CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA

Video 03 — Parte 3: Aprendizaje en Mecatrónica, Machine Learning, Redes Neuronales y Aprendizaje por Refuerzo — Optimización por Gradiente, SVM, Random Forest, PCA, Perceptrón Multicapa, CNN/LSTM, MDP y Actor-Crítico

Parte del Curso: Parte 3 (Tema 1C) — Aprendizaje en Mecatrónica: Machine Learning, Redes Neuronales y Aprendizaje por Refuerzo
Arquitectura Cognitiva / Algoritmo Estudiado: Optimización por Gradiente, SVM, Random Forest, PCA, Perceptrón Multicapa (MLP), Redes CNN/LSTM y Políticas Actor-Crítico

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla J(\theta) - \lambda \cdot \theta \mid h = \mathrm{ReLU}(W_1 \cdot x + b_1) \mid Q^*(s,a) = E[\, r + \gamma \cdot \max_{a'} Q^*(s',a') \,]$$

1. Síntesis de Conceptos Clave y Reglas de Ingeniería Cognitiva

- Paradigmas de aprendizaje en mecatrónica (Supervisado, No Supervisado y por Refuerzo) y formulación matemática mediante minimización de la función de costo J(θ) por descenso de gradiente / Adam con regularización L1 y L2.
- Aprendizaje supervisado y no supervisado en planta: Regresión Polinomial para sensores virtuales (Soft Sensors), Máquinas de Soporte Vectorial (SVM con Kernel RBF), Árboles de Decisión, Random Forest, agrupamiento K-Means y reducción dimensional PCA para detección de anomalías.
- Redes Neuronales Artificiales y Deep Learning: el Perceptrón Multicapa (MLP) con activaciones ReLU/Tanh y algoritmo Backpropagation; Redes Convolucionales (CNN) para visión artificial y Redes Recurrentes (LSTM) para series temporales dinámicas.
- Aprendizaje por Refuerzo en robótica: Procesos de Decisión de Markov (MDP), diseño de funciones de recompensa, Ecuación de Bellman, algoritmo Q-Learning y políticas continuas Actor-Crítico (DDPG, PPO) con transferencia Sim-to-Real.

2. Parámetros Nominales de Configuración de IA y Control (Video 03)

Parámetro del Sistema Cognitivo / Controlador	Símbolo	Valor Nominal	Función en Percepción, Razonamiento, Aprendizaje o Automatización
Frecuencia de Muestreo y Ventana FFT	`fs [Hz] / N`	1000 Hz, N = 256	Adquisición determinista para descriptores RMS, Curtosis y espectro FFT
Covarianzas del Filtro de Kalman	`Q / R`	Q = 0.003, R = 0.06	Ponderación óptima entre incertidumbre del modelo dinámico y ruido del sensor
Tasa de Aprendizaje por Gradiente	`η (Learning Rate)`	0.02	Paso de actualización de pesos sinápticos en Backpropagation / Adam
Factor de Escala de Cuantización TinyML	`S (FP32 -> INT8)`	0.00781	Compresión de pesos y activaciones a enteros de 8 bits [-128, 127]
Ganancias Base del Lazo Neuro-PID	`[Kp0, Ki0, Kd0]`	[3.6, 1.1, 0.08]	Ganancias nominales moduladas en línea por la red neuronal y el bloque difuso

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: ¿Qué es el Aprendizaje en Ingeniería Mecatrónica?

De los Modelos Físicos Analíticos al Aprendizaje Supervisado, No Supervisado y por Refuerzo

- **Turno 1 (Masculina):** En los módulos anteriores diseñamos sistemas cuyas reglas o ecuaciones dinámicas eran escritas explícitamente por el ingeniero; pero ¿qué ocurre cuando la fricción cambiante, el desgaste o la complejidad visual hacen imposible deducir una fórmula exacta?
- **Turno 2 (Femenina):** Aquí interviene el tercer pilar de nuestro curso: el Aprendizaje de Máquina o Machine Learning, donde el sistema mecatrónico ajusta automáticamente sus parámetros internos a partir de datos experimentales de planta para mejorar su desempeño.
- **Turno 3 (Masculina):** Clasificamos el aprendizaje en tres grandes paradigmas: primero, el Aprendizaje Supervisado, donde entrenamos un modelo con pares de entrada y salida etiquetados para predecir variables continuas o clasificar estados de falla.
- **Turno 4 (Femenina):** Segundo, el Aprendizaje No Supervisado, que descubre agrupamientos y anomalías ocultas en los sensores sin necesidad de etiquetas previas; y tercero, el Aprendizaje por Refuerzo, donde un robot aprende a controlarse interactuando por ensayo y error guiado por recompensas.

