CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA

Video 01 — Parte 1: ¿Qué Significa IA en Mecatrónica?, Percepción Sensorial, Visión 2D/3D y Cognición — Agentes Racionales PEAS, Ciclo Percepción-Cognición-Acción, FFT, Visión Convolucional, LiDAR y SLAM

Parte del Curso: Parte 1 (Tema 1A) — ¿Qué Significa IA en Mecatrónica?, Percepción Sensorial, Visión 2D/3D y Cognición
Arquitectura Cognitiva / Algoritmo Estudiado: Agente Racional PEAS, Extracción de Características (RMS, Factor de Cresta, FFT), Convolución 2D Sobel/Canny, LiDAR y Fusión Sensorial

$$X[m] = \Sigma\, x[n]\cdot w[n]\cdot e^{(-j2\pi mn/N)} \mid s\cdot[u, v, 1]^T = K\cdot[R \mid t]\cdot[Xw, Yw, Zw, 1]^T \mid Z = (f \cdot B) / d$$

1. Síntesis de Conceptos Clave y Reglas de Ingeniería Cognitiva

- Definición de IA en Mecatrónica mediante agentes racionales físicos, arquitectura PEAS y convergencia entre IA Simbólica, IA Conexionista e IA Física Embebida en el ciclo Percepción-Cognición-Acción.
- Acondicionamiento y ventaneo deslizante (Hann/Hamming) para extraer descriptores en el dominio del tiempo (RMS, varianza, factor de cresta, curtosis) y en el dominio de la frecuencia mediante la Transformada Rápida de Fourier (FFT).
- Percepción visual artificial: calibración intrínseca K y extrínseca [R|t] del modelo Pinhole, filtrado gaussiano por convolución 2D, detección de bordes Sobel y Canny, segmentación de Otsu y momentos invariantes de Hu.
- Percepción espacial 3D con cámaras estereoscópicas, RGB-D ToF y nubes de puntos LiDAR (registro ICP), combinadas mediante fusión multisensorial ponderada por varianza inversa y Localización y Mapeo Simultáneos (SLAM).

2. Parámetros Nominales de Configuración de IA y Control (Video 01)

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 = 256	Adquisición determinista para descriptores RMS, Curtosis y espectro FFT
Covarianzas del Filtro de Kalman	`Q / R`	Q = 0.002, R = 0.05	Ponderación óptima entre incertidumbre del modelo dinámico y ruido del sensor
Tasa de Aprendizaje por Gradiente	`η (Learning Rate)`	0.01	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]`	[4, 1.4, 0.1]	Ganancias nominales moduladas en línea por la red neuronal y el bloque difuso

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: ¿Qué Significa Inteligencia Artificial en Mecatrónica?

Agentes Racionales Corporizados y Modelo PEAS (Rendimiento, Entorno, Actuadores y Sensores)

- **Turno 1 (Masculina):** ¿Qué significa realmente hablar de Inteligencia Artificial dentro de la ingeniería mecatrónica? A diferencia del software puramente digital, en mecatrónica la inteligencia reside en un agente racional físico que interactúa directamente con la materia y la energía.
- **Turno 2 (Femenina):** Definimos un agente mecatrónico racional como aquel sistema autónomo que percibe el estado de su entorno mediante transductores y ejecuta acciones físicas a través de actuadores para maximizar matemáticamente una medida de rendimiento.
- **Turno 3 (Masculina):** Para diseñar cualquier agente inteligente utilizamos la arquitectura P E A S: primero definimos el Rendimiento, como minimizar el error de trayectoria y el consumo energético; y segundo caracterizamos el Entorno de trabajo y sus perturbaciones.
- **Turno 4 (Femenina):** Los dos elementos restantes del modelo P E A S son el hardware mecatrónico: los Actuadores, como servomotores, válvulas proporcionales o pinzas robóticas, y los Sensores, como encoders, cámaras industriales, unidades inerciales y escáneres láser.

