CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA

Video 02 — Parte 2: Razonamiento en Mecatrónica, Lógica Difusa, Inferencia Bayesiana y Planificación — Sistemas Expertos, Control Difuso Mamdani/Sugeno, Filtro de Kalman Extendido (EKF) y Algoritmos A* y RRT

Parte del Curso: Parte 2 (Tema 1B) — Razonamiento en Mecatrónica: Lógica Difusa, Inferencia Bayesiana, Kalman y Planificación
Arquitectura Cognitiva / Algoritmo Estudiado: Sistemas Expertos, Controlador Difuso Takagi-Sugeno / Mamdani (Fuzzy PD/PID), Filtro de Kalman Extendido (EKF) y Planificador A*

$$u_Fuzzy = (\Sigma\ \mu_i \cdot u_i) / (\Sigma\ \mu_i) \mid K_k = P_\{k|k\text{-}1\} \cdot H\hat{}T \cdot (H \cdot P_\{k|k\text{-}1\} \cdot H\hat{}T + R)\hat{}(-1) \mid f(n) = g(n) + h(n)$$

1. Síntesis de Conceptos Clave y Reglas de Ingeniería Cognitiva

- Separación entre Base de Conocimiento y Motor de Inferencia en sistemas expertos de diagnóstico industrial utilizando Encadenamiento hacia Adelante (guiado por datos) y hacia Atrás (guiado por hipótesis).
- Lógica Difusa de Zadeh: conjuntos difusos, funciones de pertenencia triangulares/trapezoidales/gaussianas $\mu(x) \in [0,1]$, inferencia Mamdani con defusificación por centroide e inferencia rápida de Takagi-Sugeno.
- Diseño de controladores difusos Fuzzy PD/PID sobre el plano de fase (error e[k] y cambio de error Δe[k]) para compensar plantas mecatrónicas no lineales con fricción seca, histéresis o zona muerta.
- Razonamiento probabilístico con el Teorema de Bayes, estimación óptima de estados ocultos con el Filtro de Kalman Lineal y Extendido (EKF), y planificación de trayectorias en el Espacio de Configuraciones (C-Space) mediante campos potenciales, Dijkstra, A* y árboles aleatorios RRT.

2. Parámetros Nominales de Configuración de IA y Control (Video 02)

Parámetro del Sistema Cognitivo / Controlador	Símbolo	Valor Nominal	Función en Percepción, Razonamiento, Aprendizaje o Automatización
Frecuencia de Muestreo y Ventana FFT	`fs [Hz] / N`	500 Hz, N = 128	Adquisición determinista para descriptores RMS, Curtosis y espectro FFT
Covarianzas del Filtro de Kalman	`Q / R`	Q = 0.001, R = 0.04	Ponderación óptima entre incertidumbre del modelo dinámico y ruido del sensor
Tasa de Aprendizaje por Gradiente	`η (Learning Rate)`	0.012	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]`	[4.5, 1.5, 0.12]	Ganancias nominales moduladas en línea por la red neuronal y el bloque difuso

3. Desarrollo Académico Diapositiva por Diapositiva

Diapositiva 01: Razonamiento en Mecatrónica y Representación del Conocimiento

Base de Hechos, Base de Reglas Lógicas, Lógica de Predicados y Motores de Inferencia Deterministas

- **Turno 1 (Masculina):** Una vez que el sistema mecatrónico ha percibido y estructurado el estado de su entorno, entra en acción el segundo gran pilar de la inteligencia artificial: el Razonamiento automatizado.
- **Turno 2 (Femenina):** Razonar en ingeniería significa procesar formalmente el conocimiento disponible junto con las observaciones actuales de los sensores para deducir diagnósticos, evaluar restricciones operativas y decidir la acción más segura y eficiente.
- **Turno 3 (Masculina):** En una arquitectura basada en conocimiento separamos estrictamente tres bloques: la Memoria de Trabajo, que almacena los hechos actuales medidos en planta; la Base de Conocimiento, que contiene las reglas de ingeniería; y el Motor de Inferencia.
- **Turno 4 (Femenina):** Esta separación resulta vital en la industria: permite actualizar las reglas de diagnóstico o seguridad de una celda de manufactura sin necesidad de reescribir el núcleo del programa de control en el microcontrolador o en el P L C.

