

### Paso bajo de primer orden

```
LowPassFilter filtro(1.0); // tau = 1 s
double pvFiltrada = filtro.Update(pvRaw, dtSeconds);
```

### Coeficiente discreto exacto

```
alpha = 1 - exp(-dt/tau)
y[k] = y[k-1] + alpha * (x[k] - y[k-1])
```

El parámetro público es la constante de tiempo física tau, no un factor arbitrario de suavizado.

### Filtro complementario

```
ComplementaryFilter filtro(0.98);

double fusion = filtro.Update(estimacionRapida, referenciaLenta);
double angulo = filtro.UpdateIntegrated(velocidadGiro, anguloAbsoluto, dt);
```

El filtro complementario necesita dos estimaciones de la misma magnitud con propiedades complementarias. No debe interpretarse como un simple paso bajo para una única señal.

Filtrado e identificación. Todo filtro añade dinámica. Un filtrado excesivo puede hacer que durante una identificación el proceso parezca tener un tiempo muerto o una constante de tiempo mayores de los reales. Conviene filtrar solo lo necesario.

## 10. Identificación en lazo abierto con StepTest

StepTest ejecuta un ensayo de respuesta a escalón sin bloquear el loop de Arduino. Su objetivo es estimar un modelo FOPDT a partir de uno o varios cambios de OP.

```
StepTest test(pid, &PV, &OP);

test.SetStepSize(10.0);
test.SetNumberOfSteps(2);
test.SetFirstStepUp(true);
test.SetProcessGain(ProcessGain::POSITIVE);

test.SetStabilityCriteria(0.30, 60.0, 0.10);
test.SetResponseThreshold(0.40);
test.SetMaxTestTime(30.0);

test.Start();
```

Durante el loop:

```
test.Update();
pid.Compute();
```

### Secuencia de trabajo

- Fuerza el controlador a MAN.
- Guarda la OP actual como bias.
- Espera un estado estable inicial.
- Aplica los saltos configurados de forma alterna.
- Comprueba que la PV responde en la dirección esperada.
- Espera el nuevo estacionario.
- Calcula un modelo para cada salto válido.
- Promedia los modelos válidos.
- Al terminar o abortar devuelve la OP al bias original y deja el PID en MAN.

### Criterio robusto de estacionario

#### Condiciones simultáneas

```
PVmax - PVmin <= 2 * band
abs(pendiente_regresion) <= maxSlope
```

La pendiente se obtiene por regresión lineal de mínimos cuadrados durante la ventana de estabilidad y se expresa en unidades de PV por minuto.

### Respuesta direccional

ProcessGain::POSITIVE indica que una subida de OP debe terminar elevando PV. NEGATIVE indica lo contrario. El criterio evita que una inercia residual en sentido incorrecto se confunda con la respuesta al escalón.

## 11. Modelo FOPDT y ganancia normalizada

StepTest estima un modelo de primer orden con tiempo muerto:

### Modelo

$$G(s) = K_p * \exp(-T_0*s) / (T_p*s + 1)$$

### Ganancia en unidades de ingeniería

#### Ganancia del proceso

$$K_p = \Delta PV / \Delta OP$$

### Estimación de $T_0$ y $T_p$

La implementación utiliza los cruces del 28,3 % y 63,2 % de la respuesta:

#### Método 28,3/63,2

$$\begin{aligned} T_p &= 1.5 * (t_{63.2} - t_{28.3}) \\ T_0 &= 1.5 * t_{28.3} - 0.5 * t_{63.2} \end{aligned}$$

### Ganancia normalizada

Los límites de seguridad del ensayo y los rangos completos de ingeniería son conceptos distintos. Los rangos solo intervienen en la normalización de la ganancia.

```
test.SetOutputLimits(0.0, 60.0); // seguridad del ensayo
test.SetPVLimits(15.0, 45.0);    // seguridad del ensayo

test.SetOutputRange(0.0, 100.0); // rango completo de OP
test.SetPVRRange(0.0, 80.0);     // rango completo de PV
```

#### Ganancia normalizada

$$\begin{aligned} K_{p\text{Normalized}} &= (\Delta PV / PV_{\text{span}}) / (\Delta OP / OP_{\text{span}}) \\ &= K_p * OP_{\text{span}} / PV_{\text{span}} \end{aligned}$$

$K_{p\text{Normalized}}$  se expresa como %PV/%OP y solo se informa cuando ambos rangos de ingeniería están configurados.

### Objeto FOPDTModel

```
FOPDTModel model = test.GetModel();

model.valid
model.Kp
model.KpNormalizedValid
model.KpNormalized
model.T0
model.Tp
model.pvRangeMin
model.pvRangeMax
model.outputRangeMin
model.outputRangeMax
```

Este objeto desacopla identificación y sintonización: puede entregarse directamente a PIDTuning.

