💻 CÓDIGO Y ALGORITMOS DE CONTROL MECATRÓNICO — MÓDULO 07

Curso: Curso Completo: Introducción a la Mecatrónica

Módulo 07: Módulo 07: Control Automático sin Misterios: Lazo Abierto, Lazo Cerrado y PID Intuitivo

Archivos en `codigo_fuente/`:

- `01_firmware_arduino_modulo_07.ino` (Control en tiempo real sobre Arduino UNO)
- `02_supervision_mecatronica_modulo_07.py` (Supervisión HMI y registro CSV/SQLite en Raspberry Pi)
- `03_pruebas_y_calibracion_modulo_07.sh` (Script automatizado de validación de banco)

---

1. Arquitectura de Control y Flujo de Señales (2200×1350)



Plano 2 — Arquitectura de Control y Cinemática del Módulo 07

---

2. Firmware Completo de Tiempo Real (`01_firmware_arduino_modulo_07.ino`)

```
/*
 * ============================================================================
 * CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 07
 * Archivo: 01_firmware_arduino_modulo_07.ino
 * Prototipo: Regulador PID Discreto en Tiempo Real (dt = 50 ms) con Anti-Windup
 * Hardware: Arduino UNO + Sensor HC-SR04 (TRIG=5, ECHO=6)
 *           + Potenciómetro Setpoint (A0) + Actuador Servo/PWM (Pin 9)
 * ============================================================================
 */

#include <Arduino.h>
#include <Servo.h>

const uint8_t PIN_POT_SP      = A0; // Consigna de distancia (10..30 cm)
const uint8_t PIN_TRIG        = 5;  // Disparo ultrasónico
const uint8_t PIN_ECHO        = 6;  // Eco ultrasónico
const uint8_t PIN_LED_ESTABLE = 8;  // Indicador |error| < 0.5 cm
const uint8_t PIN_ACTUADOR    = 9;  // Actuador de corrección mecánica

// --- Ganancias Sintonizables del Controlador PID ---
float Kp = 4.50; // Acción Proporcional (El Presente)
float Ki = 0.80; // Acción Integral (El Pasado: elimina error residual)
float Kd = 1.20; // Acción Derivativa (El Futuro: amortigua rebotes)

const unsigned long PERIODO_PID_MS = 50;   // 20 Hz exactos
const float DT_SEG                 = 0.05; // 50 ms expresados en segundos

Servo actuadorControl;

float setpointCm       = 20.0;
float distanciaMedida  = 20.0;
float errorAnterior    = 0.0;
float sumaIntegral     = 0.0;
bool usarSetpointLocal = true;

unsigned long ultimoCicloPidMs = 0;

float medirDistanciaCm() {
  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  unsigned long duracionUs = pulseIn(PIN_ECHO, HIGH, 25000UL);
  if (duracionUs == 0) {
    return distanciaMedida; // Mantener última lectura válida si no hay eco
  }
  float cm = (duracionUs * 0.0343) / 2.0;
  return constrain(cm, 4.0, 50.0);
}

void procesarSintoniaSerie() {
  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    cmd.toUpperCase();
```

```
    if (cmd.startsWith("KP:")) {
      Kp = cmd.substring(3).toFloat();
    } else if (cmd.startsWith("KI:")) {
      Ki = cmd.substring(3).toFloat();
      sumaIntegral = 0.0;
    } else if (cmd.startsWith("KD:")) {
      Kd = cmd.substring(3).toFloat();
    } else if (cmd.startsWith("SP:")) {
      setpointCm = constrain(cmd.substring(3).toFloat(), 10.0, 30.0);
      usarSetpointLocal = false;
    } else if (cmd == "AUTO") {
      usarSetpointLocal = true;
    }
  }
}

void setup() {
  Serial.begin(115200);

  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
  pinMode(PIN_LED_ESTABLE, OUTPUT);

  actuadorControl.attach(PIN_ACTUADOR);
  actuadorControl.write(90); // Posición neutra de equilibrio (90°)

  Serial.println(F("{"modulo":7,"evento":"CONTROLADOR_PID_INICIADO"}"));
}

void loop() {
  procesarSintoniaSerie();

  unsigned long ahora = millis();
  if (ahora - ultimoCicloPidMs >= PERIODO_PID_MS) {
    ultimoCicloPidMs = ahora;

    if (usarSetpointLocal) {
      int adc = analogRead(PIN_POT_SP);
      setpointCm = 10.0 + (adc / 1023.0) * 20.0; // Rango 10.0 a 30.0 cm
    }

    // Filtro suavizado exponencial (EMA) para medición limpia
    float nuevaLectura = medirDistanciaCm();
    distanciaMedida = (0.7 * distanciaMedida) + (0.3 * nuevaLectura);

    // ========================================================================
    // LAS 5 LÍNEAS FUNDAMENTALES DEL ALGORITMO PID DISCRETO
    // ========================================================================
    float error = setpointCm - distanciaMedida;
    float P     = Kp * error;
    sumaIntegral = constrain(sumaIntegral + (error * DT_SEG), -40.0, 40.0); // Anti-Windup
    float I     = Ki * sumaIntegral;
    float D     = Kd * ((error - errorAnterior) / DT_SEG);

    float esfuerzoU = constrain(90.0 + P + I + D, 15.0, 165.0);
    errorAnterior   = error;

    actuadorControl.write((int)esfuerzoU);
```

```
    digitalWrite(PIN_LED_ESTABLE, (fabs(error) < 0.5) ? HIGH : LOW);

    // Salida JSON para supervisión en Raspberry Pi
    Serial.print(F("{"modulo":7,"sp_cm":"));
    Serial.print(setpointCm, 2);
    Serial.print(F(","med_cm":"));
    Serial.print(distanciaMedida, 2);
    Serial.print(F(","err_cm":"));
    Serial.print(error, 2);
    Serial.print(F(","P":"));
    Serial.print(P, 1);
    Serial.print(F(","I":"));
    Serial.print(I, 1);
    Serial.print(F(","D":"));
    Serial.print(D, 1);
    Serial.print(F(","u_out":"));
    Serial.print(esfuerzoU, 1);
    Serial.println(F("}"));
  }
}
```