Diapositiva 02: Sistemas Expertos para Diagnóstico y Supervisión Industrial

Encadenamiento hacia Adelante (Forward Chaining) frente a Encadenamiento hacia Atrás (Backward Chaining)

- **Turno 1 (Masculina):** Los Sistemas Expertos capturan el conocimiento acumulado por los ingenieros de mantenimiento y control en reglas condicionales del tipo SI ocurre una premisa ENTONCES ejecuta una conclusión, operando bajo dos estrategias de búsqueda.
- **Turno 2 (Femenina):** La primera estrategia es el Encadenamiento hacia Adelante, guiado por los datos: el motor lee en cada ciclo los síntomas medidos, como sobrecorriente estatórica y alta temperatura en el rodamiento, y dispara en cascada las reglas hasta activar una alarma o reducir la carga.
- **Turno 3 (Masculina):** La segunda estrategia es el Encadenamiento hacia Atrás, guiado por objetivos: cuando una línea se detiene por una falla desconocida, el motor parte de una hipótesis de avería y retrocede verificando qué subcondiciones eléctricas o neumáticas la explican.
- **Turno 4 (Femenina):** En la supervisión mecatrónica moderna combinamos ambos métodos: el encadenamiento hacia adelante vigila en tiempo real las variables críticas, mientras el encadenamiento hacia atrás aísla al instante la causa raíz exacta para el equipo técnico.

Diapositiva 03: Razonamiento Aproximado con Lógica Difusa (Fuzzy Logic)

Conjuntos Difusos de Zadeh, Variables Lingüísticas y Funciones de Pertenencia Continuas μ(x)

- **Turno 1 (Masculina):** La lógica booleana clásica impone fronteras rígidas de todo o nada que resultan antinaturales en los sistemas físicos continuos: un motor no pasa de estar frío a estar sobrecalentado por cruzar una centésima de grado.
- **Turno 2 (Femenina):** Para razonar con transiciones graduales empleamos la Lógica Difusa, formulada por Lotfi Zadeh, donde cada variable física puede pertenecer parcialmente a varios conjuntos al mismo tiempo con un grado de pertenencia miu comprendido entre cero y uno.
- **Turno 3 (Masculina):** Definimos así variables lingüísticas, como el Error de Velocidad o la Derivada del Error, y las particionamos en etiquetas comprensibles: Negativo Grande, Negativo Pequeño, Cero, Positivo Pequeño y Positivo Grande.
- **Turno 4 (Femenina):** Cada etiqueta se modela matemáticamente mediante una función de pertenencia triangular, trapezoidal o gaussiana, permitiendo que una lectura numérica active simultáneamente dos reglas vecinas para producir una transición suave y libre de saltos.

Diapositiva 04: Motores de Inferencia Difusa Mamdani y Takagi-Sugeno

Evaluación de Reglas IF-THEN, Operadores T-Norma (Mínimo/Producto) y Defusificación por Centroide

- **Turno 1 (Masculina):** El procesamiento dentro de un sistema de inferencia difusa se ejecuta en tres fases deterministas: fusificación de las entradas medidas, evaluación paralela de la matriz de reglas y defusificación hacia el actuador.
- **Turno 2 (Femenina):** En la evaluación de reglas combinamos las premisas mediante operadores T-Norma: el conectivo lógico Y se calcula tomando el valor mínimo o el producto algebraico de los grados de pertenencia, mientras el conectivo O emplea el valor máximo.
- **Turno 3 (Masculina):** En el método clásico de Mamdani, los consecuentes de las reglas son también conjuntos difusos que se recortan, se agregan en una envolvente común y se convierten en un comando numérico exacto calculando el Centro de Gravedad o Centroide del área.
- **Turno 4 (Femenina):** Por su parte, en el método de Takagi-Sugeno el consecuente de cada regla es una función lineal o una constante directa, calculando la salida final como un promedio ponderado ultrarrápido ideal para microcontroladores de bajos recursos.

Diapositiva 05: Diseño de un Controlador Difuso (Fuzzy PD / PID) en Mecatrónica

Matriz de Reglas en el Plano de Fase (Error vs. Cambio de Error) y Superficie de Control No Lineal

- **Turno 1 (Masculina):** ¿Cuándo supera un controlador difuso a un regulador pe i de convencional? Precisamente cuando la planta mecatrónica presenta fuertes no linealidades, como fricción seca cambiante, zona muerta en válvulas hidráulicas o cargas variables.
- **Turno 2 (Femenina):** Un controlador difuso tipo pe de recibe como entradas el error actual e de ka y el incremento del error delta e de ka: si el error es Positivo Grande y el cambio es Cero, la regla ordena aplicar un esfuerzo Positivo Grande para acelerar al máximo.
- **Turno 3 (Masculina):** Pero si el error es Positivo Pequeño y el cambio del error es Negativo Grande, significa que la masa se acerca muy rápido a la meta; la regla aplica entonces un esfuerzo Negativo para frenar con anticipación y evitar el sobreimpulso.
- **Turno 4 (Femenina):** Al graficar las veinticinco reglas de una tabla de cinco por cinco obtenemos una superficie de control tridimensional no lineal, que ofrece alta ganancia lejos de la referencia y un amortiguamiento suave alrededor del punto de equilibrio.

