CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA

Video 06 — Parte 3B: Plataformas Computacionales — Raspberry Pi, Alternativas SBC, Linux Embebido y Arquitectura Híbrida — SBC vs. MCU, Raspberry Pi 4 y 5 (Chip RP1), Kernel PREEMPT_RT, Jetson Orin, BeagleBone PRU y Enlace SBC+MCU

Parte del Curso: Parte 3B — Plataformas Computacionales: Raspberry Pi, Alternativas SBC y Arquitectura Híbrida
Arquitectura / Controlador Estudiado: Linux Embebido en Raspberry Pi 5 (Chip RP1 y libgpiod), Parche PREEMPT_RT, Jetson Orin, BeagleBone PRU y Enlace SBC+MCU

```
Trama Binaria Híbrida SBC <--> MCU: [SOF 0xAA 0x55 | CMD | LEN | PAYLOAD (Setpoint float32) | CRC-16-CCITT]
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería

- Diferencia entre SBC (procesador Cortex-A en GHz con MMU, memoria virtual y Linux multiusuario) y MCU; arquitectura de Raspberry Pi 4 y Raspberry Pi 5 (Cortex-A76 a 2.4 GHz, PCIe y puente sur RP1 con GPIO a 3.3V).
- Mitigación del Jitter de planificación en Linux mediante el parche de kernel de tiempo real PREEMPT_RT, bloqueo de páginas en RAM (mlockall) y aislamiento de núcleos (isolcpus + SCHED_FIFO).
- Alternativas SBC especializadas: NVIDIA Jetson Orin Nano (GPU con Tensor Cores para visión artificial y ROS 2), BeagleBone Black/AI-64 (unidades de tiempo real programables PRU a 200 MHz con latencia de 5 ns) y PCs industriales en riel DIN (Compute Module 4/5 con eMMC, Revolution Pi, IOT2050).
- Configuración mediante Device Tree Overlays (config.txt), control de pines moderno por descriptor de carácter /dev/gpiochip (libgpiod/lgpio), demonios systemd y arquitectura híbrida Maestro (SBC) + Coprocesador de Tiempo Real (MCU) con verificación CRC-16.

2. Parámetros Nominales de Configuración del Controlador (Video 06)

Parámetro del Controlador / Interfase	Símbolo	Valor Nominal	Función en el Lazo de Control / Automatización
Ganancia Proporcional	`Kp (PB = 100/Kp)`	3.2	Esfuerzo inmediato proporcional al error actual e(t)
Tiempo Integral	`Ti [s]`	0.5 s	Elimina el error estacionario (Offset) acumulando la integral del error
Tiempo Derivativo y Filtro Divisor	`Td [s] / N`	Td = 0.09 s, N = 10	Anticipa la tendencia sobre -PV y atenúa el ruido de alta frecuencia
Periodo de Muestreo en Tiempo Real	`Ts [ms]`	5 ms	Intervalo determinista de conversión ADC, cálculo discreto y actualización PWM
Límites de Saturación Anti-Windup	`[u_min, u_max]`	[-10, 10]	Cota física del actuador para integración condicional (Clamping)
Rango Escalado de Ingeniería (SCALE_X)	`[PV_min, PV_max]`	[0, 1000]	Conversión lineal desde cuentas crudas del ADC (0..1023 o 0..27648)

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Computadoras de Placa Única (SBC) frente a Microcontroladores

Procesadores de Aplicación (ARM Cortex-A), Unidad de Gestión de Memoria (MMU) y Sistema Operativo Completo

- **Turno 1 (Masculina):** En la Parte tres B entramos al terreno de las Computadoras de Placa Única, conocidas por su sigla S B C: verdaderos ordenadores completos integrados en una sola tarjeta del tamaño de una tarjeta de crédito.
- **Turno 2 (Femenina):** La diferencia arquitectónica clave frente a un microcontrolador es que una S B C emplea un Procesador de Aplicación, como la familia ARM Cortex-A de sesenta y cuatro bits a varios gigahercios, equipado con una Unidad de Gestión de Memoria o M M U.
- **Turno 3 (Masculina):** La unidad M M U traduce direcciones virtuales a físicas y aísla la memoria de cada proceso, lo que permite ejecutar un sistema operativo completo como GNU Linux con sistema de archivos, pila de red TCP IP, servidor gráfico y múltiples gigabytes de memoria RAM.
- **Turno 4 (Femenina):** Gracias a ello, una S B C puede ejecutar tareas imposibles para un microcontrolador pequeño: visión artificial con cámaras de alta velocidad, cinemática inversa de robots complejos, bases de datos locales y paneles de control web interactivos.

Diapositiva 02: Arquitectura de Raspberry Pi 4 y Raspberry Pi 5

Procesador Broadcom Quad-Core Cortex-A76 a 2.4 GHz, Bus PCIe, Controlador I/O RP1 y Cabecera GPIO de 40 Pines

- **Turno 1 (Masculina):** El referente mundial de las computadoras de placa única en educación, investigación e industria ligera es la familia Raspberry Pi, encabezada hoy por los modelos Raspberry Pi cuatro y Raspberry Pi cinco.
- **Turno 2 (Femenina):** La Raspberry Pi cinco incorpora el procesador Broadcom de cuatro núcleos ARM Cortex-A76 de sesenta y cuatro bits a dos punto cuatro gigahercios, acompañado de hasta ocho o dieciséis gigabytes de memoria LPDDR cuatro equis y un carril PCI Express para discos de estado sólido y aceleradores de inteligencia artificial.
- **Turno 3 (Masculina):** Una novedad fundamental en la arquitectura de Raspberry Pi cinco es el chip controlador de entrada y salida dedicado llamado erre pe uno: un puente sur diseñado a medida que gestiona el tráfico USB, las cámaras MIPI y la cabecera estándar de cuarenta pines.
- **Turno 4 (Femenina):** Esa cabecera de cuarenta pines expone veintiocho líneas G P I O multifunción con buses I dos C, S P I, UART y canales P W M; pero recuerda siempre su regla eléctrica inquebrantable: todos sus pines operan a tres punto tres voltios sin protección contra cinco voltios.

Diapositiva 03: Determinismo en Linux Embebido y el Parche de Tiempo Real PREEMPT_RT