El resultado de StepTest también permanece en el objeto tras finalizar. En el sketch TCLab v1.4.2 puede recuperarse con IDENT STEP mientras no se ejecute RESET, no se reinicie la placa y no se inicie otro StepTest que sobrescriba el estado.

## 12. Sintonización IMC/Lambda y Tyreus-Luyben con PIDTuning

PIDTuning convierte resultados de identificación en parámetros de controlador sin aplicarlos automáticamente. Con un modelo FOPDT ofrece una sintonía IMC/Lambda PI y una sintonía Lambda PID. A partir de  $K_u$  y  $T_u$  obtenidos por RelayTest ofrece Tyreus-Luyben PI y PID.

```
PIDTuning tuning(model);

PIDTuningResult pi = tuning.IMC_PI();
PIDTuningResult pid = tuning.Lambda_PID();

PIDTuningResult piRobust = tuning.IMC_PI(IMCSpeed::ROBUST);
PIDTuningResult pidRobust = tuning.Lambda_PID(IMCSpeed::ROBUST);
```

### IMC / Lambda PI

```
Kc = Tp / (abs(Kp) * (Tf + T0))
Ti = Tp
Td = 0
Ki = Kc / Ti
Kd = 0
```

| Preset               | Tf                  |
|----------------------|---------------------|
| IMCSpeed::AGGRESSIVE | $T_f = 1 \cdot T_0$ |
| IMCSpeed::NORMAL     | $T_f = 2 \cdot T_0$ |
| IMCSpeed::ROBUST     | $T_f = 3 \cdot T_0$ |

Las sobrecargas sin parámetros usan el preset NORMAL, por lo que  $T_f = 2 \cdot T_0$  tanto para PI como para PID. Si  $T_0$  es numéricamente cero, PIDControl utiliza  $0,1 \cdot T_p$  como valor de respaldo. AGGRESSIVE, NORMAL y ROBUST son presets prácticos de la librería y no reglas universales del método.

### Lambda PID

La regla Rivera/Lambda PID utiliza el mismo modelo FOPDT y un parámetro  $T_f$  que fija la velocidad deseada del lazo cerrado:

```
Kc = (Tp + T0/2) / (abs(Kp) * (Tf + T0/2))
Ti = Tp + T0/2
Td = Tp*T0 / (2*Tp + T0)
Ki = Kc / Ti
Kd = Kc * Td
```

En PIDControl, `Lambda_PID()` devuelve  $K_c$ ,  $T_i$ ,  $T_d$ ,  $K_i$  y  $K_d$ . El campo `lambda` de `PIDTuningResult` conserva ese nombre por compatibilidad con la API, pero en esta documentación representa el valor  $T_f$  utilizado para calcular la sintonía.

```
PIDTuningResult pid = tuning.Lambda_PID();
PIDTuningResult pid60 = tuning.Lambda_PID(60.0);
PIDTuningResult pidRobust = tuning.Lambda_PID(IMCSpeed::ROBUST);
```

**Signo de  $K_p$ .** PIDTuning utiliza  $\text{abs}(K_p)$  para calcular una  $K_c$  positiva. La dirección del controlador se gestiona de forma independiente mediante `PIDAction`.

La aplicación de la sintonía es explícita:

```
PIDTuning::Apply(pid, pi);
```

`Apply()` solo cambia  $K_c$ ,  $K_i$  y  $K_d$ . No modifica estructura, modo, acción ni SP.

## 13. Ensayo de relé con RelayTest

RelayTest implementa un ensayo no bloqueante inspirado en el principio de autotuning por relé de Åström-Hägglund. Su función es identificar  $K_u$  y  $T_u$ ; la transformación de esos valores en parámetros de control se realiza después mediante PIDTuning.

```
RelayTest relay(pid, &PV, &OP);

relay.SetRelayAmplitude(10.0);
relay.SetHysteresis(0.5);
relay.SetCycles(3);
relay.UseCurrentPVAsReference();
relay.SetMaxTestTime(15.0);
relay.SetOutputLimits(20.0, 60.0);
relay.SetPVLimits(15.0, 48.0);

relay.Start();
```

### Configuración compacta en la API

Desde v1.4.1 la propia clase RelayTest dispone de RelayTestConfig y Start(config):

```
RelayTestConfig cfg;
cfg.amplitude = 10.0;
cfg.hysteresis = 0.5;
cfg.cycles = 3;
cfg.useCurrentReference = true;
cfg.maxTestTimeMinutes = 15.0;
cfg.outputMin = 20.0;
cfg.outputMax = 60.0;
cfg.pvMin = 15.0;
cfg.pvMax = 48.0;

relay.Start(cfg);
```

**Diferencia importante.** RelayTestConfig forma parte de la librería. La sintaxis textual RELAY(...) sería una responsabilidad del parser de un sketch concreto; PIDControl no interpreta directamente comandos de Serial.

