CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA

Video 04 — Parte 4: Automatización Cognitiva, Edge AI y TinyML, Control Inteligente y Mantenimiento Predictivo — Cuantización INT8 en MCU, Sintonización Neuro-PID, Modelos NARX, Estimación RUL, Cobots e Industria 5.0

Parte del Curso: Parte 4 (Tema 1D) — Automatización Cognitiva, Edge AI y TinyML, Control Inteligente y Mantenimiento Predictivo
Arquitectura Cognitiva / Algoritmo Estudiado: Cuantización INT8 para TinyML (ARM Cortex-M / ESP32-S3), Control Híbrido Neuro-PID + NARX, Estimación RUL y Robótica Colaborativa

```
q_int8 = clip(round(r_fp32 / S) + Z, -128, 127) | u(t) = u_NeuroPID(t) + τ_FF_NN(q, q̇) | HI(t) -> RUL [h]
```

1. Síntesis de Conceptos Clave y Reglas de Ingeniería Cognitiva

- Evolución hacia la Automatización Cognitiva mediante Edge AI y TinyML: poda de conexiones (Pruning) y cuantización simétrica/asimétrica de coma flotante FP32 a enteros de 8 bits (INT8) ejecutada en TensorFlow Lite Micro + CMSIS-NN y NVIDIA TensorRT.
- Control Inteligente Híbrido: sintonización dinámica en línea Neuro-PID de ganancias [Kp(t), Ki(t), Kd(t)], prealimentación neuronal para cancelar fricción no lineal de Stribeck, e identificación de sistemas dinámicos con redes NARX acopladas a Control Predictivo (MPC).
- Inspección visual automatizada de alta velocidad sincronizada por hardware con el registro de corrimiento (SHRB) del PLC, y Mantenimiento Predictivo Inteligente (PdM) combinando espectros de vibración (BPFO/BPFI) y corriente estatórica (MCSA) para estimar la Vida Útil Remanente (RUL).
- Robótica Colaborativa Cognitiva (Cobots bajo ISO/TS 15066) con observadores de par externo, Control de Impedancia Mecánica, aprendizaje por imitación y sincronización con Gemelos Digitales Cognitivos por OPC UA / MQTT en la Industria 5.0.

2. Parámetros Nominales de Configuración de IA y Control (Video 04)

Parámetro del Sistema Cognitivo / Controlador	Símbolo	Valor Nominal	Función en Percepción, Razonamiento, Aprendizaje o Automatización
Frecuencia de Muestreo y Ventana FFT	`fs [Hz] / N`	2000 Hz, N = 512	Adquisición determinista para descriptores RMS, Curtosis y espectro FFT
Covarianzas del Filtro de Kalman	`Q / R`	Q = 0.0015, R = 0.03	Ponderación óptima entre incertidumbre del modelo dinámico y ruido del sensor
Tasa de Aprendizaje por Gradiente	`η (Learning Rate)`	0.008	Paso de actualización de pesos sinápticos en Backpropagation / Adam
Factor de Escala de Cuantización TinyML	`S (FP32 -> INT8)`	0.00392	Compresión de pesos y activaciones a enteros de 8 bits [-128, 127]
Ganancias Base del Lazo Neuro-PID	`[Kp0, Ki0, Kd0]`	[5.2, 1.8, 0.14]	Ganancias nominales moduladas en línea por la red neuronal y el bloque difuso

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: De la Automatización Rígida a la Automatización Cognitiva

Evolución de los Niveles de Autonomía Industrial e Integración de IA en el Lazo de Campo

- **Turno 1 (Masculina):** Llegamos al cuarto y último pilar de nuestro curso: la Automatización Cognitiva, donde la percepción, el razonamiento y el aprendizaje se funden con los controladores industriales para operar líneas de producción autónomas y autoadaptables.
- **Turno 2 (Femenina):** En la automatización tradicional rígida, un programa secuencial ejecuta ciegamente los mismos movimientos una y otra vez: si cambia la viscosidad del material, se desgasta una herramienta o varía la posición de una pieza, el proceso falla.
- **Turno 3 (Masculina):** Por el contrario, una celda de automatización cognitiva cierra el lazo en múltiples escalas: percibe la variación en línea, diagnostica su origen, recalcula sus ganancias o trayectorias en milisegundos y mantiene la calidad sin detener la planta.
- **Turno 4 (Femenina):** Para lograrlo con seguridad industrial, la inteligencia artificial no puede vivir únicamente en servidores remotos: debe ejecutarse al pie de la máquina, directamente en los microcontroladores, procesadores de borde y controladores lógicos programables.

Diapositiva 02: IA en el Borde (Edge AI y TinyML): Cuantización y Poda

Compresión de Modelos de Coma Flotante (FP32) a Enteros de 8 Bits (INT8) y Latencia Determinista

- **Turno 1 (Masculina):** ¿Por qué ejecutamos los modelos de inteligencia artificial directamente en el borde, disciplina conocida como Edge A I y Tiny M L, en lugar de enviar las lecturas de los sensores a la nube por internet?
- **Turno 2 (Femenina):** Por tres razones críticas en mecatrónica: primero, un lazo de control o protección exige latencia determinista inferior a pocos milisegundos; segundo, la máquina debe seguir operando aunque se corte la red; y tercero, por privacidad y ancho de banda.
- **Turno 3 (Masculina):** Para alojar una red neuronal dentro de los pocos kilobytes de memoria SRAM de un microcontrolador aplicamos dos técnicas matemáticas: la Poda o Pruning, que elimina las conexiones sinápticas de peso cercano a cero, y la Cuantización.
- **Turno 4 (Femenina):** La cuantización transforma los pesos y activaciones desde números en coma flotante de treinta y dos bits hacia enteros con signo de ocho bits, reduciendo cuatro veces el uso de memoria Flash y acelerando drásticamente la inferencia sin pérdida apreciable de exactitud.