Diapositiva 02: Formulación Matemática del Aprendizaje de Máquina

Función de Costo J(θ), Descenso de Gradiente, Optimizador Adam y Regularización L1/L2 contra Sobreajuste

- **Turno 1 (Masculina):** Matemáticamente, todo algoritmo de aprendizaje en mecatrónica se formula como un problema de optimización numérica cuyo objetivo es encontrar el vector de pesos teta que minimiza una función de costo jota de teta.
- **Turno 2 (Femenina):** En problemas de regresión e identificación de sistemas empleamos el Error Cuadrático Medio entre la salida medida en la máquina y la predicción del modelo, mientras que en clasificación de defectos minimizamos la Entropía Cruzada.
- **Turno 3 (Masculina):** El motor que ajusta los pesos paso a paso es el Descenso de Gradiente: en cada iteración calculamos el vector de derivadas parciales de la función de costo y desplazamos los parámetros en dirección opuesta al gradiente multiplicado por la tasa de aprendizaje eta.
- **Turno 4 (Femenina):** Para evitar que el modelo memorice el ruido particular del conjunto de entrenamiento, fenómeno conocido como sobreajuste u Overfitting, añadimos un término de regularización ele uno o ele dos que penaliza los pesos excesivamente grandes y garantiza capacidad de generalización.

Diapositiva 03: Aprendizaje Supervisado en Sensores y Actuadores

Regresión Polinomial para Sensores Virtuales (Soft Sensors) y Máquinas de Soporte Vectorial (SVM)

- **Turno 1 (Masculina):** Una de las aplicaciones más rentables del aprendizaje supervisado en mecatrónica es el desarrollo de Sensores Virtuales o Soft Sensors: modelos de regresión que estiman variables difíciles de medir físicamente a partir de sensores económicos.
- **Turno 2 (Femenina):** Por ejemplo, mediante regresión polinomial regularizada podemos estimar en tiempo real la temperatura interna del rotor o el par de carga en el eje combinando las lecturas de corriente estatórica, tensión de bus y velocidad angular.
- **Turno 3 (Masculina):** Cuando la tarea consiste en clasificar el régimen de operación de una máquina con muestras limitadas, las Máquinas de Soporte Vectorial, abreviadas ese ve eme, constituyen uno de los clasificadores más sólidos de la ingeniería.
- **Turno 4 (Femenina):** Una máquina de soporte vectorial busca el hiperplano óptimo que separa las clases maximizando el margen geométrico de seguridad a los puntos más cercanos, utilizando funciones Kernel de base radial para separar fronteras altamente no lineales.

Diapositiva 04: Árboles de Decisión y Ensambles Random Forest en Diagnóstico

Partición por Impureza de Gini / Entropía de Shannon, Bagging e Importancia de Características

- **Turno 1 (Masculina):** En la industria manufacturera valoramos enormemente los modelos explicables y veloces: los Árboles de Decisión cumplen ambos requisitos al dividir el espacio de características sensoriales mediante comparaciones secuenciales de umbral.
- **Turno 2 (Femenina):** Durante el entrenamiento, el algoritmo selecciona en cada nodo la variable física y el valor de corte que logran la máxima reducción en la impureza de Gini o la mayor ganancia de información según la entropía de Shannon.
- **Turno 3 (Masculina):** Una ventaja extraordinaria para la mecatrónica es que un árbol de decisión entrenado en Python se traduce directamente a una estructura de condicionales anidados en lenguaje C o en Texto Estructurado de P L C que se ejecuta en nanosegundos.
- **Turno 4 (Femenina):** Y para alcanzar la máxima precisión diagnóstica combinamos decenas de árboles independientes entrenados por muestreo aleatorio en un ensamble Random Forest, capaz de clasificar fallas combinadas de rodamientos, engranajes y excentricidad sin sufrir sobreajuste.

Diapositiva 05: Aprendizaje No Supervisado y Detección de Anomalías en Planta

Agrupamiento K-Means, Análisis de Componentes Principales (PCA) y Error de Reconstrucción

- **Turno 1 (Masculina):** En la gran mayoría de las plantas industriales disponemos de millones de horas de datos de operación sana, pero no contamos con muestras previas de cada posible avería catastrófica; aquí resulta indispensable el Aprendizaje No Supervisado.
- **Turno 2 (Femenina):** Mediante el algoritmo de agrupamiento K-Means identificamos automáticamente los distintos regímenes normales de trabajo de una máquina agrupando los vectores de sensores alrededor de ka centroides que minimizan la distancia cuadrática interna.
- **Turno 3 (Masculina):** Cuando una celda cuenta con decenas de canales correlacionados de vibración, presión y corriente, aplicamos el Análisis de Componentes Principales, abreviado pe ce a, que proyecta los datos sobre las direcciones ortogonales de máxima varianza.
- **Turno 4 (Femenina):** El pe ce a y los bosques de aislamiento permiten detectar anomalías incipientes al instante: cualquier degradación mecánica que rompa la correlación física habitual entre sensores dispara el error de reconstrucción mucho antes de que salte un umbral fijo.