Diapositiva 02: Evolución de los Paradigmas de IA en Sistemas Físicos

IA Simbólica Basada en Reglas, IA Conexionista (Redes Neuronales) e IA Física Embebida

- **Turno 1 (Masculina):** El desarrollo de la Inteligencia Artificial aplicada a las máquinas ha atravesado tres grandes paradigmas complementarios que hoy convergen dentro de los controladores industriales modernos.
- **Turno 2 (Femenina):** El primer paradigma es la Inteligencia Artificial Simbólica, basada en reglas lógicas explícitas y sistemas expertos: destaca por su total transparencia y determinismo en el diagnóstico de fallas, pero fracasa ante datos sensoriales ruidosos no estructurados.
- **Turno 3 (Masculina):** El segundo paradigma es la Inteligencia Artificial Conexionista, impulsada por el aprendizaje estadístico y las redes neuronales artificiales, capaz de descubrir patrones complejos directamente a partir de imágenes, vibraciones mecánicas y series temporales.
- **Turno 4 (Femenina):** El tercer paradigma es la Inteligencia Artificial Física Embebida, donde los algoritmos cognitivos se acoplan estrechamente con las leyes de la dinámica de Newton, respetando límites térmicos, saturación de actuadores y plazos estrictos de tiempo real.

Diapositiva 03: El Ciclo Percepción–Cognición–Acción en Tiempo Real

Arquitectura Jerárquica Multitasa desde el Muestreo de Sensores hasta el Lazo Servo de Actuación

- **Turno 1 (Masculina):** Toda máquina mecatrónica autónoma opera gobernada por el ciclo continuo de Percepción, Cognición y Acción, estructurado en tres niveles jerárquicos que se ejecutan a diferentes frecuencias de reloj.
- **Turno 2 (Femenina):** En la etapa de Percepción, el sistema adquiere señales físicas crudas de múltiples transductores, filtra el ruido electrónico y extrae un vector compacto de características que describe qué está ocurriendo en la planta en ese instante.
- **Turno 3 (Masculina):** En la etapa de Cognición, el procesador fusiona esas observaciones para actualizar su modelo interno del mundo, evalúa el estado de salud de la máquina y planifica la trayectoria o consigna óptima cumpliendo las restricciones de seguridad.
- **Turno 4 (Femenina):** Finalmente, en la etapa de Acción, esa consigna se entrega a los controladores de bajo nivel que cierran los lazos de corriente, velocidad y posición a frecuencias de kilohertz sobre los servomotores y etapas de potencia.

Diapositiva 04: Percepción Sensorial Mecatrónica y Extracción de Características

Acondicionamiento, Ventaneo Deslizante y Descriptores en Dominio del Tiempo y Frecuencia (FFT)

- **Turno 1 (Masculina):** Un modelo de inteligencia artificial jamás debe alimentarse con muestras analógicas crudas contaminadas por ruido; la percepción sensorial comienza siempre con un filtrado antisolapamiento y un muestreo periódico de alta fidelidad.
- **Turno 2 (Femenina):** Dado que una sola muestra aislada carece de contexto dinámico, agrupamos la señal en ventanas temporales deslizantes de longitud fija, por ejemplo de doscientas cincuenta y seis muestras con un solapamiento del cincuenta por ciento.
- **Turno 3 (Masculina):** Sobre cada ventana calculamos descriptores estadísticos en el dominio del tiempo de bajo costo computacional: el valor eficaz o erre eme ese, la varianza, el valor pico a pico, el factor de cresta y la curtosis de impacto.
- **Turno 4 (Femenina):** Simultáneamente, aplicamos una ventana de Hann y ejecutamos la Transformada Rápida de Fourier, abreviada efe efe te, para extraer el espectro de frecuencias e identificar armónicos exactos de desbalanceo, resonancia estructural o desgaste de rodamientos.

Diapositiva 05: Percepción Visual Artificial: Modelo Pinhole y Convolución 2D

Calibración Intrínseca y Extrínseca de Cámara y Filtrado Espacial por Núcleos Matriciales

- **Turno 1 (Masculina):** Para que un robot industrial pueda manipular una pieza guiado por visión, necesitamos transformar las coordenadas bidimensionales en píxeles de la imagen en coordenadas métricas tridimensionales expresadas en milímetros.
- **Turno 2 (Femenina):** El modelo geométrico Pinhole resuelve esta proyección mediante dos matrices: la matriz intrínseca ka, que contiene las distancias focales y el centro óptico del sensor, y la matriz extrínseca de rotación y traslación respecto a la base del robot.
- **Turno 3 (Masculina):** Tras corregir la distorsión radial de la lente mediante calibración con patrón de ajedrez, aplicamos la operación matemática fundamental de la visión artificial: la convolución discreta bidimensional.
- **Turno 4 (Femenina):** La convolución desliza una pequeña matriz de pesos llamada núcleo o kernel sobre cada píxel de la imagen, permitiendo atenuar el ruido electrónico de alta frecuencia mediante un filtro gaussiano antes de buscar contornos.