Diapositiva 03: Despliegue de Modelos Neuronales en Microcontroladores y GPUs Embebidas

Flujo ONNX / TensorFlow Lite Micro con CMSIS-NN en ARM Cortex-M y Aceleración TensorRT en Jetson

- **Turno 1 (Masculina):** El flujo de ingeniería profesional para desplegar un modelo de inteligencia artificial comienza entrenando y validando la arquitectura en Python sobre una estación de trabajo, para luego exportarla a formatos estándar como ONNX o TensorFlow Lite.
- **Turno 2 (Femenina):** En sistemas basados en microcontroladores ARM Cortex-M, como la familia ese te eme treinta y dos, o en módulos e ese pe treinta y dos ese tres, el modelo cuantizado se convierte en un arreglo constante de bytes en lenguaje C.
- **Turno 3 (Masculina):** El intérprete de TensorFlow Lite Micro reserva un área estática de memoria llamada Tensor Arena, evitando cualquier asignación dinámica fragmentable, y ejecuta las multiplicaciones matriciales en un solo ciclo mediante las instrucciones vectoriales CMSIS-NN.
- **Turno 4 (Femenina):** Cuando la aplicación requiere procesar visión artificial profunda de alta resolución, desplegamos el modelo sobre módulos NVIDIA Jetson Orin utilizando el compilador TensorRT, que fusiona capas convolucionales sobre los núcleos tensoriales de hardware.

Diapositiva 04: Control Inteligente Híbrido: Neuro-PID y Compensación de Fricción

Auto-Sintonización en Línea de Ganancias [Kp, Ki, Kd] y Feedforward Neuronal de No Linealidades

- **Turno 1 (Masculina):** En la ingeniería de control mecatrónico moderna no reemplazamos a ciegas los controladores pe i de probados por cajas negras; en su lugar, construimos arquitecturas de Control Inteligente Híbrido que unen lo mejor de ambos mundos.
- **Turno 2 (Femenina):** El primer esquema híbrido es el regulador Neuro-PID auto-sintonizable: el lazo de realimentación conserva la estructura matemática clásica del pe i de, pero una red neuronal ligera supervisa el error y ajusta dinámicamente las ganancias ka pe, ka i y ka de en cada ciclo.
- **Turno 3 (Masculina):** El segundo esquema de altísimo impacto industrial es la compensación en prealimentación o Feedforward Neuronal: mantenemos un controlador lineal estable en el lazo principal y colocamos una red neuronal en paralelo que aprende las dinámicas parásitas del mecanismo.
- **Turno 4 (Femenina):** Esta red neuronal predice e inyecta exactamente el par necesario para cancelar la fricción no lineal de Stribeck a velocidades cercanas a cero, la histéresis magnética y el par gravitatorio variable, reduciendo el error de seguimiento en un orden de magnitud.

Diapositiva 05: Identificación Neuronal (NARX) y Control Predictivo Aprendido

Modelos Autorregresivos No Lineales con Entradas Exógenas y Optimización en Horizonte Deslizante

- **Turno 1 (Masculina):** Para aplicar técnicas avanzadas de control óptimo sobre máquinas térmicas, químicas o neumáticas cuya física exacta es desconocida, primero realizamos la Identificación de Sistemas mediante redes neuronales dinámicas.
- **Turno 2 (Femenina):** La arquitectura estándar para este fin es la red NARX, siglas de Modelo Autorregresivo No Lineal con Entradas Exógenas, la cual predice la salida futura de la planta en función de las últimas muestras pasadas de la salida medida y del esfuerzo de control.
- **Turno 3 (Masculina):** Una vez entrenada y validada la red NARX con señales de excitación ricas en frecuencias, la integramos como modelo interno dentro de un Controlador Predictivo Basado en Modelo, abreviado eme pe ce.
- **Turno 4 (Femenina):** En cada periodo de muestreo, el controlador predictivo consulta a la red neuronal para simular qué ocurrirá varios pasos hacia el futuro y calcula la secuencia óptima de actuación que minimiza el error sin violar jamás los límites físicos de tensión, presión o velocidad.

Diapositiva 06: Inspección de Calidad Automatizada por Visión de Alta Velocidad

Sincronismo Estroboscópico por Hardware, Inferencia Convolucional en Milisegundos y Rechazo por PLC

- **Turno 1 (Masculina):** Analicemos la arquitectura completa de una celda mecatrónica de inspección óptica automatizada capaz de verificar el cien por ciento de la producción a velocidades de decenas de piezas por segundo.
- **Turno 2 (Femenina):** El ciclo inicia cuando la barrera fotoeléctrica detecta el paso del producto sobre la banda transportadora y dispara de forma síncrona un pulso de iluminación estroboscópica LED junto con el obturador global de la cámara industrial.
- **Turno 3 (Masculina):** La unidad de procesado en el borde ejecuta en menos de ocho milisegundos una red neuronal convolucional de detección o segmentación, identificando microfisuras, errores de ensamblaje, faltantes de rosca o defectos de soldadura invisibles al ojo humano.
- **Turno 4 (Femenina):** Si la pieza es clasificada como defectuosa, el nodo de visión activa una señal digital o trama industrial hacia el registro de corrimiento del P L C, el cual rastrea el avance con el encoder de la banda y dispara la válvula neumática expulsora en el instante exacto.

Diapositiva 07: Mantenimiento Predictivo Inteligente (PdM) y Vida Útil Remanente (RUL)

Firmas de Vibración en Rodamientos (BPFO/BPFI), Corriente Estatórica (MCSA) y Pronóstico de Salud

- **Turno 1 (Masculina):** Una sola parada imprevista en una línea robotizada de manufactura puede costar decenas de miles de dólares por hora; el Mantenimiento Predictivo Inteligente utiliza la inteligencia artificial para anticipar las fallas antes de que detengan la planta.
- **Turno 2 (Femenina):** Instalamos acelerómetros MEMS triaxiales sobre los alojamientos de los rodamientos y sensores de efecto Hall en las fases del motor para capturar de forma continua la firma vibratoria y el espectro de corriente estatórica.
- **Turno 3 (Masculina):** A partir de las amplitudes espectrales en las frecuencias características de paso de bolas por la pista exterior e interior, un modelo de aprendizaje automático calcula en tiempo real un Índice de Salud normalizado entre uno y cero.
- **Turno 4 (Femenina):** A continuación, una red recurrente proyecta la curva de degradación hacia adelante para estimar la Vida Útil Remanente, abreviada erre u ele, indicando con semanas de antelación cuántas horas de operación restan antes de requerir el reemplazo programado del componente.