Causas del Jitter de Planificación en un Sistema Operativo Estándar y Aislamiento de Núcleos (isolcpus)

- **Turno 1 (Masculina):** Si una Raspberry Pi corre a dos punto cuatro gigahercios, ¿por qué no deberíamos conectar directamente un puente H o leer un encoder de alta frecuencia por software desde un programa normal en Linux estándar?
- **Turno 2 (Femenina):** Porque el kernel Linux convencional está diseñado para maximizar el rendimiento global y la equidad entre decenas de procesos, no el determinismo temporal: en cualquier instante nuestro hilo de control puede ser suspendido varios milisegundos por el planificador, por escrituras en disco o por un fallo de página de memoria.
- **Turno 3 (Masculina):** Esa variación impredecible en el instante de muestreo se denomina Jitter o fluctuación de latencia, y destruye el margen de fase de cualquier lazo de control rápido o deforma los pulsos de un motor paso a paso.
- **Turno 4 (Femenina):** Cuando necesitamos ejecutar control en tiempo real directamente sobre Linux, aplicamos tres configuraciones de ingeniería: compilamos el kernel con el parche de tiempo real PREEMPT erre te, bloqueamos las páginas de memoria en RAM para impedir intercambios a disco, y aislamos uno de los cuatro núcleos en exclusiva para el hilo de control.

Diapositiva 04: Alternativa SBC 1 — NVIDIA Jetson Nano y Jetson Orin Nano

Aceleración por GPU con Tensor Cores, Visión Artificial por Computadora, Inferencia en el Borde y ROS 2

- **Turno 1 (Masculina):** Cuando el requerimiento central de un robot autónomo, vehículo guiado automático o sistema de inspección de calidad es procesar imágenes e inteligencia artificial en tiempo real, la alternativa líder frente a Raspberry Pi es la familia NVIDIA Jetson.
- **Turno 2 (Femenina):** Placas como NVIDIA Jetson Nano y Jetson Orin Nano combinan un procesador multicore ARM Cortex-A con una potente unidad de procesamiento gráfico NVIDIA integrada en el mismo módulo, dotada de cientos de núcleos CUDA y núcleos Tensor especializados en álgebra matricial.
- **Turno 3 (Masculina):** Mientras una CPU tradicional tardaría cientos de milisegundos en evaluar una red neuronal profunda de detección de objetos, la Jetson Orin Nano alcanza hasta cuarenta billones de operaciones por segundo, procesando múltiples cámaras en paralelo a más de sesenta cuadros por segundo.
- **Turno 4 (Femenina):** Sobre su sistema operativo basado en Ubuntu se ejecuta de forma nativa el ecosistema robótico ROS dos junto con aceleración por hardware TensorRT, convirtiéndola en el cerebro de percepción visual y navegación autónoma por excelencia.

Diapositiva 05: Alternativa SBC 2 — BeagleBone Black y BeagleBone AI-64 (Unidades PRU)

Procesadores Texas Instruments Sitara con Coprocesadores de Tiempo Real Programables (PRU a 200 MHz)

- **Turno 1 (Masculina):** La segunda gran alternativa en el mundo de las computadoras de placa única resuelve de manera brillante el dilema entre tener Linux completo y tener tiempo real duro en un solo chip: se trata de la familia BeagleBone, impulsada por procesadores Sitara de Texas Instruments.
- **Turno 2 (Femenina):** Tanto la clásica BeagleBone Black como la potente BeagleBone A I sesenta y cuatro integran dentro del mismo circuito integrado del procesador principal un subsistema llamado Unidades de Tiempo Real Programables, abreviado P R U.
- **Turno 3 (Masculina):** Las unidades P R U son microcontroladores RISC independientes de treinta y dos bits que corren a doscientos megahercios sin memoria caché ni segmentación compleja, ejecutando cada instrucción en exactamente cinco nanosegundos con acceso directo a los pines físicos.
- **Turno 4 (Femenina):** De este modo, mientras el núcleo ARM principal ejecuta Linux, servidores de red y bases de datos, las unidades P R U ejecutan en paralelo el lazo de control de motores y la lectura de encoders a megahercios, comunicándose con Linux a través de memoria RAM compartida.

Diapositiva 06: Alternativa SBC 3 — PCs Industriales y Variantes Robustas para Riel DIN

Raspberry Pi Compute Module (CM4/CM5 con eMMC), Revolution Pi, Siemens IOT2050 y LattePanda x86

- **Turno 1 (Masculina):** En una planta de manufactura real nunca debemos instalar una placa comercial de escritorio con tarjeta microSD suelta y cables puente dentro de un gabinete sometido a vibración mecánica, polvo, calor y transitorios electromagnéticos.
- **Turno 2 (Femenina):** La tarjeta microSD es el punto de falla más frecuente por desgaste de escrituras y falsos contactos mecánicos; por ello la versión industrial llamada Raspberry Pi Compute Module cuatro y cinco reemplaza la ranura por memoria Flash e M M C soldada directamente en placa y conectores de alta densidad.
- **Turno 3 (Masculina):** Sobre ese módulo se construyen computadoras industriales para riel DIN como Kunbus Revolution Pi, o bien pasarelas industriales como Siemens SIMATIC IOT veinte cincuenta, dotadas de carcasa blindada, alimentación de veinticuatro voltios, aislamiento galvánico y puertos RS cuatrocientos ochenta y cinco.
- **Turno 4 (Femenina):** Y cuando la aplicación exige compatibilidad con software industrial compilado para arquitectura equis ochenta y seis de Intel, empleamos placas como LattePanda o PCs industriales sin ventilador con disipación pasiva por chasis de aluminio.