## Funcionamiento

### Niveles del relé

```
OPhigh = bias + d
OPlow = bias - d
```

El bias es la OP existente al iniciar. Por defecto la referencia del relé es la PV existente en Start(). Para acción REVERSE, asociada normalmente a ganancia positiva, la lógica es:

```
PV >= referencia + histéresis -> OP low
PV <= referencia - histéresis -> OP high
```

En DIRECT el sentido de conmutación se invierte.

### Cálculo de $K_u$ y $T_u$

#### Aproximación de relé simétrico ideal

```
a = (PVmax - PVmin) / 2
Ku = 4*d / (pi*a)
Tu = periodo medio entre transiciones equivalentes
```

La primera oscilación completa se descarta como transitorio de arranque. El número configurado con SetCycles() corresponde a los ciclos posteriores usados en las medias.

Persistencia del resultado en memoria

GetResult() puede consultarse de nuevo después de finalizar el ensayo. En el sketch TCLab v1.4.2, IDENT RELAY vuelve a imprimir  $K_u$ ,  $T_u$  y las propuestas Tyreus-Luyben mientras el resultado continúe almacenado en memoria y no haya sido reseteado o sobrescrito por otro ensayo de relé.

Sintonización integrada en v1.4.2. RelayTest continúa limitándose a identificar  $K_u$  y  $T_u$ . PIDTuning añade Tyreus-Luyben PI y PID como etapa separada. La librería no aplica automáticamente ninguna sintonización: el usuario decide cuál utilizar.

## Ejemplo real de validación en TCLab

En la validación realizada sobre TCLab, el ensayo se ejecutó desde un bias de OP=40 %, con amplitud de relé de 10 %, histéresis de 0,5 °C y tres ciclos válidos. La OP osciló entre 30 % y 50 % y el ensayo terminó devolviendo el bias original en MAN.

```
Ku = 14.233237
Tu = 180.300 s
Amplitud PV = 0.903066 °C
Amplitud del relé = 10 %
Ciclos válidos = 3
```

Como comprobación externa a la librería, estos valores pueden utilizarse para calcular una sintonía Ziegler-Nichols. Para PI, usando  $K_c = 0,45 \cdot K_u$  y  $T_i = T_u/1,2$ , se obtiene aproximadamente:

```
Kc = 6.404957
Ti = 150.250 s
Ki = Kc/Ti = 0.042629 1/s
Kd = 0
```

## Tyreus-Luyben integrado en PIDControl 1.4.2

A partir del mismo  $K_u$  y  $T_u$ , PIDTuning ofrece directamente dos alternativas menos agresivas que el Ziegler-Nichols clásico:

```
Tyreus-Luyben PI:
Kc = Ku / 3.2
Ti = 2.2*Tu
Ki = Kc/Ti

Tyreus-Luyben PID:
Kc = Ku / 3.3
Ti = 2.2*Tu
Td = Tu / 6.3
Ki = Kc/Ti
Kd = Kc*Td
```

Con el ensayo real  $K_u=14,233237$  y  $T_u=180,300$  s se obtiene aproximadamente:

```
Tyreus-Luyben PI: Kc=4.447887 Ti=396.660 s Ki=0.011213 Kd=0
Tyreus-Luyben PID: Kc=4.313102 Ti=396.660 s Td=28.619 s Ki=0.010874 Kd=123.437
```

La aplicación sigue siendo explícita. PIDTuning::Apply() o SetTunings() modifican la sintonía solo cuando el usuario lo solicita.

# 14. Estados, errores y comportamiento seguro de los ensayos

## StepTestState

| Estado | Significado        |
|--------|--------------------|
| IDLE   | Sin ensayo activo. |

| Estado               | Significado                                        |
|----------------------|----------------------------------------------------|
| WAITING_STABLE       | Esperando estacionario inicial o de transición.    |
| RUNNING_STEP         | Escalón aplicado y esperando respuesta.            |
| WAITING_FINAL_STABLE | Respuesta detectada; esperando estacionario final. |
| FINISHED             | Identificación terminada.                          |
| ABORTED              | Ensayo abortado.                                   |