Diapositiva 08: Robótica Colaborativa Cognitiva (Cobots) e Interacción Segura

Estimación de Par Externo sin Sensores, Control de Impedancia Mecánica y Aprendizaje por Imitación

- **Turno 1 (Masculina):** En la robótica industrial clásica los manipuladores operaban encerrados en jaulas metálicas; hoy, los Robots Colaborativos o Cobots trabajan hombro a hombro con los operarios humanos gracias a la percepción y al control inteligente.
- **Turno 2 (Femenina):** Cumpliendo con la normativa internacional ISO quince mil sesenta y seis, el controlador del cobot estima continuamente el par dinámico esperado en cada articulación y detecta al instante cualquier fuerza de contacto externa para detenerse en milisegundos.
- **Turno 3 (Masculina):** Además, para realizar tareas delicadas de ensamblaje o pulido, sustituimos el control rígido de posición por el Control de Impedancia Mecánica, que hace que el brazo se comporte como un sistema masa-resorte-amortiguador virtual programable.
- **Turno 4 (Femenina):** Y mediante el Aprendizaje por Imitación o Demostración, un técnico puede tomar el brazo con su mano y guiarlo a lo largo de una operación compleja para que el robot codifique y generalice automáticamente la trayectoria aprendida.

Diapositiva 09: Gemelos Digitales Cognitivos e Industria 5.0

Sincronización Bidireccional OPC UA / MQTT, Optimización Energética de Planta y Diseño Centrado en el Humano

- **Turno 1 (Masculina):** Cuando conectamos todas las celdas mecatrónicas cognitivas a través de protocolos industriales como OPC UA y MQTT Sparkplug B, damos vida al Gemelo Digital Cognitivo de la planta completa.
- **Turno 2 (Femenina):** A diferencia de una simple maqueta tridimensional animada, un Gemelo Digital Cognitivo recibe en tiempo real la telemetría de cada sensor y actualiza continuamente sus modelos físicos y neuronales para reflejar el estado exacto de desgaste de la fábrica real.
- **Turno 3 (Masculina):** Sobre este gemelo virtual los algoritmos de optimización simulan miles de escenarios por minuto para minimizar los picos de consumo eléctrico, reducir el gasto de aire comprimido y balancear dinámicamente la carga entre estaciones.
- **Turno 4 (Femenina):** Así culmina la evolución hacia la Industria cinco punto cero: una ingeniería mecatrónica donde la inteligencia artificial garantiza procesos sostenibles, resilientes ante fallos y diseñados para potenciar el talento y la seguridad del ser humano.

Diapositiva 10: Síntesis de la Parte 4 y Transición al Cierre del Curso

Integración de TinyML, Control Híbrido Neuro-PID, Mantenimiento Predictivo y Robótica Colaborativa

- **Turno 1 (Masculina):** Sinteticemos los logros de ingeniería de esta cuarta parte: hemos aprendido a comprimir y cuantizar redes neuronales a enteros de ocho bits para ejecutarlas con latencia determinista en microcontroladores y aceleradores de borde.
- **Turno 2 (Femenina):** Integramos el aprendizaje neuronal con la teoría de control mediante esquemas híbridos Neuro-PID, prealimentación compensadora de fricción e identificación dinámica NARX acoplada a controladores predictivos.
- **Turno 3 (Masculina):** Finalmente, diseñamos estaciones de inspección visual de alta velocidad, sistemas de mantenimiento predictivo con estimación de vida útil remanente, robots colaborativos con control de impedancia y gemelos digitales cognitivos.
- **Turno 4 (Femenina):** Acompáñanos ahora al siguiente módulo de cierre ejecutivo para integrar en una visión global los cuatro pilares del curso de Inteligencia Artificial en Mecatrónica.