Diapositiva 06: Redes Neuronales Artificiales y Perceptrón Multicapa (MLP)

Arquitectura Densa, Funciones de Activación (ReLU, Sigmoide, Tanh) y Algoritmo Backpropagation

- **Turno 1 (Masculina):** Para aproximar funciones continuas de arbitraria complejidad, como la cinemática inversa de un robot redundante o la dinámica térmica no lineal de un proceso, empleamos Redes Neuronales Artificiales.
- **Turno 2 (Femenina):** En un Perceptrón Multicapa, abreviado eme ele pe, cada neurona calcula la suma ponderada de sus entradas más un sesgo be, y pasa el resultado por una función de activación no lineal.
- **Turno 3 (Masculina):** En las capas ocultas modernas predomina la Unidad Lineal Rectificada o ReLU, que retorna cero para valores negativos y mantiene la identidad para valores positivos, evitando el desvanecimiento del gradiente junto a las funciones Tangente Hiperbólica y Sigmoide.
- **Turno 4 (Femenina):** El aprendizaje de todas las capas ocurre mediante la Retropropagación del Error o Backpropagation, que aplica de forma recursiva la regla de la cadena del cálculo diferencial desde la capa de salida hacia atrás para actualizar cada peso sináptico.

Diapositiva 07: Aprendizaje Profundo (Deep Learning): Redes CNN y LSTM

Redes Convolucionales para Visión Mecatrónica y Redes Recurrentes con Compuertas de Memoria Temporal

- **Turno 1 (Masculina):** El Aprendizaje Profundo o Deep Learning revolucionó la percepción mecatrónica al reemplazar el diseño manual de descriptores por arquitecturas profundas que aprenden jerarquías de características directamente desde los datos crudos.
- **Turno 2 (Femenina):** En visión artificial industrial utilizamos Redes Neuronales Convolucionales, abreviadas ce ene ene: sus primeras capas aprenden a detectar bordes y texturas simples, las capas intermedias reconocen geometrías mecánicas y las capas finales localizan piezas y defectos con precisión submilimétrica.
- **Turno 3 (Masculina):** Por otro lado, cuando procesamos señales dinámicas con memoria histórica, como corrientes de servomotores o degradación vibratoria, una red estática resulta insuficiente porque el orden temporal importa.
- **Turno 4 (Femenina):** Para ello empleamos Redes Recurrentes de Memoria a Corto y Largo Plazo, conocidas como ele ese te eme, cuyas compuertas internas de olvido, entrada y salida deciden qué información pasada retener o descartar a lo largo de cientos de instantes de muestreo.

Diapositiva 08: Aprendizaje por Refuerzo (Reinforcement Learning) en Robótica

Procesos de Decisión de Markov (MDP), Diseño de la Función de Recompensa y Ecuación de Bellman

- **Turno 1 (Masculina):** El paradigma más cercano al control automático humano es el Aprendizaje por Refuerzo: aquí nadie le dice al robot cuál es el comando exacto de torque en cada milisegundo, sino que el agente descubre la política óptima interactuando con su entorno.
- **Turno 2 (Femenina):** Formalizamos esta interacción como un Proceso de Decisión de Markov: en cada paso discreto te, el controlador observa el vector de estado ese sub te, aplica una acción de control a sub te sobre los actuadores y la planta evoluciona hacia un nuevo estado.
- **Turno 3 (Masculina):** Tras cada transición, el agente recibe una señal escalar de Recompensa erre sub te diseñada por el ingeniero: por ejemplo, premiar la cercanía al objetivo y penalizar el exceso de corriente eléctrica, las sacudidas bruscas o las colisiones.
- **Turno 4 (Femenina):** El objetivo matemático del agente es hallar la política pi que maximiza la suma descontada de recompensas futuras, gobernada por la Ecuación de Optimalidad de Bellman sobre la función de valor estado-acción cu estrella.