## StepTestError

| Error            | Motivo                                                   |
|------------------|----------------------------------------------------------|
| NONE             | Sin error.                                               |
| INVALID_CONFIG   | Configuración no válida.                                 |
| PV_LOW / PV_HIGH | PV fuera de los límites del ensayo.                      |
| OP_LIMIT         | OP requerida fuera de los límites del ensayo.            |
| TIMEOUT          | Tiempo máximo excedido.                                  |
| NO_RESPONSE      | No se detectó una respuesta válida.                      |
| INVALID_MODEL    | Hubo respuesta, pero no pudo obtenerse un modelo válido. |
| USER_ABORT       | Abortado por la aplicación/usuario.                      |

## RelayTestState

| Estado   | Significado        |
|----------|--------------------|
| IDLE     | Sin ensayo activo. |
| RUNNING  | Relé en ejecución. |
| FINISHED | Ku/Tu calculados.  |
| ABORTED  | Ensayo abortado.   |

## RelayTestError

| Error            | Motivo                                                   |
|------------------|----------------------------------------------------------|
| NONE             | Sin error.                                               |
| INVALID_CONFIG   | Parámetros incompatibles o referencia fuera de rango.    |
| PV_LOW / PV_HIGH | PV fuera de límites.                                     |
| OP_LIMIT         | Bias u OP high/low fuera de límites.                     |
| TIMEOUT          | Hay ciclos pero no se completan los requeridos a tiempo. |

| Error          | Motivo                                 |
|----------------|----------------------------------------|
| NO_OSCILLATION | Expira el tiempo sin completar ciclos. |
| USER_ABORT     | Abortado por la aplicación/usuario.    |

Tanto StepTest como RelayTest devuelven el controlador a MAN y restauran el bias original al finalizar o abortar. Reset() limpia estado, errores y resultados, conservando la configuración del objeto. El RESET global del sketch TCLab, en cambio, vuelve a aplicar explícitamente sus valores por defecto.

## 15. Sketch TCLab: adquisición, filtrado, seguridad y validación

La librería se ha validado sobre una TCLab con Arduino UNO R4 WiFi. Este sketch no forma parte del núcleo algorítmico: es una aplicación de prueba que integra hardware, seguridad, consola y representación gráfica.

| Elemento        | Configuración                                         |
|-----------------|-------------------------------------------------------|
| T1              | A0                                                    |
| T2              | A2                                                    |
| Q1              | D3                                                    |
| Q2              | D5                                                    |
| Serie           | 115200 baud                                           |
| ADC             | 14 bits, referencia externa 3,300 V                   |
| Q1 100 %        | PWM máximo 200/255                                    |
| Q2              | PWM máximo 100/255, mantenido a 0 en el ensayo actual |
| Periodo I/O     | 100 ms                                                |
| Periodo de plot | 500 ms                                                |

### Cadena de adquisición

```

ADC
-> descarte de primera conversión tras cambiar de canal
-> promedio de 10 conversiones
-> MedianFilter<5>
-> LowPassFilter(tau)
-> PV utilizada por PID / StepTest / RelayTest

```

La conversión del sensor sigue la relación TMP36:

#### Temperatura

$$T [C] = (V - 0.5) * 100$$

### Rango operativo y protección térmica

| Concepto                | Valor actual                                |
|-------------------------|---------------------------------------------|
| SP de operación normal  | 10.80 °C                                    |
| Trip por PV/T2 filtrada | >= 80 °C                                    |
| Trip por PV/T2 raw      | >= 82 °C                                    |
| Liberación de alarma    | < 78 °C en T1 y T2                          |
| Acción tras trip        | MAN, OP=0; no vuelve automáticamente a AUTO |

La protección térmica de TCLab es independiente de los límites de StepTest y RelayTest. También se aplica un clamp final de OP antes de escribir el hardware.

## 16. Interfaz de comandos serie del sketch TCLab

**No confundir API y consola.** Los siguientes comandos no pertenecen al núcleo de PIDControl. Son una interfaz textual implementada por el sketch TCLab para probar la librería desde Serial/Arduino COM Plotter.