4. Firmware C++/MCU: Kalman, Difuso, TinyML INT8 y Neuro-PID (ia_embebida_kalman_difuso_tinymL_video_04.ino)

```
/*
 * ============================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 04: Video 04 — Parte 4: Automatización Cognitiva, Edge AI y TinyML, Control Inteligente y Mantenimiento
Predictivo
 * Archivo: ia_embebida_kalman_difuso_tinymL_video_04.ino
 * ----------------------------------------------------------------------------
 * Descripción Técnica:
 *   Firmware determinista en C/C++ Embebido para Microcontrolador de 32 bits
 *   (ARM Cortex-M4/M7 STM32 / ESP32-S3 / Teensy 4.1) que integra los 4 pilares
 *   de la Inteligencia Artificial en Mecatrónica en tiempo real:
 *     1. PERCEPCIÓN: Ventaneo temporal deslizante y extracción de descriptores
 *        estadísticos (Valor Eficaz RMS, Varianza y Factor de Cresta).
 *     2. RAZONAMIENTO: Filtro de Kalman 1D recursivo para estimación óptima
 *        de estado y Motor de Inferencia Difusa Takagi-Sugeno (Fuzzy PD).
 *     3. APRENDIZAJE / TINYML: Inferencia neuronal densa cuantizada en 8 bits
 *        (INT8) con activación ReLU para ajuste dinámico de ganancias.
 *     4. AUTOMATIZACIÓN COGNITIVA: Controlador Híbrido Neuro-PID con
 *        Anti-Windup, diagnóstico de salud (HI) y telemetría CRC-16-CCITT.
 *
 * Enfoque del Módulo:
 *   Cuantización INT8 para TinyML (ARM Cortex-M / ESP32-S3), Control Híbrido Neuro-PID + NARX, Estimación RUL y
Robótica Colaborativa
 * ============================================================================
 */

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

// --- Definición de Pines de Hardware (Sensores, Actuador PWM y Alarma) ---
const uint8_t PIN_ADC_SENSOR_PLANTA = A0; // Señal medida PV (Posición / Velocidad)
const uint8_t PIN_ADC_VIBRACION     = A1; // Acelerómetro MEMS para diagnóstico PdM
const uint8_t PIN_DI_HABILITACION   = 2;  // Entrada digital de habilitación segura
const uint8_t PIN_PWM_SERVO         = 9;  // Salida PWM hacia etapa de potencia
const uint8_t PIN_DO_ALARMA_PDM     = 8;  // Salida de alerta de anomalía / RUL

// --- Parámetros Nominales del Módulo (Video 04) ---
const float FS_HZ          = 2000.0f;
const float TS_SEC         = 1.0f / FS_HZ;
const float KALMAN_Q       = 0.0015f;
const float KALMAN_R       = 0.0300f;
const float QUANT_SCALE    = 0.00392f;
const float KP_BASE        = 5.20f;
const float KI_BASE        = 1.80f;
const float KD_BASE        = 0.14f;

// --- 1. Estructura del Estimador Estadístico: Filtro de Kalman 1D ---
struct FiltroKalman1D {
  float x_est; // Estado estimado óptimo
  float p_cov; // Covarianza de incertidumbre del error
  float q_proc;
  float r_meas;
};

FiltroKalman1D g_kalman = {0.0f, 1.0f, KALMAN_Q, KALMAN_R};

float actualizarFiltroKalman(FiltroKalman1D &kf, float medicionZ, float entradaU) {
  // Paso 1: Predicción a priori
  float x_pred = kf.x_est + 0.02f * entradaU * TS_SEC;
  float p_pred = kf.p_cov + kf.q_proc;

  // Paso 2: Corrección óptima de Bayes-Kalman
  float gananciaK = p_pred / (p_pred + kf.r_meas);
  kf.x_est = x_pred + gananciaK * (medicionZ - x_pred);
  kf.p_cov = (1.0f - gananciaK) * p_pred;
  return kf.x_est;
```

```
}

// --- 2. Motor de Inferencia Difusa Takagi-Sugeno (Compensador No Lineal) ---
float evaluarInferenciaDifusaSugeno(float errorNorm, float deltaErrorNorm) {
  // Funciones de pertenencia triangulares para Negativo (N), Cero (Z) y Positivo (P)
  float mu_N = constrain(-errorNorm, 0.0f, 1.0f);
  float mu_P = constrain( errorNorm, 0.0f, 1.0f);
  float mu_Z = constrain(1.0f - fabsf(errorNorm), 0.0f, 1.0f);

  // Consecuentes lineales de Takagi-Sugeno: u_i = p_i * e + q_i * de
  float u_regla_N = -18.0f + 4.0f * errorNorm + 1.5f * deltaErrorNorm;
  float u_regla_Z =   2.5f * errorNorm + 0.8f * deltaErrorNorm;
  float u_regla_P =  18.0f + 4.0f * errorNorm + 1.5f * deltaErrorNorm;

  float sumaPesos = mu_N + mu_Z + mu_P + 1e-6f;
  return (mu_N * u_regla_N + mu_Z * u_regla_Z + mu_P * u_regla_P) / sumaPesos;
}

// --- 3. Inferencia Neuronal TinyML Cuantizada en INT8 (2 Entradas -> 4 Ocultas -> 1 Salida) ---
const int8_t W1_INT8[4][2] = { {42, 18}, {-35, 29}, {51, -24}, {22, 46} };
const int8_t B1_INT8[4]    = { 5, -3, 4, 8 };
const int8_t W2_INT8[4]    = { 38, -27, 44, 31 };
const int8_t B2_INT8       = 6;

float ejecutarInferenciaTinyML_INT8(float e_norm, float de_norm) {
  int8_t x_q[2];
  x_q[0] = (int8_t)constrain((int)roundf(e_norm / QUANT_SCALE), -128, 127);
  x_q[1] = (int8_t)constrain((int)roundf(de_norm / QUANT_SCALE), -128, 127);

  int32_t acumSalida = (int32_t)B2_INT8 * 128;
  for (uint8_t i = 0; i < 4; i++) {
    int32_t z_oculto = (int32_t)B1_INT8[i] * 64 +
                       (int32_t)W1_INT8[i][0] * x_q[0] +
                       (int32_t)W1_INT8[i][1] * x_q[1];
    // Activación ReLU entera
    int32_t h_relu = (z_oculto > 0) ? (z_oculto >> 6) : 0;
    acumSalida += (int32_t)W2_INT8[i] * h_relu;
  }
  // Des-cuantización a factor multiplicador de ganancia [0.70 .. 1.50]
  float deltaGanancia = (float)acumSalida * (QUANT_SCALE * 0.0015f);
  return constrain(1.0f + deltaGanancia, 0.70f, 1.50f);
}

// --- 4. Extracción de Características de Vibración en Ventana (RMS y Salud HI) ---
const uint8_t TAM_VENTANA = 32;
float g_bufferVibracion[TAM_VENTANA];
uint8_t g_idxVentana = 0;

float calcularRmsVentana() {
  float sumaCuadrados = 0.0f;
  for (uint8_t i = 0; i < TAM_VENTANA; i++) {
    sumaCuadrados += g_bufferVibracion[i] * g_bufferVibracion[i];
  }
  return sqrtf(sumaCuadrados / (float)TAM_VENTANA);
}

// --- Cálculo de Verificación 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++) {
      crc = (crc & 0x8000U) ? ((crc << 1) ^ 0x1021U) : (crc << 1);
    }
  }
  return crc;
}
```

```
// --- Variables Globales del Lazo Neuro-PID ---
float    g_integralError = 0.0f;
float    g_errorPrevio   = 0.0f;
float    g_comandoPrevio = 0.0f;
uint32_t g_prevMicros    = 0;
uint32_t g_ciclo         = 0;

void setup() {
  Serial.begin(115200);
  pinMode(PIN_DI_HABILITACION, INPUT_PULLUP);
  pinMode(PIN_PWM_SERVO, OUTPUT);
  pinMode(PIN_DO_ALARMA_PDM, OUTPUT);

  for (uint8_t i = 0; i < TAM_VENTANA; i++) g_bufferVibracion[i] = 0.0f;
  g_prevMicros = micros();

  Serial.println(F("# ================================================================="));
  Serial.println(F("# CURSO IA EN MECATRONICA - VIDEO 04"));
  Serial.println(F("# Columnas: t_s,SP,PV_Kalman,u_NeuroPID,factor_TinyML,vib_RMS,HI,crc16"));
  Serial.println(F("# ================================================================="));
}

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

  if (ahoraMicros - g_prevMicros >= periodoMicros) {
    g_prevMicros += periodoMicros;
    float t_sec = g_ciclo * TS_SEC;

    // 1. Percepción Sensorial y Filtrado Óptimo de Kalman
    float spDeseado = 50.0f + 25.0f * sinf(2.0f * 3.14159f * 0.25f * t_sec);
    float pvCrudo   = (analogRead(PIN_ADC_SENSOR_PLANTA) / 1023.0f) * 100.0f;
    float vibCruda  = ((analogRead(PIN_ADC_VIBRACION) - 512) / 512.0f) * 5.0f;

    g_bufferVibracion[g_idxVentana] = vibCruda;
    g_idxVentana = (g_idxVentana + 1) % TAM_VENTANA;
    float vibRMS = calcularRmsVentana();
    float indiceSaludHI = constrain(1.0f - (vibRMS / 4.5f), 0.0f, 1.0f);
    digitalWrite(PIN_DO_ALARMA_PDM, (indiceSaludHI < 0.35f) ? HIGH : LOW);

    float pvEstimado = actualizarFiltroKalman(g_kalman, pvCrudo, g_comandoPrevio);

    // 2. Razonamiento Difuso + Auto-Sintonización TinyML INT8 + Neuro-PID
    float errorActual = spDeseado - pvEstimado;
    float deltaError  = errorActual - g_errorPrevio;
    float e_norm      = constrain(errorActual / 50.0f, -1.0f, 1.0f);
    float de_norm     = constrain(deltaError / 10.0f, -1.0f, 1.0f);

    float factorTinyML = ejecutarInferenciaTinyML_INT8(e_norm, de_norm);
    float uDifuso      = evaluarInferenciaDifusaSugeno(e_norm, de_norm);

    float kpAdapt = KP_BASE * factorTinyML;
    float kiAdapt = KI_BASE * factorTinyML;
    float kdAdapt = KD_BASE * factorTinyML;

    float candIntegral = g_integralError + kiAdapt * TS_SEC * errorActual;
    float derivTermino = kdAdapt * (deltaError / TS_SEC);
    float uTotalUnsat  = kpAdapt * errorActual + candIntegral + derivTermino + 0.30f * uDifuso;
    float uSaturado    = constrain(uTotalUnsat, 0.0f, 100.0f);

    // Anti-Windup condicional (Clamping)
    if (uTotalUnsat == uSaturado) {
      g_integralError = candIntegral;
    }

    g_errorPrevio   = errorActual;
    g_comandoPrevio = uSaturado;
```

```
    int pwmDuty = constrain((int)((uSaturado / 100.0f) * 255.0f), 0, 255);
    analogWrite(PIN_PWM_SERVO, pwmDuty);

    // 3. Telemetría Estructurada con CRC-16 cada 20 ciclos
    if (g_ciclo % 20 == 0) {
      uint8_t payload[4] = {
        (uint8_t)constrain((int)spDeseado, 0, 255),
        (uint8_t)constrain((int)pvEstimado, 0, 255),
        (uint8_t)pwmDuty,
        (uint8_t)(indiceSaludHI * 100.0f)
      };
      uint16_t crc = calcularCRC16(payload, 4);

      Serial.print(t_sec, 3);          Serial.print(',');
      Serial.print(spDeseado, 2);      Serial.print(',');
      Serial.print(pvEstimado, 2);     Serial.print(',');
      Serial.print(uSaturado, 2);      Serial.print(',');
      Serial.print(factorTinyML, 3);   Serial.print(',');
      Serial.print(vibRMS, 3);         Serial.print(',');
      Serial.print(indiceSaludHI, 3);  Serial.print(",0x");
      Serial.println(crc, HEX);
    }
    g_ciclo++;
  }
}
```