Diapositiva 09: De Q-Learning a Políticas Actor-Crítico (PPO / DDPG) y Sim-to-Real

Control Continuo de Actuadores mediante Redes Actor-Crítico y Transferencia del Simulador al Robot Físico

- **Turno 1 (Masculina):** En problemas con estados y acciones discretas, el algoritmo Q-Learning actualiza iterativamente una tabla o una red neuronal profunda con el valor esperado de cada par estado-acción hasta converger a la estrategia óptima.
- **Turno 2 (Femenina):** Sin embargo, los servomotores y actuadores mecatrónicos requieren comandos continuos de tensión, corriente o par; para gobernarlos empleamos arquitecturas de Gradiente de Política del tipo Actor-Crítico, como de de pe ge y pe pe o.
- **Turno 3 (Masculina):** En un sistema Actor-Crítico operan dos redes neuronales en equipo: la red Actor genera directamente la acción continua de control a partir del estado medido, mientras la red Crítico evalúa qué tan buena fue esa acción para guiar la mejora del Actor.
- **Turno 4 (Femenina):** Para no dañar la máquina real durante los millones de ensayos iniciales, entrenamos estas políticas en simuladores físicos acelerados aplicando aleatorización de masas y fricciones, logrando transferir la red entrenada directamente al robot físico real.

Diapositiva 10: Síntesis de la Parte 3 y Transición al Siguiente Módulo

Criterios de Selección de Algoritmos de Aprendizaje y Paso hacia la Automatización Cognitiva

- **Turno 1 (Masculina):** Recapitulemos el recorrido de esta tercera parte: comprendimos cómo el aprendizaje de máquina optimiza funciones de costo mediante descenso de gradiente con regularización para extraer modelos directamente desde los datos de planta.
- **Turno 2 (Femenina):** Dominamos las técnicas supervisadas como regresión, máquinas de soporte vectorial y Random Forest para sensores virtuales y clasificación de fallas, así como los métodos no supervisados K-Means y pe ce a para detección temprana de anomalías.
- **Turno 3 (Masculina):** Exploramos además las redes neuronales profundas convolucionales y recurrentes, llegando hasta las políticas Actor-Crítico de aprendizaje por refuerzo para el control autónomo continuo.
- **Turno 4 (Femenina):** En el siguiente módulo, correspondiente a la Parte cuatro, llevaremos todos estos modelos al campo industrial real, estudiando la Automatización Cognitiva con TinyML, control híbrido Neuro-PID, mantenimiento predictivo y robótica colaborativa.

4. Firmware C++/MCU: Kalman, Difuso, TinyML INT8 y Neuro-PID (ia_embebida_kalman_difuso_tinymL_video_03.ino)

```
/*
 * ================================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 03: Video 03 — Parte 3: Aprendizaje en Mecatrónica, Machine Learning, Redes Neuronales y Aprendizaje por Refuerzo
 * Archivo: ia_embebida_kalman_difuso_tinymL_video_03.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:
 *   Optimización por Gradiente, SVM, Random Forest, PCA, Perceptrón Multicapa (MLP), Redes CNN/LSTM y Políticas Actor-Crítico
 * ================================================================================
 */

#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 03) ---
const float FS_HZ          = 1000.0f;
const float TS_SEC         = 1.0f / FS_HZ;
const float KALMAN_Q       = 0.0030f;
const float KALMAN_R       = 0.0600f;
const float QUANT_SCALE    = 0.00781f;
const float KP_BASE        = 3.60f;
const float KI_BASE        = 1.10f;
const float KD_BASE        = 0.08f;

// --- 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 03"));
  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_03.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
VIDEO 03: Video 03 – Parte 3: Aprendizaje en Mecatrónica, Machine Learning, Redes Neuronales y Aprendizaje por
Refuerzo
Archivo: ia_percepcion_kalman_mlp_y_planificacion_video_03.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 03
FS_HZ        = 1000
WINDOW_N     = 256
KALMAN_Q     = 0.003
KALMAN_R     = 0.06
LR_INIT      = 0.02
QUANT_SCALE  = 0.00781


# ==============================================================================
# 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 03")
    print(" Enfoque:", "Optimización por Gradiente, SVM, Random Forest, PCA, Perceptrón Multicapa (MLP), Redes
CNN/LSTM y Políticas Actor-Crítico")
    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_03.st)

```
(* ================================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 03: Video 03 — Parte 3: Aprendizaje en Mecatrónica, Machine Learning, Redes Neuronales y Aprendizaje por
Refuerzo
 * Archivo: automatizacion_cognitiva_plc_iec61131_video_03.st
 * --------------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) para PLC Industrial
 * Descripción Técnica:
 *   Bloque de Función Cognitivo (FB_CeldaMecatronicaCognitiva_V03) 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_V03
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 := 0.0030;
    rKalmanR               : REAL := 0.0600;
    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 := 3.60;
    rKiBase                : REAL := 1.10;
    rKdBase                : REAL := 0.08;
    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
```