Diapositiva 06: Razonamiento Probabilístico bajo Incertidumbre: Teorema de Bayes

Actualización Recursiva de Creencias (Prior, Verosimilitud y Posterior) y Redes Bayesianas Causales

- **Turno 1 (Masculina):** En los sistemas mecatrónicos reales la incertidumbre aleatoria es inevitable: los sensores ultrasónicos producen ecos espurios, las cámaras sufren reflejos y un mismo síntoma vibratorio puede provenir de distintas causas mecánicas.
- **Turno 2 (Femenina):** El fundamento matemático para razonar bajo incertidumbre es el Teorema de Bayes, el cual calcula la probabilidad posterior de un estado oculto equis dado que hemos observado una medición sensorial zeta.
- **Turno 3 (Masculina):** La ecuación de Bayes multiplica el conocimiento previo o Prior pe de equis por la verosimilitud física del sensor pe de zeta dado equis, y lo normaliza entre la probabilidad marginal de la observación.
- **Turno 4 (Femenina):** Al extender este principio a grafos acíclicos dirigidos construimos Redes Bayesianas de diagnóstico industrial, capaces de inferir la probabilidad exacta de fallo en un inversor, una bomba o un rodamiento a partir de múltiples lecturas parciales.

Diapositiva 07: Estimación Óptima Recursiva: Filtro de Kalman Lineal y Extendido (EKF)

Ciclo Predicción-Corrección, Matriz de Covarianza P, Ganancia Óptima de Kalman K y Linealización Jacobiana

- **Turno 1 (Masculina):** Cuando aplicamos la inferencia bayesiana de forma continua a sistemas dinámicos descritos en variables de estado con ruido gaussiano, obtenemos el algoritmo más célebre de la ingeniería de control y robótica: el Filtro de Kalman.
- **Turno 2 (Femenina):** El Filtro de Kalman opera como un observador estadístico óptimo en dos pasos por cada periodo de muestreo: primero, en la fase de Predicción, utiliza el modelo matemático de la planta y el comando del actuador para proyectar el estado futuro y su covarianza de incertidumbre pe.
- **Turno 3 (Masculina):** Segundo, en la fase de Corrección, calcula la Ganancia de Kalman ka, la cual pondera de manera óptima cuánto confiar en el modelo dinámico frente a cuánto confiar en la nueva medición ruidosa del sensor.
- **Turno 4 (Femenina):** Para sistemas robóticos con trigonometría no lineal empleamos el Filtro de Kalman Extendido, abreviado e ka efe, el cual linealiza las ecuaciones en cada instante mediante matrices de derivadas parciales o Jacobianos.

Diapositiva 08: Razonamiento Espacial y Planificación en el Espacio de Configuraciones

Transformación al C-Space (C_free vs. C_obs), Dilatación de Obstáculos y Campos Potenciales Artificiales

- **Turno 1 (Masculina):** ¿Cómo razona espacialmente un brazo robótico de seis ejes o un vehículo autónomo para desplazarse desde su postura inicial hasta la meta sin chocar contra ningún obstáculo de la celda?
- **Turno 2 (Femenina):** La clave matemática consiste en transformar el problema desde el espacio cartesiano tridimensional hacia el Espacio de Configuraciones, denominado C-Space, donde cada eje corresponde a una coordenada articular cu del robot.
- **Turno 3 (Masculina):** En el Espacio de Configuraciones, el robot completo, sin importar su geometría compleja, se reduce a un único punto móvil, mientras que los obstáculos físicos se expanden formando regiones prohibidas llamadas ce sub o be ese.
- **Turno 4 (Femenina):** Para la evasión reactiva en tiempo real podemos superponer un Campo Potencial Artificial sobre el espacio libre: la meta genera un potencial atractivo cuadrático que guía al robot, mientras cada obstáculo genera una fuerza repulsiva que impide el contacto.