5. Script Python IA: FFT, Planificación A*, Red MLP INT8 y RUL (ia_percepcion_kalman_mlp_y_planificacion_video_04.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
VIDEO 04: Video 04 — Parte 4: Automatización Cognitiva, Edge AI y TinyML, Control Inteligente y Mantenimiento
Predictivo
Archivo: ia_percepcion_kalman_mlp_y_planificacion_video_04.py
--------------------------------------------------------------------------------
Descripción Técnica:
  Script integral en Python para ingeniería mecatrónica cognitiva que
  implementa de forma autocontenida y reproducible:
    1. PERCEPCIÓN: Extracción de descriptores temporales (RMS, Varianza,
       Factor de Cresta, Curtosis) y Transformada Discreta de Fourier (DFT/FFT).
    2. RAZONAMIENTO: Filtro de Kalman 1D, Controlador Difuso Takagi-Sugeno y
       planificador heurístico de trayectorias A* sobre rejilla de ocupación.
    3. APRENDIZAJE Y TINYML: Entrenamiento de un Perceptrón Multicapa (MLP)
       mediante Backpropagation y cuantización simétrica FP32 -> INT8.
    4. AUTOMATIZACIÓN COGNITIVA: Simulación en lazo cerrado Neuro-PID y
       estimación de Vida Útil Remanente (RUL) para mantenimiento predictivo.
================================================================================
"""

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

# Parámetros nominales del Video 04
FS_HZ        = 2000
WINDOW_N     = 512
KALMAN_Q     = 0.0015
KALMAN_R     = 0.03
LR_INIT      = 0.008
QUANT_SCALE  = 0.00392