### Comandos generales

| Comando               | Función                                                               | Ejemplo                  |
|-----------------------|-----------------------------------------------------------------------|--------------------------|
| HELP                  | Muestra ayuda.                                                        | HELP                     |
| STATUS                | Muestra estado de PID, filtros y ensayos.                             | STATUS                   |
| RESET                 | MAN, OP=0 y restaura defaults del sketch.                             | RESET                    |
| STOP                  | Aborta ensayos, MAN y OP=0.                                           | STOP                     |
| MAN [op]              | Pasa a MAN y opcionalmente fija OP.                                   | MAN 40                   |
| AUTO [sp]             | Pasa a AUTO y opcionalmente fija SP.                                  | AUTO 50                  |
| SP <valor>            | Actualiza SP de operador, limitado a 10..80 °C.                       | SP 50                    |
| OP <valor>            | Actualiza OP manual.                                                  | OP 35                    |
| TUNE <Kc> <Ki> <Kd>   | Actualiza sintonía.                                                   | TUNE 6.404957 0.042629 0 |
| STRUCT PID PI_D I_PD  | Selecciona estructura.                                                | STRUCT PI_D              |
| ACTION DIRECT REVERSE | Selecciona acción.                                                    | ACTION REVERSE           |
| TRACK ON OFF          | Activa/desactiva PV Tracking.                                         | TRACK ON                 |
| OUTLIMITS <min> <max> | Límites PID de salida.                                                | OUTLIMITS 0 100          |
| TS <ms>               | Tiempo de muestreo.                                                   | TS 100                   |
| FILTER <tau>          | Mediana(5) + LPF con tau indicado.                                    | FILTER 1.0               |
| FILTER ON / OFF       | Activa LPF default o lo desactiva conservando mediana.                | FILTER OFF               |
| IDENT [STEP RELAY]    | Vuelve a mostrar resultados de identificación conservados en memoria. | IDENT                    |

### Comandos StepTest

| Comando                 | Función                                |
|-------------------------|----------------------------------------|
| STEP START / STEP ABORT | Inicia o aborta el ensayo configurado. |
| STEP SIZE <percent>     | Tamaño del salto de OP.                |
| STEP N <1..10>          | Número de saltos.                      |
| STEP DIR UP DOWN        | Dirección del primer salto.            |

| Comando                                                        | Función                                             |
|----------------------------------------------------------------|-----------------------------------------------------|
| STEP GAIN POS NEG                                              | Signo esperado de la ganancia del proceso.          |
| STEP STABLE <band> <time> [slope]                              | Criterio de estacionario.                           |
| STEP RESPONSE <deltaPV>                                        | Umbral direccional mínimo de respuesta.             |
| STEP TIME <minutes>                                            | Tiempo máximo.                                      |
| STEP LIMITS <opMin> <opMax> <pvMin> <pvMax>                    | Límites de seguridad del ensayo.                    |
| STEP RANGE <pvRangeMin> <pvRangeMax> <opRangeMin> <opRangeMax> | Rangos completos de ingeniería para Kp normalizado. |

Formato compacto estándar:

```
STEP(size,n,UP|DOWN,POS|NEG,stableBand,stableTime,maxSlope,response,maxMin,opMin,opMax,pvMin,pvMax)
```

Con rangos de ingeniería opcionales:

```
STEP(...,pvMin,pvMax,pvRangeMin,pvRangeMax,opRangeMin,opRangeMax)
```

Ejemplo validado:

```
STEP(10,2,UP,POS,0.3,60,0.10,0.4,30,0,60,15,45,0,80,0,100)
```

## Comandos RelayTest

| Comando                                                           | Función                                          |
|-------------------------------------------------------------------|--------------------------------------------------|
| RELAY START / RELAY ABORT                                         | Inicia o aborta el ensayo.                       |
| RELAY AMP <percent>                                               | Amplitud d alrededor del bias.                   |
| RELAY HYST <degC>                                                 | Histéresis de conmutación.                       |
| RELAY CYCLES <n>                                                  | Número de ciclos válidos a promediar.            |
| RELAY REF CURRENT                                                 | Usa PV actual como referencia al iniciar.        |
| RELAY REF <degC>                                                  | Usa una referencia fija.                         |
| RELAY TIME <minutes>                                              | Tiempo máximo.                                   |
| RELAY LIMITS <opMin> <opMax> <pvMin> <pvMax>                      | Límites de seguridad.                            |
| RELAY(amp,hyst,cycles,CURRENT ref,maxMin,opMin,opMax,pvMin,pvMax) | Configura y arranca RelayTest en una sola orden. |

Formato compacto en v1.4.2. El sketch TCLab interpreta RELAY(...) y convierte sus parámetros en RelayTestConfig. La lógica del ensayo, validación y cálculo continúa dentro de PIDControl. STEP(...) mantiene su formato compacto existente.

## Formato compacto RelayTest

El formato compacto validado en TCLab es:

```
RELAY(10,0.5,3,CURRENT,15,20,60,15,48)
```

Equivala a amplitud  $\pm 10$  %OP, histéresis  $\pm 0,5$  °C, tres ciclos válidos, referencia igual a la PV actual, timeout de 15 min, OP permitida 20..60 % y PV permitida 15..48 °C.