Diapositiva 07: Configuración de Bajo Nivel en Linux Embebido

Árbol de Dispositivos (Device Tree Overlays en config.txt) y Subsistema Moderno /dev/gpiochip (libgpiod)

- **Turno 1 (Masculina):** ¿Cómo sabe el kernel de Linux en una Raspberry Pi o BeagleBone qué pines de la cabecera deben funcionar como bus I dos C, como puerto serie UART o como salida P W M por hardware? Mediante el Árbol de Dispositivos o Device Tree.
- **Turno 2 (Femenina):** Al encender la placa, el cargador de arranque lee el archivo de configuración config punto txt y aplica capas llamadas Device Tree Overlays: pequeñas plantillas declarativas que habilitan los controladores del kernel y multiplexan los pines físicos sin necesidad de recompilar el sistema operativo.
- **Turno 3 (Masculina):** En cuanto al control de pines desde nuestros programas, debemos saber que la antigua interfaz basada en escribir texto en archivos virtuales dentro de la carpeta sys class gpio ha sido declarada obsoleta y eliminada de los kernels modernos por su lentitud y falta de seguridad.
- **Turno 4 (Femenina):** El estándar oficial actual de Linux es el dispositivo de carácter dev gpiochip, gobernado mediante las librerías libgpiod en C más más y Python, o lgpio en Raspberry Pi cinco: permite conmutar múltiples líneas simultáneamente y capturar flancos de interrupción con marca de tiempo exacta en el kernel.

Diapositiva 08: Programación en Python y C++ sobre Raspberry Pi para Mecatrónica

Arquitectura Multihilo, Comunicación Inter-Procesos (IPC), Demonios systemd y Telemetría MQTT/WebSockets

- **Turno 1 (Masculina):** Al desarrollar el software de supervisión y control en una Raspberry Pi estructuramos la aplicación separando las responsabilidades en procesos o hilos independientes para que la interfaz gráfica jamás congele la comunicación con la máquina.
- **Turno 2 (Femenina):** Un hilo dedicado de alta prioridad gestiona en bucle continuo la lectura y escritura del puerto serie o bus CAN hacia los actuadores; un segundo hilo ejecuta el procesamiento matemático o de visión; y un tercer hilo sirve la telemetría por protocolo MQTT o WebSockets hacia el panel de operador.
- **Turno 3 (Masculina):** Cuando combinamos un proceso rápido escrito en C más más con un panel o modelo de inteligencia artificial escrito en Python, ambos programas intercambian estructuras de datos en microsegundos mediante Comunicación Inter-Procesos usando memoria compartida o sockets locales.
- **Turno 4 (Femenina):** Finalmente, para que la máquina opere de manera autónoma en fábrica sin teclado ni monitor, registramos nuestro programa como un servicio o demonio de systemd configurado con arranque automático al encender y reinicio inmediato en caso de cualquier fallo.

Diapositiva 09: Arquitectura Mecatrónica Híbrida Maestro-Coprocesador (SBC + MCU)

Reparto Óptimo de Tareas entre Alto Nivel (SBC) y Tiempo Real Duro (MCU) mediante Tramas Binarias con CRC-16

- **Turno 1 (Masculina):** Llegamos así al patrón arquitectónico más importante de la robótica moderna, las impresoras industriales y los vehículos autónomos: la arquitectura híbrida que une una computadora S B C con uno o varios microcontroladores dedicados.
- **Turno 2 (Femenina):** En lugar de forzar a Linux a hacer tiempo real de microsegundos o forzar a un pequeño microcontrolador a procesar cámaras y pantallas, ponemos a cada procesador a hacer aquello para lo que fue diseñado.
- **Turno 3 (Masculina):** La computadora S B C —como una Raspberry Pi cinco o Jetson— actúa como Maestro de Alto Nivel: procesa visión artificial, resuelve la planificación de trayectorias, gestiona la interfaz táctil H M I y se conecta a la nube industrial.
- **Turno 4 (Femenina):** A través de un enlace rápido UART, USB, S P I o bus CAN con tramas binarias protegidas por suma de verificación C R C dieciséis, la S B C envía las consignas de velocidad o posición al microcontrolador STM32 o ESP32, el cual ejecuta el lazo pe i de duro a diez kilohercios y garantiza la seguridad instantánea.

Diapositiva 10: Síntesis de Plataformas SBC y Arquitecturas Híbridas

Integración de Cómputo de Alto Nivel y Control de Tiempo Real rumbo a los Sistemas de Entrada/Salida

- **Turno 1 (Masculina):** Con este módulo completamos la Parte tres del curso, habiendo dominado tanto los microcontroladores de tiempo real como las computadoras de placa única y su integración en arquitecturas híbridas maestro-coprocesador.
- **Turno 2 (Femenina):** Ahora sabemos cuándo elegir una Raspberry Pi cinco, una NVIDIA Jetson para inteligencia artificial visual, una BeagleBone con unidades P R U o un Compute Module industrial, y cómo configurar el árbol de dispositivos y los demonios de Linux.
- **Turno 3 (Masculina):** Pero tanto los microcontroladores como las computadoras S B C operan con señales lógicas débiles de tres punto tres voltios que necesitan conectarse al exigente mundo eléctrico de los sensores, periféricos y actuadores industriales.
- **Turno 4 (Femenina):** En la siguiente parte del curso nos adentraremos en los Sistemas de Entrada y Salida, estudiando la estructura interna de las interfases, el direccionamiento, los protocolos de sincronización y los adaptadores periféricos.

4. Firmware C++/MCU: PID Discreto Tustin, FSM y CRC-16 (controlador_tiempo_real_video_06.ino)