# ================================================================================
# 1. PERCEPCIÓN SENSORIAL: DESCRIPTORES TEMPORALES Y ESPECTRALES (FFT)
# ================================================================================
def extraer_caracteristicas_vibracion(senal: List[float], fs: float) -> Dict[str, float]:
    """Calcula RMS, Varianza, Factor de Cresta, Curtosis y pico espectral."""
    n = len(senal)
    media = sum(senal) / n
    varianza = sum((x - media) ** 2 for x in senal) / n
    rms = math.sqrt(sum(x * x for x in senal) / n)
    pico = max(abs(x) for x in senal)
    factor_cresta = pico / max(rms, 1e-9)
    curtosis = (sum((x - media) ** 4 for x in senal) / n) / max(varianza ** 2, 1e-12)

    # Evaluación de energía espectral con ventana de Hann en los primeros 32 bines
    mag_max = 0.0
    freq_dominante = 0.0
    for m in range(1, min(32, n // 2)):
        re = 0.0
        im = 0.0
        for k in range(n):
            w_hann = 0.5 * (1.0 - math.cos(2.0 * math.pi * k / (n - 1)))
            ang = 2.0 * math.pi * m * k / n
            re += senal[k] * w_hann * math.cos(ang)
            im -= senal[k] * w_hann * math.sin(ang)
        mag = math.sqrt(re * re + im * im) / n
        if mag > mag_max:
            mag_max = mag
            freq_dominante = (m * fs) / n

    return {
        "RMS": round(rms, 4),
```

```
        "Varianza": round(varianza, 4),
        "Factor_Cresta": round(factor_cresta, 4),
        "Curtosis": round(curtosis, 4),
        "Frecuencia_Dominante_Hz": round(freq_dominante, 2),
        "Amplitud_Pico_FFT": round(mag_max, 4),
    }


# ============================================================================
# 2. RAZONAMIENTO: PLANIFICACIÓN ÓPTIMA DE TRAYECTORIA CON ALGORITMO A*
# ============================================================================
def planificar_ruta_a_estrella(
    mapa_ocupacion: List[List[int]], inicio: Tuple[int, int], meta: Tuple[int, int]
) -> List[Tuple[int, int]]:
    """Encuentra el camino óptimo libre de colisiones minimizando f(n) = g(n) + h(n)."""
    filas, cols = len(mapa_ocupacion), len(mapa_ocupacion[0])
    abiertos = [(0.0, inicio)]
    came_from: Dict[Tuple[int, int], Tuple[int, int]] = {}
    g_score = {inicio: 0.0}

    vecinos = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]

    while abiertos:
        _, actual = heapq.heappop(abiertos)
        if actual == meta:
            ruta = [actual]
            while actual in came_from:
                actual = came_from[actual]
                ruta.append(actual)
            return list(reversed(ruta))

        for dr, dc in vecinos:
            nr, nc = actual[0] + dr, actual[1] + dc
            if 0 <= nr < filas and 0 <= nc < cols and mapa_ocupacion[nr][nc] == 0:
                costo_paso = math.hypot(dr, dc)
                tentativo_g = g_score[actual] + costo_paso
                vecino = (nr, nc)
                if tentativo_g < g_score.get(vecino, float("inf")):
                    came_from[vecino] = actual
                    g_score[vecino] = tentativo_g
                    heuristica_h = math.hypot(meta[0] - nr, meta[1] - nc)
                    heapq.heappush(abiertos, (tentativo_g + heuristica_h, vecino))
    return []


# ============================================================================
# 3. APRENDIZAJE Y TINYML: RED NEURONAL MLP Y CUANTIZACIÓN FP32 -> INT8
# ============================================================================
def entrenar_mlp_y_cuantizar_int8(epocas: int = 120, lr: float = LR_INIT) -> Dict[str, object]:
    """
    Entrena una red neuronal ligera para compensar la fricción no lineal de Stribeck
    y cuantiza los pesos entrenados a enteros de 8 bits (INT8) para TinyML.
    """
    # Datos sintéticos de fricción no lineal: par_friccion = f(velocidad)
    muestras_x = [-1.0 + 0.1 * i for i in range(21)]
    muestras_y = [
        0.45 * (1.0 if v >= 0 else -1.0) * math.exp(-abs(v) / 0.25) + 0.25 * v
        for v in muestras_x
    ]

    # Pesos iniciales deterministas (1 entrada -> 4 neuronas ocultas Tanh -> 1 salida)
    w1 = [0.6, -0.5, 0.8, -0.7]
    b1 = [0.1, -0.1, 0.05, 0.15]
    w2 = [0.5, -0.4, 0.6, -0.5]
    b2 = 0.0

    mse_actual = 0.0
    for _ in range(epocas):
```

```
        mse_acum = 0.0
        for x_val, y_real in zip(muestras_x, muestras_y):
            # Forward pass
            z1 = [w1[i] * x_val + b1[i] for i in range(4)]
            h1 = [math.tanh(z) for z in z1]
            y_pred = sum(w2[i] * h1[i] for i in range(4)) + b2

            err = y_pred - y_real
            mse_acum += err * err

            # Backpropagation (Regla de la cadena)
            d_ypred = 2.0 * err / len(muestras_x)
            for i in range(4):
                grad_w2 = d_ypred * h1[i] + 1e-4 * w2[i]  # Regularización L2
                grad_h1 = d_ypred * w2[i] * (1.0 - h1[i] ** 2)
                w2[i] -= lr * grad_w2
                w1[i] -= lr * (grad_h1 * x_val + 1e-4 * w1[i])
                b1[i] -= lr * grad_h1
            b2 -= lr * d_ypred
        mse_actual = mse_acum / len(muestras_x)

    # Cuantización simétrica FP32 -> INT8
    max_abs = max(max(abs(v) for v in w1), max(abs(v) for v in w2), 1e-6)
    escala_s = max_abs / 127.0
    w1_int8 = [max(-128, min(127, int(round(w / escala_s)))) for w in w1]
    w2_int8 = [max(-128, min(127, int(round(w / escala_s)))) for w in w2]

    return {
        "MSE_Final_Entrenamiento": round(mse_actual, 6),
        "Escala_Cuantizacion_S": round(escala_s, 6),
        "W1_INT8": w1_int8,
        "W2_INT8": w2_int8,
    }