## Recuperación de resultados con IDENT

Después de terminar un StepTest o RelayTest, el resultado calculado permanece almacenado en memoria. Si se limpia o pierde el log del puerto serie puede solicitarse de nuevo sin repetir el ensayo:

```
IDENT
IDENT STEP
IDENT RELAY
```

IDENT muestra  $K_p$ ,  $T_0$  y  $T_p$  del StepTest almacenado, o  $K_u$  y  $T_u$  del RelayTest almacenado. También vuelve a mostrar las propuestas de sintonización disponibles. La memoria es volátil: RESET, un reset/pérdida de alimentación de la placa o un nuevo ensayo del mismo tipo eliminan o sobrescriben esos datos.

## Trama para Arduino COM Plotter

```
#SP PV OP T2
```

La trama empieza por '#', utiliza espacio como separador y contiene cuatro variables.

## 17. Flujos de trabajo recomendados

### Operación normal

```
pid.SetOutputLimits(0.0, 100.0);
pid.SetSampleTime(100);
pid.SetStructure(PIDType::PI_D);
pid.SetDirection(PIDAction::REVERSE);
pid.SetPVTracking(true);

pid.SetManualOutput(40.0);
pid.SetMode(PIDMode::MAN);

// Cuando proceda:
pid.SetMode(PIDMode::AUTO);
pid.SetSetpoint(50.0);
```

### Identificación FOPDT + IMC/Lambda

1. Estabilizar el proceso en MAN.
2. Configurar límites y rangos de StepTest.
3. Ejecutar test.Start() y llamar a test.Update() en loop().
4. Obtener FOPDTModel model = test.GetModel().
5. Calcular una propuesta IMC/Lambda PI y otra Lambda PID.
6. Revisar Kp, T0, Tp y la coherencia del ensayo.
7. Aplicar explícitamente la sintonía elegida.
8. Pasar de MAN a AUTO con PV Tracking.
9. Si se pierde el log del sketch TCLab, usar IDENT STEP mientras el resultado siga en memoria.

### Ensayo de relé

1. Estabilizar el proceso en MAN con un bias adecuado.
2. Elegir amplitud e histéresis.
3. Verificar que bias+d y bias-d quedan dentro de límites.
4. Ejecutar relay.Start(config) o el comando compacto RELAY(...).
5. Llamar a relay.Update() y pid.Compute() en loop().
6. Obtener Ku y Tu.
7. Calcular Tyreus-Luyben PI y PID mediante PIDTuning.
8. El ensayo restaura el bias y permanece en MAN.
9. Si se pierde el log del sketch TCLab, usar IDENT RELAY mientras el resultado siga en memoria.

**Seguridad.** Los ensayos de identificación deben tener límites propios más conservadores que los límites físicos absolutos de la planta o maqueta. Además, una protección independiente de hardware/aplicación debe seguir activa.

## 18. Referencia completa de la API

### PIDControl

| Método                              | Descripción                                                                 |
|-------------------------------------|-----------------------------------------------------------------------------|
| PIDControl(...)                     | Constructor con punteros PV/OP/SP, Kc/Ki/Kd, estructura, acción y tracking. |
| Compute()                           | Ejecuta MAN/tracking o cálculo AUTO cuando corresponde.                     |
| SetMode()/GetMode()                 | Cambia/consulta MAN o AUTO.                                                 |
| SetManualOutput()/GetManualOutput() | Fija/consulta OP manual.                                                    |
| SetTunings()                        | Fija Kc, Ki y Kd; ignora parámetros negativos.                              |
| GetKc()/GetKi()/GetKd()             | Devuelve parámetros.                                                        |
| SetStructure()/GetStructure()       | PID, PI_D o I_PD.                                                           |
| SetDirection()/GetDirection()       | DIRECT o REVERSE.                                                           |
| SetPVTracking()/GetPVTracking()     | Activa/consulta tracking.                                                   |
| SetSetpoint()/GetOperatorSetpoint() | Actualiza/consulta consigna del operador.                                   |
| SetOutputLimits()                   | Configura límites de OP.                                                    |
| GetOutputMinimum()/Maximum()        | Consulta límites.                                                           |
| SetSampleTime()/GetSampleTime()     | Configura/consulta periodo de cálculo en ms.                                |