```
/*
 * ============================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 06: Video 06 — Parte 3B: Plataformas Computacionales — Raspberry Pi, Alternativas SBC, Linux Embebido y Arquitectura Híbrida
 * Archivo: controlador_tiempo_real_video_06.ino
 * ----------------------------------------------------------------------------
 * Descripción Técnica:
 *   Firmware determinista en C/C++ Embebido para Microcontrolador (compatible
 *   con arquitectura AVR / ARM Cortex-M STM32 / ESP32 / RP2040) que implementa:
 *     1. Acceso a nivel de registros con punteros 'volatile' y máscaras de bits.
 *     2. Controlador PID Discreto por Transformación Bilineal de Tustin con
 *        derivada sobre la variable medida (-PV), filtro derivativo N y
 *        esquema Anti-Windup por retro-cálculo y Clamping.
 *     3. Máquina de Estados Finitos (FSM de Moore) no bloqueante para
 *        secuenciación segura e interbloqueo de emergencia.
 *     4. Trama de telemetría con verificación CRC-16-CCITT hacia SBC/PLC.
 *
 * Enfoque del Módulo:
 *   Linux Embebido en Raspberry Pi 5 (Chip RP1 y libgpiod), Parche PREEMPT_RT, Jetson Orin, BeagleBone PRU y Enlace SBC+MCU
 * ============================================================================
 */