# ==============================================================================
# 4. AUTOMATIZACIÓN COGNITIVA: PRONÓSTICO DE VIDA ÚTIL REMANENTE (RUL)
# ==============================================================================
def estimar_vida_util_remanente_rul(rms_actual: float, rms_nominal: float = 0.45, rms_critico: float = 3.20,
tasa_deg_hora: float = 0.012) -> Dict[str, float]:
    """Calcula el Índice de Salud HI ∈ [0,1] y proyecta la Vida Útil Remanente (RUL) en horas."""
    hi = 1.0 - (rms_actual - rms_nominal) / max(rms_critico - rms_nominal, 1e-6)
    hi_clamped = max(0.0, min(1.0, hi))
    margen_restante = max(0.0, rms_critico - rms_actual)
    rul_horas = margen_restante / max(tasa_deg_hora, 1e-6)
    return {
        "Indice_Salud_HI": round(hi_clamped, 4),
        "RUL_Estimado_Horas": round(rul_horas, 1),
    }


if __name__ == "__main__":
    print("=" * 78)
    print(" CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA — VIDEO 04")
    print(" Enfoque:", "Cuantización INT8 para TinyML (ARM Cortex-M / ESP32-S3), Control Híbrido Neuro-PID + NARX,
Estimación RUL y Robótica Colaborativa")
    print("=" * 78)

    # 1. Prueba de Percepción Espectral
    senal_prueba = [
        0.65 * math.sin(2.0 * math.pi * 120.0 * (k / FS_HZ))
        + 0.20 * math.cos(2.0 * math.pi * 360.0 * (k / FS_HZ))
        for k in range(128)
    ]
    rasgos = extraer_caracteristicas_vibracion(senal_prueba, FS_HZ)
    print("\n[1] Percepción Sensorial (Tiempo y Frecuencia FFT):", rasgos)

    # 2. Prueba de Planificación A* en C-Space
```

```
    rejilla = [
        [0, 0, 0, 1, 0, 0],
        [0, 1, 0, 1, 0, 0],
        [0, 1, 0, 0, 0, 1],
        [0, 0, 1, 1, 0, 0],
        [0, 0, 0, 0, 0, 0],
    ]
    ruta_optima = planificar_ruta_a_estrella(rejilla, (0, 0), (4, 5))
    print("[2] Ruta Óptima Planificada con A*:", ruta_optima)

    # 3. Prueba de Aprendizaje Neuronal y Cuantización TinyML INT8
    modelo_q = entrenar_mlp_y_cuantizar_int8()
    print("[3] Red Neuronal Entrenada y Cuantizada a INT8:", modelo_q)

    # 4. Prueba de Mantenimiento Predictivo (HI y RUL)
    pronostico = estimar_vida_util_remanente_rul(rms_actual=1.15)
    print("[4] Pronóstico de Mantenimiento Predictivo (HI & RUL):", pronostico)
```

6. Programa PLC IEC 61131-3 (ST): Kalman, Takagi-Sugeno, Visión SHRB y PdM (automatizacion_cognitiva_plc_iec61131_video_04.st)

```
(* ================================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 04: Video 04 — Parte 4: Automatización Cognitiva, Edge AI y TinyML, Control Inteligente y Mantenimiento
Predictivo
 * Archivo: automatizacion_cognitiva_plc_iec61131_video_04.st
 * --------------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) para PLC Industrial
 * Descripción Técnica:
 *   Bloque de Función Cognitivo (FB_CeldaMecatronicaCognitiva_V04) que integra:
 *     1. Escalado analógico NORM_X / SCALE_X de sensores de proceso y vibración.
 *     2. Filtro Estadístico de Kalman 1D en tiempo de ciclo Scan para rechazo
 *        de ruido gaussiano en instrumentación de planta.
 *     3. Inferencia Difusa Takagi-Sugeno + Lazo Híbrido Neuro-PID con Anti-Windup.
 *     4. Registro de Corrimiento (SHRB) sincronizado con cámara de Visión IA
 *        y cálculo de Índice de Salud (HI) y Vida Útil Remanente (RUL).
 * ================================================================================ *)

FUNCTION_BLOCK FB_CeldaMecatronicaCognitiva_V04
VAR_INPUT
    (* Entradas Digitales de Campo (%IX) *)
    xMarchaCelda_NO        : BOOL; (* %IX0.0 - Orden de Marcha *)
    xParoEmergencia_NC     : BOOL; (* %IX0.1 - Seguridad Físicamente NC *)
    xVisionPiezaDefecto    : BOOL; (* %IX0.2 - Inferencia CNN en Borde: 1 = Defecto *)
    xPulsoEncoderBanda     : BOOL; (* %IX0.3 - Sincronismo de Banda Transportadora *)

    (* Entradas Analógicas Crudas 0..27648 Cuentas (%IW) *)
    iSensorProcesoRaw      : INT;  (* %IW64  - Variable de Proceso (4-20 mA) *)
    iSensorVibracionRaw    : INT;  (* %IW66  - Transmisor de Vibración RMS (mm/s) *)
    rSetpointIngenieria    : REAL := 65.0;
END_VAR

VAR_OUTPUT
    (* Salidas Digitales hacia Planta (%QX) *)
    xContactorBanda        : BOOL; (* %QX0.0 - Motor Principal de Línea *)
    xValvulaExpulsoraIA    : BOOL; (* %QX0.1 - Rechazo de Calidad en Estación 5 *)
    xAlertaPredictivaRUL   : BOOL; (* %QX0.2 - Alerta Temprana de Mantenimiento *)