Diapositiva 09: Algoritmos de Búsqueda y Planificación Óptima: Dijkstra, A* y RRT

Función de Costo Heurístico f(n) = g(n) + h(n) en Grafos y Árboles Aleatorios de Exploración Rápida

- **Turno 1 (Masculina):** Como los campos potenciales locales pueden quedar atrapados en mínimos locales frente a obstáculos cóncavos, los robots autónomos emplean planificadores globales sobre grafos y muestreo probabilístico.
- **Turno 2 (Femenina):** Mientras el algoritmo clásico de Dijkstra explora uniformemente en todas las direcciones buscando el camino más corto, el algoritmo heurístico A estrella dirige la búsqueda directamente hacia la meta minimizando la suma efe de ene igual a ge de ene más hache de ene.
- **Turno 3 (Masculina):** En esta ecuación, ge de ene representa el costo exacto acumulado desde el nodo de partida, y hache de ene es la heurística admisible, como la distancia euclidiana restante hasta el objetivo, logrando encontrar la ruta óptima en una fracción del tiempo.
- **Turno 4 (Femenina):** Por último, cuando planificamos el movimiento de un manipulador industrial con muchos grados de libertad, utilizamos Árboles Aleatorios de Exploración Rápida, conocidos como erre erre te, que muestrean configuraciones libres de colisión en pocos milisegundos.

Diapositiva 10: Síntesis de la Parte 2 y Transición al Siguiente Módulo

Integración de Lógica Difusa, Inferencia de Kalman y Planificación de Trayectorias

- **Turno 1 (Masculina):** Resumamos las herramientas de razonamiento que hemos incorporado en esta segunda parte del curso: aprendimos a estructurar el conocimiento experto en motores de inferencia con encadenamiento hacia adelante y hacia atrás.
- **Turno 2 (Femenina):** Diseñamos controladores de lógica difusa Mamdani y Takagi-Sugeno capaces de gobernar plantas mecatrónicas altamente no lineales mediante funciones de pertenencia continuas y matrices de reglas en el plano de fase.
- **Turno 3 (Masculina):** Asimismo, dominamos la inferencia bayesiana recursiva a través del Filtro de Kalman Extendido para estimar estados ocultos bajo ruido, y los algoritmos A estrella y erre erre te para planificar trayectorias libres de colisión.
- **Turno 4 (Femenina):** En el siguiente módulo, correspondiente a la Parte tres, daremos el salto hacia el Aprendizaje en Mecatrónica: máquinas que optimizan sus propios modelos mediante aprendizaje supervisado, no supervisado, redes neuronales profundas y aprendizaje por refuerzo.

4. Firmware C++/MCU: Kalman, Difuso, TinyML INT8 y Neuro-PID (ia_embebida_kalman_difuso_tinymL_video_02.ino)

```
/*
 * ============================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 02: Video 02 — Parte 2: Razonamiento en Mecatrónica, Lógica Difusa, Inferencia Bayesiana y Planificación
 * Archivo: ia_embebida_kalman_difuso_tinymL_video_02.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:
 *   Sistemas Expertos, Controlador Difuso Takagi-Sugeno / Mamdani (Fuzzy PD/PID), Filtro de Kalman Extendido
(EKF) y Planificador A*
 * ============================================================================
 */

#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 02) ---
const float FS_HZ          = 500.0f;
const float TS_SEC         = 1.0f / FS_HZ;
const float KALMAN_Q       = 0.0010f;
const float KALMAN_R       = 0.0400f;
const float QUANT_SCALE    = 0.00781f;
const float KP_BASE        = 4.50f;
const float KI_BASE        = 1.50f;
const float KD_BASE        = 0.12f;

// --- 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 02"));
  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_02.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
VIDEO 02: Video 02 – Parte 2: Razonamiento en Mecatrónica, Lógica Difusa, Inferencia Bayesiana y Planificación
Archivo: ia_percepcion_kalman_mlp_y_planificacion_video_02.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 02
FS_HZ        = 500
WINDOW_N     = 128
KALMAN_Q     = 0.001
KALMAN_R     = 0.04
LR_INIT      = 0.012
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 02")
    print(" Enfoque:", "Sistemas Expertos, Controlador Difuso Takagi-Sugeno / Mamdani (Fuzzy PD/PID), Filtro de
Kalman Extendido (EKF) y Planificador A*")
    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_02.st)

```
(* ============================================================================
 * CURSO #11: INTELIGENCIA ARTIFICIAL EN MECATRÓNICA
 * VIDEO 02: Video 02 — Parte 2: Razonamiento en Mecatrónica, Lógica Difusa, Inferencia Bayesiana y Planificación
 * Archivo: automatizacion_cognitiva_plc_iec61131_video_02.st
 * ----------------------------------------------------------------------------
 * Estándar: IEC 61131-3 — Texto Estructurado (ST) para PLC Industrial
 * Descripción Técnica:
 *   Bloque de Función Cognitivo (FB_CeldaMecatronicaCognitiva_V02) 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_V02
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 := 0.0010;
    rKalmanR               : REAL := 0.0400;
    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 := 4.50;
    rKiBase                : REAL := 1.50;
    rKdBase                : REAL := 0.12;
    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
```