#include <Arduino.h>
#include <stdint.h>

// --- Definición de Pines de Interfase (Entradas Optoaisladas y Salidas PWM/MOSFET) ---
const uint8_t PIN_ADC_PV_SENSOR   = A0; // Entrada analógica PV (ej. Shunt 165 ohm de lazo 4-20 mA)
const uint8_t PIN_ADC_SP_CONSIGNA = A1; // Consigna local o referencia analógica (0..5V / 0..3.3V)
const uint8_t PIN_DI_MARCHA_NO    = 2;  // Entrada optoacoplada PC817: Pulsador de Marcha (NO)
const uint8_t PIN_DI_PARO_NC      = 3;  // Entrada optoacoplada PC817: Paro / Emergencia (Cableado NC)
const uint8_t PIN_PWM_ACTUADOR    = 9;  // Salida PWM hacia Driver MOSFET / Puente H
const uint8_t PIN_DO_SSR_ALARMA   = 8;  // Salida hacia Relé de Estado Sólido (SSR) / Piloto

// --- Parámetros Nominales del Controlador PID Discreto (Video 06) ---
const float KP_GAIN      = 3.200f;
const float TI_SEC       = 0.500f;
const float TD_SEC       = 0.090f;
const float N_FILT       = 10.0f;
const float TS_SEC       = 0.0050f; // Periodo de muestreo Ts = 5 ms
const float U_MIN_LIM    = -10.0f;
const float U_MAX_LIM    = 10.0f;
const float PV_MIN_ENG   = 0.0f;
const float PV_MAX_ENG   = 1000.0f;

// --- Variables Volátiles Compartidas con Interrupción (ISR) ---
volatile bool     g_flagTickControl = false;
volatile uint32_t g_contadorPulsosEncoder = 0;

// --- Estructura de Estado del Controlador PID Discreto (Tustin + Anti-Windup) ---
struct ControladorPIDDiscreto {
  float kp;
  float ki;
  float kd;
  float n_divisor;
  float ts;
  float u_min;
  float u_max;
  float integral_acum;
  float termino_d_prev;
  float pv_prev;
  float error_prev;
  float u_salida;
};

ControladorPIDDiscreto g_pid;
```

```
// --- Máquina de Estados Finitos (FSM No Bloqueante de Moore) ---
enum EstadoControlador : uint8_t {
  ESTADO_REPOSO_SEGURO   = 0,
  ESTADO_PRECARGA_SUAVE  = 1,
  ESTADO_LAZO_PID_ACTIVO = 2,
  ESTADO_FALLA_PARO_NC   = 3
};

EstadoControlador g_estadoActual = ESTADO_REPOSO_SEGURO;
uint32_t g_prevMicrosLazo  = 0;
uint32_t g_timestampEstado = 0;
uint32_t g_pasoMuestreo    = 0;

// --- Rutina de Servicio de Interrupción Externa (Encoder / Sensor Rápido) ---
void isrPulsoEncoder() {
  g_contadorPulsosEncoder++;
}

// --- Inicialización de Coeficientes del PID Discreto ---
void inicializarPID(ControladorPIDDiscreto &c) {
  c.kp             = KP_GAIN;
  c.ki             = (TI_SEC > 1e-5f) ? (KP_GAIN / TI_SEC) : 0.0f;
  c.kd             = KP_GAIN * TD_SEC;
  c.n_divisor      = N_FILT;
  c.ts             = TS_SEC;
  c.u_min          = U_MIN_LIM;
  c.u_max          = U_MAX_LIM;
  c.integral_acum  = 0.0f;
  c.termino_d_prev = 0.0f;
  c.pv_prev        = 0.0f;
  c.error_prev     = 0.0f;
  c.u_salida       = 0.0f;
}

// --- Evaluación en Tiempo Real del PID por Regla Trapezoidal de Tustin ---
float evaluarPasoPID_Tustin(ControladorPIDDiscreto &c, float sp, float pv) {
  float error = sp - pv;

  // 1. Término Proporcional
  float p_term = c.kp * error;

  // 2. Término Derivativo sobre -PV filtrado por Tustin (Evita Derivative Kick)
  float tau_d  = (c.kp > 1e-6f) ? ((c.kd / c.kp) / c.n_divisor) : 0.0f;
  float denom  = 2.0f * tau_d + c.ts;
  float alpha1 = (2.0f * tau_d - c.ts) / denom;
  float beta0  = (2.0f * c.kd) / denom;
  float d_term = alpha1 * c.termino_d_prev - beta0 * (pv - c.pv_prev);

  // 3. Candidato de Integración Trapezoidal (Tustin)
  float delta_i = 0.5f * c.ki * c.ts * (error + c.error_prev);
  float i_cand  = c.integral_acum + delta_i;

  // 4. Suma Pre-Saturación y Esquema Anti-Windup Condicional (Clamping)
  float u_unsat = p_term + i_cand + d_term;
  float u_sat   = u_unsat;
  if (u_unsat > c.u_max) {
    u_sat = c.u_max;
    if (error < 0.0f) c.integral_acum = i_cand; // Solo integra si ayuda a desaturar
  } else if (u_unsat < c.u_min) {
    u_sat = c.u_min;
    if (error > 0.0f) c.integral_acum = i_cand;
  } else {
    c.integral_acum = i_cand;
  }

  // Actualizar memoria de estados discretos z^-1
  c.termino_d_prev = d_term;
```

```
  c.pv_prev        = pv;
  c.error_prev     = error;
  c.u_salida       = u_sat;
  return u_sat;
}

// --- Escalado Lineal Equivalente a NORM_X + SCALE_X ---
inline float escalarCuentasADC(int cuentasCrudas, int rawMin, int rawMax, float engMin, float engMax) {
  float u_norm = (float)(cuentasCrudas - rawMin) / (float)(rawMax - rawMin);
  if (u_norm < 0.0f) u_norm = 0.0f;
  if (u_norm > 1.0f) u_norm = 1.0f;
  return engMin + u_norm * (engMax - engMin);
}

// --- Cálculo de Verificación de Redundancia Cíclica CRC-16-CCITT (Polinomio 0x1021) ---
uint16_t calcularCRC16(const uint8_t *datos, uint8_t longitud) {
  uint16_t crc = 0xFFFFU;
  for (uint8_t i = 0; i < longitud; i++) {
    crc ^= ((uint16_t)datos[i] << 8);
    for (uint8_t b = 0; b < 8; b++) {
      if (crc & 0x8000U) {
        crc = (crc << 1) ^ 0x1021U;
      } else {
        crc <<= 1;
      }
    }
  }
  return crc;
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN_DI_MARCHA_NO, INPUT_PULLUP);
  pinMode(PIN_DI_PARO_NC, INPUT_PULLUP);
  pinMode(PIN_PWM_ACTUADOR, OUTPUT);
  pinMode(PIN_DO_SSR_ALARMA, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(PIN_DI_MARCHA_NO), isrPulsoEncoder, RISING);
  inicializarPID(g_pid);
  g_prevMicrosLazo  = micros();
  g_timestampEstado = millis();