### StepTest

| Método                            | Descripción                                 |
|-----------------------------------|---------------------------------------------|
| SetOutputLimits()                 | Límites OP durante el test.                 |
| SetPVLimits()                     | Límites PV durante el test.                 |
| SetOutputRange()/SetPVRange()     | Rangos completos para normalización.        |
| HasEngineeringRanges()            | Indica si ambos rangos están configurados.  |
| SetStepSize()                     | Tamaño del escalón.                         |
| SetNumberOfSteps()                | Número de escalones, máximo por defecto 10. |
| SetFirstStepUp()                  | Dirección inicial.                          |
| SetProcessGain()                  | POSITIVE/NEGATIVE.                          |
| SetStabilityCriteria()            | Banda, tiempo y pendiente máxima opcional.  |
| SetMaxStabilitySlope()            | Pendiente máxima.                           |
| SetResponseThreshold()            | Umbral de respuesta.                        |
| SetMaxTestTime()/GetMaxTestTime() | Timeout en minutos.                         |
| Start()/Update()/Abort()/Reset()  | Ciclo de vida.                              |

| Método                                   | Descripción                             |
|------------------------------------------|-----------------------------------------|
| GetState()/GetError()                    | Estado y error.                         |
| GetResult()                              | Resultado agregado.                     |
| GetModel()                               | FOPDTModel transferible.                |
| GetStepResult()                          | Resultado individual de un escalón.     |
| GetCurrentStabilitySlope()               | Pendiente actual de la ventana estable. |
| GetBiasOutput()/GetCurrentTargetOutput() | Bias y target OP.                       |
| GetCurrentStepIndex()                    | Índice del escalón actual.              |

## RelayTest

| Método                               | Descripción                                       |
|--------------------------------------|---------------------------------------------------|
| SetOutputLimits()/SetPVLimits()      | Límites de seguridad del ensayo.                  |
| SetRelayAmplitude()                  | Amplitud d.                                       |
| SetHysteresis()                      | Histéresis en PV.                                 |
| SetCycles()                          | Ciclos válidos a promediar.                       |
| SetReference()                       | Referencia fija.                                  |
| UseCurrentPVAsReference()            | Referencia = PV al iniciar.                       |
| SetMaxTestTime()/GetMaxTestTime()    | Timeout en minutos.                               |
| Start()                              | Arranca con configuración previamente almacenada. |
| Start(const RelayTestConfig&)        | Configura y arranca en una llamada.               |
| Update()/Abort()/Reset()             | Ciclo de vida.                                    |
| IsRunning()/IsFinished()/IsAborted() | Consultas de estado.                              |
| GetState()/GetError()/GetResult()    | Estado, error y Ku/Tu.                            |

## PIDTuning

| Método           | Descripción                         |
|------------------|-------------------------------------|
| PIDTuning(model) | Constructor a partir de FOPDTModel. |
| IMC_PI()         | IMC-PI con Tf por defecto.          |
| IMC_PI(Tf)       | IMC-PI con Tf explícito.            |
| IMC_PI(IMCSpeed) | IMC-PI con preset de Tf.            |

| Método                                               | Descripción                                                                  |
|------------------------------------------------------|------------------------------------------------------------------------------|
| DefaultLambda()                                      | Devuelve el Tf por defecto; el nombre se conserva por compatibilidad de API. |
| GetModel()                                           | Devuelve el modelo almacenado.                                               |
| Apply(pid,result)                                    | Aplica Kc/Ki/Kd de un resultado válido.                                      |
| Lambda_PI() / Lambda_PI(Tf) / Lambda_PI(IMCSpeed)    | Alias explícito para la regla IMC/Lambda PI.                                 |
| Lambda_PID() / Lambda_PID(Tf) / Lambda_PID(IMCSpeed) | Calcula Lambda PID desde FOPDT y devuelve Kc, Ti, Td, Ki y Kd.               |
| TyreusLuybenPI(Ku,Tu)                                | Calcula PI de Tyreus-Luyben a partir de Ku y Tu.                             |
| TyreusLuybenPID(Ku,Tu)                               | Calcula PID de Tyreus-Luyben a partir de Ku y Tu.                            |

## PIDControlFilters

| Clase                  | Métodos principales                                                                      |
|------------------------|------------------------------------------------------------------------------------------|
| MovingAverageFilter<N> | Reset(), Reset(x0), Update(x), Value(), IsInitialized(), Count(), WindowSize().          |
| MedianFilter<N>        | Reset(), Reset(x0), Update(x), Value(), IsInitialized(), Count(), WindowSize().          |
| LowPassFilter          | SetTimeConstant(), GetTimeConstant(), Reset(), Update(x,dt), Value(), IsInitialized().   |
| ComplementaryFilter    | SetAlpha(), GetAlpha(), Reset(), Update(), UpdateIntegrated(), Value(), IsInitialized(). |