    (* Salidas Analógicas y Diagnóstico Cognitivo hacia SCADA / Gemelo Digital *)
    iSalidaActuadorRaw     : INT;  (* %QW80  - Mando 0..27648 hacia Servoválvula/VFD *)
    rEstadoKalmanPV        : REAL; (* Estado filtrado óptimo por Kalman *)
    rComandoNeuroPID       : REAL; (* Esfuerzo de control 0.0 a 100.0 % *)
    rIndiceSaludHI         : REAL; (* Health Index normalizado [0.0 .. 1.0] *)
    rVidaUtilRemanenteH    : REAL; (* RUL estimado en horas de operación *)
END_VAR

VAR
    (* Parámetros del Estimador de Kalman 1D *)
    rKalmanX               : REAL := 0.0;
    rKalmanP               : REAL := 1.0;
    rKalmanQ               : REAL := 0.0015;
    rKalmanR               : REAL := 0.0300;
    rGananciaKalmanK       : REAL := 0.0;

    (* Variables de Escalado y Diagnóstico Predictivo *)
    rPvMedidaBruta         : REAL := 0.0;
    rVibracionRmsMmS       : REAL := 0.0;

    (* Registro de Corrimiento de 16 Bits para Inspección Visual IA *)
    wShiftRechazoVision    : WORD := 16#0000;
    xEncoderPrev           : BOOL := FALSE;
    xFlancoEncoder         : BOOL := FALSE;

    (* Memoria del Controlador Híbrido Difuso + Neuro-PID *)
    rKpBase                : REAL := 5.20;
    rKiBase                : REAL := 1.80;
    rKdBase                : REAL := 0.14;
    rTsScanSeg             : REAL := 0.010;
```

```
    rIntegralAcum          : REAL := 0.0;
    rErrorPrev             : REAL := 0.0;
END_VAR

(* ================================================================================
 * 1. ENCLAVAMIENTO DE SEGURIDAD Y ESCALADO ANALÓGICO (NORM_X / SCALE_X)
 * ================================================================================ *)
xContactorBanda := (xMarchaCelda_NO OR xContactorBanda) AND xParoEmergencia_NC;

rPvMedidaBruta   := (INT_TO_REAL(iSensorProcesoRaw) / 27648.0) * 100.0;
rVibracionRmsMmS := (INT_TO_REAL(iSensorVibracionRaw) / 27648.0) * 10.0;

(* ================================================================================
 * 2. FILTRO DE KALMAN 1D RECURSIVO (PREDICCIÓN + CORRECCIÓN BAYESIANA)
 * ================================================================================ *)
rKalmanP         := rKalmanP + rKalmanQ;
rGananciaKalmanK := rKalmanP / (rKalmanP + rKalmanR);
rKalmanX         := rKalmanX + rGananciaKalmanK * (rPvMedidaBruta - rKalmanX);
rKalmanP         := (1.0 - rGananciaKalmanK) * rKalmanP;
rEstadoKalmanPV  := rKalmanX;

(* ================================================================================
 * 3. MANTENIMIENTO PREDICTIVO (CÁLCULO DE HEALTH INDEX HI Y RUL EN HORAS)
 * ================================================================================ *)
rIndiceSaludHI := 1.0 - ((rVibracionRmsMmS - 0.5) / 4.0);
IF rIndiceSaludHI > 1.0 THEN rIndiceSaludHI := 1.0; END_IF;
IF rIndiceSaludHI < 0.0 THEN rIndiceSaludHI := 0.0; END_IF;

rVidaUtilRemanenteH := rIndiceSaludHI * 850.0;
xAlertaPredictivaRUL := (rIndiceSaludHI < 0.30);

(* ================================================================================
 * 4. SINCRONISMO DE VISIÓN ARTIFICIAL DE ALTA VELOCIDAD CON SHRB
 * ================================================================================ *)
xFlancoEncoder := xPulsoEncoderBanda AND NOT xEncoderPrev;
xEncoderPrev   := xPulsoEncoderBanda;

IF xFlancoEncoder AND xContactorBanda THEN
    wShiftRechazoVision := SHL(IN := wShiftRechazoVision, N := 1);
    IF xVisionPiezaDefecto THEN
        wShiftRechazoVision := wShiftRechazoVision OR 16#0001;
    END_IF;
END_IF;

xValvulaExpulsoraIA := (wShiftRechazoVision AND 16#0010) <> 16#0000;

(* ================================================================================
 * 5. CONTROLADOR HÍBRIDO DIFUSO TAKAGI-SUGENO + NEURO-PID AUTO-SINTONIZABLE
 * ================================================================================ *)
IF xContactorBanda THEN
    VAR
        rError      : REAL;
        rDeltaError : REAL;
        rFactorIA   : REAL;
        rCompDifusa : REAL;
        rCandI      : REAL;
        rSumaUnsat  : REAL;
    END_VAR
    rError      := rSetpointIngenieria - rEstadoKalmanPV;
    rDeltaError := rError - rErrorPrev;

    (* Factor de adaptación neuronal en función de la magnitud del error *)
    rFactorIA   := 1.0 + 0.012 * ABS(rError);
    IF rFactorIA > 1.45 THEN rFactorIA := 1.45; END_IF;

    (* Consecuente de Takagi-Sugeno para amortiguamiento no lineal *)
    rCompDifusa := 0.25 * rError + 0.40 * rDeltaError;
```

```
    rCandI     := rIntegralAcum + (rKiBase * rFactorIA) * rTsScanSeg * rError;
    rSumaUnsat := (rKpBase * rFactorIA) * rError
                  + rCandI
                  + (rKdBase * rFactorIA) * (rDeltaError / rTsScanSeg)
                  + rCompDifusa;

    IF rSumaUnsat > 100.0 THEN
        rComandoNeuroPID := 100.0;
    ELSIF rSumaUnsat < 0.0 THEN
        rComandoNeuroPID := 0.0;
    ELSE
        rComandoNeuroPID := rSumaUnsat;
        rIntegralAcum    := rCandI;
    END_IF;

    rErrorPrev := rError;
ELSE
    rComandoNeuroPID := 0.0;
    rIntegralAcum    := 0.0;
END_IF;

iSalidaActuadorRaw := REAL_TO_INT((rComandoNeuroPID / 100.0) * 27648.0);

END_FUNCTION_BLOCK
```