  Serial.println(F("# ================================================================"));
  Serial.println(F("# CURSO CONTROLADORES EN MECATRONICA - VIDEO 06"));
  Serial.println(F("# Columnas: t_s,estado_fsm,SP_eng,PV_eng,u_PID,pulsos_enc,crc16"));
  Serial.println(F("# ================================================================"));
}

void loop() {
  uint32_t ahoraMicros = micros();
  uint32_t periodoMicros = (uint32_t)(TS_SEC * 1000000.0f);

  if (ahoraMicros - g_prevMicrosLazo >= periodoMicros) {
    g_prevMicrosLazo += periodoMicros;
    float t_sec = g_pasoMuestreo * TS_SEC;

    // 1. Lectura de Entradas Digitales (Seguridad de Paro NC y Marcha NO)
    bool paroCableadoOK = (digitalRead(PIN_DI_PARO_NC) == HIGH);
    bool ordenMarcha    = (digitalRead(PIN_DI_MARCHA_NO) == LOW);

    // 2. Lectura y Escalado de Entradas Analógicas (NORM_X -> SCALE_X)
    int rawPV = analogRead(PIN_ADC_PV_SENSOR);
    int rawSP = analogRead(PIN_ADC_SP_CONSIGNA);
    float pv_eng = escalarCuentasADC(rawPV, 0, 1023, PV_MIN_ENG, PV_MAX_ENG);
    float sp_eng = escalarCuentasADC(rawSP, 0, 1023, PV_MIN_ENG, PV_MAX_ENG);

    // 3. Evaluación de la Máquina de Estados Finitos (FSM de Moore No Bloqueante)
    if (!paroCableadoOK) {
```

```
    g_estadoActual = ESTADO_FALLA_PARO_NC;
  }

  float mandoActuador = 0.0f;
  switch (g_estadoActual) {
    case ESTADO_REPOSO_SEGURO:
      mandoActuador = 0.0f;
      digitalWrite(PIN_DO_SSR_ALARMA, LOW);
      if (ordenMarcha && paroCableadoOK) {
        g_timestampEstado = millis();
        g_estadoActual    = ESTADO_PRECARGA_SUAVE;
      }
      break;

    case ESTADO_PRECARGA_SUAVE:
      mandoActuador = 0.25f * U_MAX_LIM;
      if (millis() - g_timestampEstado >= 200UL) {
        // Transferencia sin saltos (Bumpless) al activar el lazo PID
        g_pid.integral_acum = mandoActuador;
        g_pid.pv_prev       = pv_eng;
        g_estadoActual      = ESTADO_LAZO_PID_ACTIVO;
      }
      break;

    case ESTADO_LAZO_PID_ACTIVO:
      mandoActuador = evaluarPasoPID_Tustin(g_pid, sp_eng, pv_eng);
      digitalWrite(PIN_DO_SSR_ALARMA, LOW);
      break;

    case ESTADO_FALLA_PARO_NC:
    default:
      mandoActuador = 0.0f;
      g_pid.integral_acum = 0.0f;
      digitalWrite(PIN_DO_SSR_ALARMA, HIGH);
      if (paroCableadoOK && ordenMarcha) {
        g_estadoActual = ESTADO_REPOSO_SEGURO;
      }
      break;
  }

  // 4. Escritura de Salida PWM hacia Etapa de Potencia (MOSFET / Puente H)
  float u_norm_pwm = (mandoActuador - U_MIN_LIM) / (U_MAX_LIM - U_MIN_LIM);
  int duty8bit     = constrain((int)(u_norm_pwm * 255.0f), 0, 255);
  analogWrite(PIN_PWM_ACTUADOR, duty8bit);

  // 5. Lectura Atómica de Contador de Interrupción y Telemetría con CRC-16
  if (g_pasoMuestreo % 10 == 0) {
    noInterrupts();
    uint32_t pulsosCopia = g_contadorPulsosEncoder;
    interrupts();

    uint8_t tramaBytes[6];
    tramaBytes[0] = (uint8_t)g_estadoActual;
    tramaBytes[1] = (uint8_t)duty8bit;
    tramaBytes[2] = (uint8_t)((rawPV >> 8) & 0xFF);
    tramaBytes[3] = (uint8_t)(rawPV & 0xFF);
    tramaBytes[4] = (uint8_t)((pulsosCopia >> 8) & 0xFF);
    tramaBytes[5] = (uint8_t)(pulsosCopia & 0xFF);
    uint16_t crc  = calcularCRC16(tramaBytes, 6);

    Serial.print(t_sec, 3);              Serial.print(',');
    Serial.print((int)g_estadoActual);   Serial.print(',');
    Serial.print(sp_eng, 2);             Serial.print(',');
    Serial.print(pv_eng, 2);             Serial.print(',');
    Serial.print(mandoActuador, 2);      Serial.print(',');
    Serial.print(pulsosCopia);           Serial.print(",0x");
    Serial.println(crc, HEX);
  }
```