Diapositiva 06: Detección de Bordes, Segmentación y Reconocimiento de Objetos

Operadores de Gradiente Sobel, Algoritmo Multietapa de Canny, Umbral de Otsu y Momentos Geométricos

- **Turno 1 (Masculina):** En una celda mecatrónica de manufactura, las fronteras físicas de las piezas mecánicas se manifiestan en la imagen como cambios bruscos de intensidad luminosa, es decir, como máximos del gradiente espacial.
- **Turno 2 (Femenina):** Mediante los núcleos direccionales de Sobel calculamos las derivadas parciales horizontal ge sub equis y vertical ge sub ye, obteniendo en cada píxel la magnitud total del borde y su ángulo exacto de orientación.
- **Turno 3 (Masculina):** Para obtener contornos limpios de un solo píxel de espesor empleamos el detector óptimo de Canny, el cual aplica supresión de no-máximos en la dirección del gradiente y una histéresis de doble umbral que elimina trazos falsos.
- **Turno 4 (Femenina):** Combinando la segmentación automática de Otsu con el cálculo de momentos geométricos espaciales e invariantes de Hu, el controlador determina en pocos milisegundos el área, el centroide exacto y el ángulo de giro de cada pieza en la banda.

Diapositiva 07: Percepción Espacial 3D en Robótica y Automatización

Estereoscopía Binocular, Cámaras de Profundidad RGB-D, Nubes de Puntos LiDAR y Odometría

- **Turno 1 (Masculina):** Cuando un manipulador colaborativo o un vehículo autónomo de planta navega en un entorno tridimensional dinámico, una imagen plana resulta insuficiente y requerimos percepción directa de profundidad.
- **Turno 2 (Femenina):** En la visión estéreo binocular comparamos dos cámaras separadas por una línea base be: al medir la disparidad en píxeles de entre ambas imágenes, obtenemos la profundidad zeta dividiendo el producto de la distancia focal y la línea base entre la disparidad.
- **Turno 3 (Masculina):** Las cámaras erre ge be de industriales complementan el color con sensores de Tiempo de Vuelo infrarrojo, entregando en cada fotograma un mapa denso de distancias para el agarre robótico de piezas apiladas al azar.
- **Turno 4 (Femenina):** A escala de planta, los escáneres láser LiDAR generan nubes de cientos de miles de puntos tridimensionales por segundo, que al filtrarse en grillas de vóxeles y alinearse mediante el algoritmo iterativo de punto más cercano permiten medir desplazamientos con precisión milimétrica.

Diapositiva 08: Cognición Mecatrónica y Fusión Multisensorial

Complementariedad entre Encoders, IMU MEMS, Visión y LiDAR mediante Ponderación Óptima

- **Turno 1 (Masculina):** La transición desde la simple percepción hacia la verdadera cognición mecatrónica ocurre mediante la Fusión Sensorial: el arte matemático de combinar múltiples sensores imperfectos para obtener una estimación superior a cualquiera de ellos por separado.
- **Turno 2 (Femenina):** Observemos por qué ningún sensor basta por sí solo: los encoders de las ruedas sufren deslizamiento sobre el piso, el giroscopio acumula deriva a baja frecuencia, el acelerómetro es ruidoso a alta frecuencia y las cámaras sufren cambios de iluminación.
- **Turno 3 (Masculina):** En la fusión de actitud inercial, un filtro complementario combina el paso bajo del acelerómetro con la integración en paso alto del giroscopio, entregando un ángulo de inclinación limpio, sin deriva y sin retardo de fase.
- **Turno 4 (Femenina):** De manera general, cuando fusionamos mediciones independientes ponderando cada sensor de forma inversamente proporcional a su varianza de ruido sigma cuadrada, reducimos matemáticamente la incertidumbre global del estado estimado.