## 19. Limitaciones y buenas prácticas

- El controlador no conoce la física ni las unidades de la planta; el escalado de PV y OP corresponde a la aplicación.
- La protección de proceso debe implementarse fuera del PID y tener prioridad sobre la salida calculada.
- StepTest supone una respuesta aproximadamente monótona y representable por FOPDT en la zona ensayada.
- Procesos integradores, fuertemente oscilatorios, con respuesta inversa o muy no lineales pueden requerir otra identificación.
- El signo de ProcessGain en StepTest debe corresponder a la relación física  $OP \rightarrow PV$ .
- RelayTest infiere el signo de proceso a partir de PIDAction: REVERSE se trata como positivo y DIRECT como negativo.
- Una histéresis de relé demasiado pequeña puede conmutar por ruido; una demasiado grande deforma el ensayo.
- El filtrado excesivo altera la dinámica observada y puede aumentar  $T_0$ .
- Los cambios de tuning, estructura, acción o límites en AUTO son bumpless, pero siempre conviene supervisar el proceso tras un cambio importante.
- Los resultados de identificación son válidos para la región de operación en la que se ha hecho el ensayo; no garantizan linealidad global.

**Sobre la calidad del modelo.** Que StepTest termine con un modelo válido significa que el algoritmo ha podido calcularlo. En aplicaciones exigentes conviene comparar varios escalones o ensayos y evaluar la repetibilidad de  $K_p$ ,  $T_0$  y  $T_p$  antes de dar por definitiva una sintonía.

## 20. Estructura del paquete y preparación para repositorio

```
PIDControl/
|-- library.properties
|-- keywords.txt
|-- README.md
|-- CHANGELOG.md
|-- src/
|   |-- PIDControl.h / .cpp
|   |-- StepTest.h / .cpp
|   |-- RelayTest.h / .cpp
|   |-- PIDTuning.h / .cpp
|   `-- PIDControlFilters.h
|-- examples/
|   |-- BasicPID/
|   |-- ManualAuto/
|   |-- StepTestIdentification/
|   |-- StepTestAnalogIO/
|   |-- RelayAutoTune/
|   |-- IMCTuning/
|   `-- FilteringPV/
|-- docs/
`-- extras/
```

### Instalación

PIDControl está disponible en Arduino Library Manager y puede instalarse directamente desde el gestor de librerías de Arduino IDE buscando `PIDControl`. También puede instalarse manualmente descargando el ZIP desde GitHub o desde la página del proyecto y seleccionando Sketch > Include Library > Add .ZIP Library.

- <https://github.com/deurus/PIDControl>
- <https://garikoitz.info/blog/descargas/PIDControl/PIDControl-1.4.2.zip>

## Apéndice A. Ejemplo mínimo completo

```
#include <PIDControl.h>

double SP = 50.0;
double PV = 0.0;
double OP = 0.0;

const bool pvTracking = true;

PIDControl pid(
    &PV, &OP, &SP,
    1.0, 0.4, 0.0,
    PIDType::PI_D,
    PIDAction::REVERSE,
    pvTracking
);

void setup()
{
    pid.SetSampleTime(100);
    pid.SetOutputLimits(0.0, 100.0);
    pid.SetManualOutput(35.0);
    pid.SetMode(PIDMode::MAN);
}

void loop()
{
    PV = LeerPV();

    pid.Compute();

    EscribirOP(OP);
}
```

El ejemplo deliberadamente deja fuera LeerPV() y EscribirOP(). Esa separación es parte del diseño: PIDControl trabaja en unidades de ingeniería y no está ligado a un tipo concreto de sensor o actuador.

## Apéndice B. Resumen rápido

| Necesidad                           | Clase / función                   |
|-------------------------------------|-----------------------------------|
| Controlar una PV                    | PIDControl                        |
| Trabajar en manual/automático       | SetMode(), SetManualOutput()      |
| Evitar saltos MAN→AUTO              | PV Tracking + bumpless            |
| Cambiar estructura                  | PIDType::PID / PI_D / I_PD        |
| Cambiar sentido de control          | PIDAction::DIRECT / REVERSE       |
| Evitar windup                       | Integración condicional interna   |
| Filtrar ruido                       | PIDControlFilters                 |
| Identificar Kp/T0/Tp                | StepTest                          |
| Normalizar ganancia                 | SetPVRange() + SetOutputRange()   |
| Sintonizar PI por IMC               | PIDTuning                         |
| Obtener Ku/Tu                       | RelayTest                         |
| Configurar RelayTest en una llamada | RelayTestConfig + Start(config)   |
| Obtener PI desde FOPDT              | PIDTuning::IMC_PI() / Lambda_PI() |
| Obtener PID desde FOPDT             | PIDTuning::Lambda_PID()           |
| Obtener PI/PID desde Ku/Tu          | PIDTuning::TyreusLuybenPI()/PID() |
| Recuperar identificación del sketch | IDENT / IDENT STEP / IDENT RELAY  |

Con esta arquitectura se puede pasar desde una lectura de sensor ruidosa hasta un controlador sintonizado sin acoplar la lógica de control al hardware específico.