```
    g_pasoMuestreo++;
  }
}
```

5. Script Python SBC: Sintonización, Índices IAE/ITAE y Tramas (supervision_sbc_y_sintonizacion_video_06.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
VIDEO 06: Video 06 — Parte 3B: Plataformas Computacionales — Raspberry Pi, Alternativas SBC, Linux Embebido y
Arquitectura Híbrida
Archivo: supervision_sbc_y_sintonizacion_video_06.py
--------------------------------------------------------------------------------
Descripción Técnica:
  Script integral en Python para Computadora de Placa Única (Raspberry Pi /
  SBC) y estación de ingeniería que implementa:
    1. Simulación en lazo cerrado de un controlador PID discreto (Tustin +
       Filtro Derivativo N + Anti-Windup por Clamping y Back-Calculation).
    2. Cálculo cuantitativo de métricas transitorias (tr, tp, Mp, ts) y de
       los cuatro índices integrales de desempeño (IAE, ISE, ITAE, ITSE).
    3. Reglas automáticas de sintonización de Ziegler-Nichols (1.º y 2.º
       método), Cohen-Coon y sintonización robusta IMC / Lambda.
    4. Construcción y validación de tramas binarias Maestro-Coprocesador
       (SBC <-> MCU/PLC) protegidas con suma de verificación CRC-16-CCITT.
================================================================================
"""

import math
import struct
from typing import Dict, List, Tuple

# Parámetros nominales del Video 06
KP_NOMINAL = 3.2
TI_NOMINAL = 0.5
TD_NOMINAL = 0.09
N_FILTRO   = 10
TS_SEG     = 0.0050
U_MIN      = -10
U_MAX      = 10


def sintonizar_pid_ziegler_nichols_y_lambda(
    K_planta: float, L_retardo: float, T_constante: float, lambda_deseado: float
) -> Dict[str, Dict[str, float]]:
    """
    Calcula las ganancias PID (Kp, Ti, Td) mediante:
      - Ziegler-Nichols 1.er Método (Curva de Reacción FOPDT)
      - Método de Cohen-Coon (para retardo significativo L/T)
      - Método IMC / Regla Lambda (respuesta sin sobreimpulso)
    """
    r = L_retardo / max(T_constante, 1e-6)

    # 1. Ziegler-Nichols Lazo Abierto (PID Estándar ISA)
    kp_zn = (1.2 * T_constante) / (K_planta * max(L_retardo, 1e-6))
    ti_zn = 2.0 * L_retardo
    td_zn = 0.5 * L_retardo

    # 2. Cohen-Coon (PID)
    kp_cc = (1.0 / (K_planta * r)) * (4.0 / 3.0 + r / 4.0)
    ti_cc = L_retardo * (32.0 + 6.0 * r) / (13.0 + 8.0 * r)
    td_cc = L_retardo * 4.0 / (11.0 + 2.0 * r)

    # 3. IMC / Lambda (PI robusto sin sobreimpulso + término D para retardo Padé)
    kp_imc = T_constante / (K_planta * (lambda_deseado + L_retardo))
    ti_imc = T_constante
    td_imc = (T_constante * L_retardo) / (2.0 * T_constante + L_retardo)

    return {
        "Ziegler_Nichols_FOPDT": {
            "Kp": round(kp_zn, 4), "Ti_s": round(ti_zn, 4), "Td_s": round(td_zn, 4)
        },
```

```
        "Cohen_Coon": {
            "Kp": round(kp_cc, 4), "Ti_s": round(ti_cc, 4), "Td_s": round(td_cc, 4)
        },
        "IMC_Lambda_Robusto": {
            "Kp": round(kp_imc, 4), "Ti_s": round(ti_imc, 4), "Td_s": round(td_imc, 4)
        },
    }


def simular_lazo_pid_discreto_tustin(
    kp: float, ti: float, td: float, n_filt: float, ts: float, pasos: int = 800
) -> Dict[str, float]:
    """
    Simula una planta mecatrónica de 2.º orden controlada por un PID discreto
    discretizado por Tustin con Anti-Windup y calcula tr, Mp, ts, IAE, ISE, ITAE.
    """
    ki = kp / ti if ti > 1e-6 else 0.0
    kd = kp * td
    tau_d = (td / n_filt) if n_filt > 0 else 0.0
    denom_d = 2.0 * tau_d + ts
    alpha_d = (2.0 * tau_d - ts) / denom_d
    beta_d  = (2.0 * kd) / denom_d

    # Estado de la planta (wn = 8 rad/s, zeta = 0.45, K = 1.0)
    wn = 8.0
    zeta = 0.45
    y_pos = 0.0
    y_vel = 0.0
    sp = 1.0

    integral = 0.0
    d_prev = 0.0
    y_prev = 0.0
    e_prev = 0.0

    iae = 0.0
    ise = 0.0
    itae = 0.0
    itse = 0.0
    y_max = 0.0
    t_pico = 0.0
    t_10 = None
    t_90 = None
    t_estable = 0.0

    for k in range(pasos):
        t = k * ts
        error = sp - y_pos

        # Acumulación de índices integrales de desempeño
        abs_e = abs(error)
        iae  += abs_e * ts
        ise  += (error ** 2) * ts
        itae += (t * abs_e) * ts
        itse += (t * (error ** 2)) * ts

        # Algoritmo PID Discreto con filtro derivativo sobre -PV
        p_term = kp * error
        d_term = alpha_d * d_prev - beta_d * (y_pos - y_prev)
        i_cand = integral + 0.5 * ki * ts * (error + e_prev)

        u_unsat = p_term + i_cand + d_term
        u_cmd = max(U_MIN, min(U_MAX, u_unsat))

        # Anti-Windup condicional (Clamping)
        if u_unsat == u_cmd or (u_unsat > U_MAX and error < 0) or (u_unsat < U_MIN and error > 0):
            integral = i_cand
```

```
        d_prev = d_term
        y_prev = y_pos
        e_prev = error

        # Integración de la planta física (Euler semi-implícito)
        acel = (wn ** 2) * u_cmd - 2.0 * zeta * wn * y_vel - (wn ** 2) * y_pos
        y_vel += acel * ts
        y_pos += y_vel * ts

        if y_pos >= 0.10 * sp and t_10 is None:
            t_10 = t
        if y_pos >= 0.90 * sp and t_90 is None:
            t_90 = t
        if y_pos > y_max:
            y_max = y_pos
            t_pico = t
        if abs(sp - y_pos) > 0.02 * sp:
            t_estable = t

    tr = (t_90 - t_10) if (t_10 is not None and t_90 is not None) else 0.0
    mp_pct = max(0.0, (y_max - sp) / sp * 100.0)

    return {
        "tiempo_subida_tr_s": round(tr, 4),
        "tiempo_pico_tp_s": round(t_pico, 4),
        "sobreimpulso_Mp_pct": round(mp_pct, 2),
        "tiempo_establecimiento_ts_2pct_s": round(t_estable, 4),
        "IAE": round(iae, 4),
        "ISE": round(ise, 4),
        "ITAE": round(itae, 4),
        "ITSE": round(itse, 4),
    }


def crc16_ccitt(data: bytes) -> int:
    """Calcula CRC-16-CCITT (0x1021, init 0xFFFF) para enlace serie SBC <-> MCU."""
    crc = 0xFFFF
    for byte in data:
        crc ^= (byte << 8)
        for _ in range(8):
            if crc & 0x8000:
                crc = ((crc << 1) ^ 0x1021) & 0xFFFF
            else:
                crc = (crc << 1) & 0xFFFF
    return crc


def empaquetar_trama_consigna_sbc(comando: int, setpoint_float: float) -> bytes:
    """Construye trama binaria: [0xAA, 0x55, CMD, LEN=4, float32_LE, CRC16_BE]."""
    cabecera_y_payload = struct.pack("<BBBf", 0xAA, 0x55, comando, 4, setpoint_float)
    crc = crc16_ccitt(cabecera_y_payload)
    return cabecera_y_payload + struct.pack(">H", crc)


if __name__ == "__main__":
    print("=" * 78)
    print(" CURSO: CONTROLADORES EN MECATRÓNICA — VIDEO 06")
    print(" Enfoque:", "Linux Embebido en Raspberry Pi 5 (Chip RP1 y libgpiod), Parche PREEMPT_RT, Jetson Orin,
BeagleBone PRU y Enlace SBC+MCU")
    print("=" * 78)

    tablas = sintonizar_pid_ziegler_nichols_y_lambda(
        K_planta=1.25, L_retardo=0.15, T_constante=0.85, lambda_deseado=0.35
    )
    print("\n[1] Sintonización Comparativa (Ziegler-Nichols, Cohen-Coon e IMC Lambda):")
    for metodo, gan in tablas.items():
        print(f"    {metodo:24s} -> Kp = {gan['Kp']:6.3f} | Ti = {gan['Ti_s']:6.3f} s | Td = {gan['Td_s']:6.3f}
s")
```

```
metricas = simular_lazo_pid_discreto_tustin(KP_NOMINAL, TI_NOMINAL, TD_NOMINAL, N_FILTRO, 0.005)
print("\n[2] Desempeño Temporal e Índices Integrales (IAE, ISE, ITAE, ITSE):")
for k_m, v_m in metricas.items():
    print(f"    {k_m:34s} : {v_m}")

trama = empaquetar_trama_consigna_sbc(comando=0x01, setpoint_float=500)
print("\n[3] Trama Binaria Híbrida SBC -> MCU con CRC-16-CCITT:", trama.hex(" ").upper())
```