Diapositiva 09: Modelo Interno del Mundo y Localización y Mapeo Simultáneos (SLAM)

Conciencia Situacional, Rejillas de Ocupación Probabilística y Cierre de Bucle (Loop Closure)

- **Turno 1 (Masculina):** El nivel más elevado de la cognición perceptiva consiste en construir y mantener actualizado en memoria un Modelo Interno del Mundo que otorgue a la máquina conciencia situacional sobre su entorno de trabajo.
- **Turno 2 (Femenina):** En robótica móvil industrial, este modelo suele representarse como una Rejilla de Ocupación Probabilística, donde el espacio se divide en celdas milimétricas y cada celda almacena la probabilidad actualizada de estar libre u ocupada por un obstáculo.
- **Turno 3 (Masculina):** Cuando el vehículo entra en una nave desconocida se enfrenta al problema dual de Localización y Mapeo Simultáneos, conocido como SLAM: necesita un mapa preciso para calcular su pose, pero necesita conocer su pose para construir el mapa.
- **Turno 4 (Femenina):** Los algoritmos modernos de SLAM visual y láser resuelven esta paradoja integrando la odometría local en un grafo de poses y corrigiendo toda la deriva acumulada en el instante en que el robot reconoce un lugar visitado previamente, proceso denominado cierre de bucle.

Diapositiva 10: Síntesis de la Parte 1 y Transición al Siguiente Módulo

Del Acondicionamiento y Extracción Espectral/Visual a la Toma de Decisiones Racional

- **Turno 1 (Masculina):** Sinteticemos los fundamentos de ingeniería de esta primera parte: hemos definido a los sistemas mecatrónicos con inteligencia artificial como agentes racionales físicos estructurados bajo el modelo P E A S y el ciclo Percepción, Cognición y Acción.
- **Turno 2 (Femenina):** Comprobamos cómo transformar señales crudas en vectores de características mediante ventaneo temporal y la Transformada Rápida de Fourier, así como extraer geometrías bidimensionales y tridimensionales mediante visión artificial y nubes de puntos LiDAR.
- **Turno 3 (Masculina):** Finalmente, vimos cómo la fusión multisensorial y los algoritmos de mapeo simultáneo SLAM integran todas estas fuentes en un modelo interno coherente y libre de deriva.
- **Turno 4 (Femenina):** En el siguiente módulo, correspondiente a la Parte dos del curso, dotaremos a este agente de capacidad de Razonamiento mediante sistemas expertos, controladores de lógica difusa, inferencia bayesiana con Filtro de Kalman y planificación óptima de trayectorias.

4. Firmware C++/MCU: Kalman, Difuso, TinyML INT8 y Neuro-PID (ia_embebida_kalman_difuso_tinymL_video_01.ino)

```
/*
 * ============================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 01: Video 01 — Parte 1: ¿Qué Significa IA en Mecatrónica?, Percepción Sensorial, Visión 2D/3D y Cognición
 * Archivo: ia_embebida_kalman_difuso_tinymL_video_01.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:
 *   Agente Racional PEAS, Extracción de Características (RMS, Factor de Cresta, FFT), Convolución 2D Sobel/Canny,
LiDAR y Fusión Sensorial
 * ============================================================================
 */

#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 01) ---
const float FS_HZ          = 2000.0f;
const float TS_SEC         = 1.0f / FS_HZ;
const float KALMAN_Q       = 0.0020f;
const float KALMAN_R       = 0.0500f;
const float QUANT_SCALE    = 0.00781f;
const float KP_BASE        = 4.00f;
const float KI_BASE        = 1.40f;
const float KD_BASE        = 0.10f;

// --- 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 01"));
  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_01.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
VIDEO 01: Video 01 — Parte 1: ¿Qué Significa IA en Mecatrónica?, Percepción Sensorial, Visión 2D/3D y Cognición
Archivo: ia_percepcion_kalman_mlp_y_planificacion_video_01.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 01
FS_HZ        = 2000
WINDOW_N     = 256
KALMAN_Q     = 0.002
KALMAN_R     = 0.05
LR_INIT      = 0.01
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 01")
    print(" Enfoque:", "Agente Racional PEAS, Extracción de Características (RMS, Factor de Cresta, FFT),
Convolución 2D Sobel/Canny, LiDAR y Fusión Sensorial")
    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_01.st)

```
(* ============================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 01: Video 01 — Parte 1: ¿Qué Significa IA en Mecatrónica?, Percepción Sensorial, Visión 2D/3D y Cognición
 * Archivo: automatizacion_cognitiva_plc_iec61131_video_01.st
 * ----------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) para PLC Industrial
 * Descripción Técnica:
 *   Bloque de Función Cognitivo (FB_CeldaMecatronicaCognitiva_V01) 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_V01
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 := 0.0020;
    rKalmanR               : REAL := 0.0500;
    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 := 4.00;
    rKiBase                : REAL := 1.40;
    rKdBase                : REAL := 0.10;
    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
```