6. Programa PLC IEC 61131-3 (ST + IL): Enclavamiento, GRAFCET y Escalado 4-20 mA (automatizacion_plc_iec61131_video_06.st)

```
(* ==========================================================================
 * CURSO COMPLETO: CONTROLADORES EN MECATRÓNICA
 * VIDEO 06: Video 06 — Parte 3B: Plataformas Computacionales — Raspberry Pi, Alternativas SBC, Linux Embebido y
Arquitectura Híbrida
 * Archivo: automatizacion_plc_iec61131_video_06.st
 * --------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) + Equivalencia en Lista de
 *           Instrucciones (IL) y Peldaños de Escalera (Ladder Diagram)
 * Descripción Técnica:
 *   Bloque de Función Industrial (FB_ControladorCeldaMecatronica_V06) que
 *   integra el Ciclo de Barrido (Scan), circuito de enclavamiento prioritario
 *   al paro (cableado NC), secuenciador GRAFCET de 4 etapas, temporizador TON,
 *   contador de lotes CTU, registro de corrimiento de rechazo de calidad SHRB,
 *   escalado analógico 4-20 mA (NORM_X / SCALE_X con detección de rotura de
 *   hilo < 3.6 mA) y algoritmo PID discreto con Anti-Windup.
 * ========================================================================== *)

FUNCTION_BLOCK FB_ControladorCeldaMecatronica_V06
VAR_INPUT
    (* Entradas Digitales de la Tabla de Imagen de Proceso PII (%IX) *)
    xPulsadorMarcha_NO   : BOOL; (* %IX0.0 - Contacto Normalmente Abierto *)
    xPulsadorParo_NC     : BOOL; (* %IX0.1 - Cableado Físicamente NC en Campo *)
    xReleTermico_NC      : BOOL; (* %IX0.2 - Guardamotor NC (1 = Sin Falla) *)
    xSensorPiezaEntrada  : BOOL; (* %IX0.3 - Sensor Inductivo PNP 24V DC *)
    xCamaraPiezaDefecto  : BOOL; (* %IX0.4 - 1 = Envase Defectuoso Detectado *)
    xPulsoEncoderBanda   : BOOL; (* %IX0.5 - Reloj de Desplazamiento de Banda *)
    xResetContadores     : BOOL; (* %IX0.6 - Reinicio Manual de Lote *)

    (* Entradas Analógicas Crudas del Convertidor ADC (%IW) *)
    iEntradaAnalogicaRaw : INT;  (* %IW64  - Lazo 4-20 mA: 0 a 27648 cuentas *)
    rSetpointIngenieria  : REAL := 600.0;
END_VAR

VAR_OUTPUT
    (* Salidas Digitales hacia la Tabla de Imagen de Salidas PIQ (%QX) *)
    xMotorBandaContactor : BOOL; (* %QX0.0 - Contactor Principal Enclavado *)
    xValvulaDosificadora : BOOL; (* %QX0.1 - Solenoide 24V DC con Diodo Flyback *)
    xCilindroExpulsor    : BOOL; (* %QX0.2 - Rechazo en Posición Bit 5 del SHRB *)
    xAlarmaRoturaCable   : BOOL; (* %QX0.3 - Corriente < 3.6 mA o Paro Activo *)
    xLoteCompletado      : BOOL; (* %QX0.4 - Salida Q del Contador CTU *)

    (* Salida Analógica Escalada y Variables de Diagnóstico HMI *)
    iSalidaAnalogicaRaw  : INT;  (* %QW80  - Mando 0..27648 hacia Variador/SSR *)
    rVariableProcesoPV   : REAL; (* Variable física escalada en unidades de ing. *)
    rSalidaControlPID    : REAL; (* Esfuerzo de control 0.0 a 100.0 % *)
    iEstadoGrafcet       : INT;  (* Etapa activa del GRAFCET: 0, 10, 20, 30 *)
END_VAR

VAR
    (* Instancias de Temporizadores y Contadores Estándar IEC 61131-3 *)
    tonDosificado        : TON;
    ctuContadorLotes     : CTU;

    (* Variables Internas de Escalado NORM_X / SCALE_X *)
    rValorNormalizado    : REAL;
    rMinIngenieria       : REAL := 0.0;
    rMaxIngenieria       : REAL := 1000.0;

    (* Registro de Corrimiento de 16 Bits (Shift Register para Rechazo) *)
    wRegistroCorrimiento : WORD := 16#0000;
    xPulsoEncoderPrev    : BOOL := FALSE;
    xFlancoEncoder       : BOOL := FALSE;

    (* Memoria Retentiva del Bloque PID Discreto *)
    rKp                  : REAL := 3.200;
    rTi_s                : REAL := 0.500;
    rTd_s                : REAL := 0.090;
```

```
    rTs_s                : REAL := 0.0050;
    rIntegralAcum        : REAL := 0.0;
    rErrorPrev           : REAL := 0.0;
    rPvPrev              : REAL := 0.0;
END_VAR

(* ================================================================================
 * 1. EQUIVALENCIA EN LISTA DE INSTRUCCIONES (IL) DEL PELDAÑO DE ENCLAVAMIENTO:
 *      LD     xPulsadorMarcha_NO
 *      OR     xMotorBandaContactor
 *      AND    xPulsadorParo_NC
 *      AND    xReleTermico_NC
 *      ANDN   xAlarmaRoturaCable
 *      ST     xMotorBandaContactor
 * ================================================================================ *)
xMotorBandaContactor := (xPulsadorMarcha_NO OR xMotorBandaContactor)
                        AND xPulsadorParo_NC
                        AND xReleTermico_NC
                        AND NOT xAlarmaRoturaCable;

(* ================================================================================
 * 2. NORMALIZACIÓN (NORM_X) Y ESCALADO LINEAL (SCALE_X) DE LAZO 4-20 mA
 *    Detección de rotura de hilo si iEntradaAnalogicaRaw < -1728 (< 3.0 mA)
 * ================================================================================ *)
IF (iEntradaAnalogicaRaw < -1728) OR (NOT xPulsadorParo_NC) OR (NOT xReleTermico_NC) THEN
    xAlarmaRoturaCable := TRUE;
    rValorNormalizado  := 0.0;
ELSE
    xAlarmaRoturaCable := FALSE;
    (* NORM_X: Convierte 0..27648 cuentas a rango flotante 0.0..1.0 *)
    rValorNormalizado  := INT_TO_REAL(iEntradaAnalogicaRaw) / 27648.0;
    IF rValorNormalizado < 0.0 THEN rValorNormalizado := 0.0; END_IF;
    IF rValorNormalizado > 1.0 THEN rValorNormalizado := 1.0; END_IF;
END_IF;

(* SCALE_X: Convierte 0.0..1.0 a Unidades de Ingeniería [rMinIngenieria .. rMaxIngenieria] *)
rVariableProcesoPV := rMinIngenieria + rValorNormalizado * (rMaxIngenieria - rMinIngenieria);

(* ================================================================================
 * 3. REGISTRO DE CORRIMIENTO DE BITS (SHRB / BSL) SINCRONIZADO CON ENCODER
 * ================================================================================ *)
xFlancoEncoder    := xPulsoEncoderBanda AND NOT xPulsoEncoderPrev;
xPulsoEncoderPrev := xPulsoEncoderBanda;

IF xFlancoEncoder AND xMotorBandaContactor THEN
    wRegistroCorrimiento := SHL(IN := wRegistroCorrimiento, N := 1);
    IF xCamaraPiezaDefecto THEN
        wRegistroCorrimiento := wRegistroCorrimiento OR 16#0001;
    END_IF;
END_IF;

(* Si el bit desplazado llega a la estación 5 (bit 4, máscara 16#0010), activa expulsor *)
xCilindroExpulsor := (wRegistroCorrimiento AND 16#0010) <> 16#0000;

(* ================================================================================
 * 4. SECUENCIADOR GRAFCET / SFC DE 4 ETAPAS CON TEMPORIZADOR TON Y CONTADOR CTU
 * ================================================================================ *)
IF NOT xMotorBandaContactor THEN
    iEstadoGrafcet := 0;
END_IF;

CASE iEstadoGrafcet OF
    0: (* Etapa 0: Reposo esperando pieza en estación *)
        xValvulaDosificadora := FALSE;
        IF xMotorBandaContactor AND xSensorPiezaEntrada THEN
            iEstadoGrafcet := 10;
        END_IF;
```

```
    10: (* Etapa 10: Dosificación temporizada con bloque TON *)
        xValvulaDosificadora := TRUE;
        IF tonDosificado.Q THEN
            xValvulaDosificadora := FALSE;
            iEstadoGrafcet       := 20;
        END_IF;

    20: (* Etapa 20: Incremento de contador de lote y espera de liberación *)
        xValvulaDosificadora := FALSE;
        IF NOT xSensorPiezaEntrada THEN
            iEstadoGrafcet := 0;
        END_IF;
END_CASE;

tonDosificado(IN := (iEstadoGrafcet = 10), PT := T#1500MS);
ctuContadorLotes(CU := (iEstadoGrafcet = 20), R := xResetContadores, PV := 50);
xLoteCompletado := ctuContadorLotes.Q;

(* ================================================================================
 * 5. BLOQUE TECNOLÓGICO PID DISCRETO CON ANTI-WINDUP Y DES-ESCALADO ANALÓGICO
 * ================================================================================ *)
IF xMotorBandaContactor AND NOT xAlarmaRoturaCable THEN
    VAR
        rErrorActual : REAL;
        rTerminoP    : REAL;
        rTerminoD    : REAL;
        rCandI       : REAL;
        rSumaUnsat   : REAL;
    END_VAR
    rErrorActual := rSetpointIngenieria - rVariableProcesoPV;
    rTerminoP    := rKp * rErrorActual;
    rTerminoD    := -rKp * rTd_s * (rVariableProcesoPV - rPvPrev) / rTs_s;
    rCandI       := rIntegralAcum + (rKp / rTi_s) * rTs_s * 0.5 * (rErrorActual + rErrorPrev);

    rSumaUnsat   := rTerminoP + rCandI + rTerminoD;
    IF rSumaUnsat > 100.0 THEN
        rSalidaControlPID := 100.0;
    ELSIF rSumaUnsat < 0.0 THEN
        rSalidaControlPID := 0.0;
    ELSE
        rSalidaControlPID := rSumaUnsat;
        rIntegralAcum     := rCandI;
    END_IF;

    rErrorPrev := rErrorActual;
    rPvPrev    := rVariableProcesoPV;
ELSE
    rSalidaControlPID := 0.0;
    rIntegralAcum     := 0.0;
END_IF;

(* Conversión de 0.0..100.0 % a 0..27648 cuentas crudas de salida analógica %QW80 *)
iSalidaAnalogicaRaw := REAL_TO_INT((rSalidaControlPID / 100.0) * 27648.0);

END_FUNCTION_BLOCK
```