---

3. Software de Supervisión y Telemetría (`02_supervision_mecatronica_modulo_07.py`)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 07
Archivo: 02_supervision_mecatronica_modulo_07.py
Descripción: Analizador de respuesta transitoria PID en Raspberry Pi.
             Registra Setpoint, Medición, Error y las componentes P, I, D
             para evaluar sobreimpulso y error en estado estacionario.
================================================================================
"""

import csv
import json
import os
import sys
import time
from datetime import datetime

try:
    import serial
except ImportError:
    serial = None

PUERTO_SERIE = os.environ.get("MECATRONICA_PORT", "/dev/ttyACM0")
BAUDIOS      = 115200
ARCHIVO_CSV  = "respuesta_pid_modulo_07.csv"


def inicializar_csv(ruta: str) -> None:
    if not os.path.exists(ruta):
        with open(ruta, mode="w", newline="", encoding="utf-8") as f:
            csv.writer(f).writerow(["timestamp", "sp_cm", "med_cm", "err_cm", "P", "I", "D",
"u_out"])


def procesar_muestra_pid(d: dict) -> None:
    ts  = datetime.now().strftime("%H:%M:%S.%f")[:-3]
    sp  = d.get("sp_cm", 20.0)
    med = d.get("med_cm", 20.0)
    err = d.get("err_cm", 0.0)
    vp  = d.get("P", 0.0)
    vi  = d.get("I", 0.0)
    vd  = d.get("D", 0.0)
    u   = d.get("u_out", 90.0)

    estable = "✅ ESTABLE" if abs(err) < 0.5 else "⚡ CORRIGIENDO"
    print(
        f"[{ts}] SP:{sp:5.2f}cm | Med:{med:5.2f}cm | Err:{err:+5.2f}cm | "
        f"P:{vp:+5.1f} I:{vi:+4.1f} D:{vd:+5.1f} -> u:{u:5.1f}° | {estable}"
    )
    with open(ARCHIVO_CSV, mode="a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow([ts, sp, med, err, vp, vi, vd, u])


def ejecutar_demo() -> None:
    print("ℹ️ Simulando respuesta al escalón y rechazo de perturbaciones con Controlador PID...")
```

```
    curva = [
        {"sp_cm": 20.0, "med_cm": 12.00, "err_cm": +8.00, "P": +36.0, "I": +1.6, "D": +12.0,
"u_out": 139.6},
        {"sp_cm": 20.0, "med_cm": 17.50, "err_cm": +2.50, "P": +11.2, "I": +2.1, "D": -8.4,
"u_out": 94.9},
        {"sp_cm": 20.0, "med_cm": 19.85, "err_cm": +0.15, "P": +0.7,  "I": +2.2, "D": -1.8,
"u_out": 91.1},
        {"sp_cm": 20.0, "med_cm": 20.00, "err_cm":  0.00, "P":  0.0,  "I": +2.2, "D":  0.0,
"u_out": 92.2},
    ]
    for m in curva:
        procesar_muestra_pid(m)
        time.sleep(0.35)


def main() -> None:
    print("=" * 78)
    print(" ANALIZADOR DE CONTROL AUTOMÁTICO MÓDULO 07 — SINTONÍA PID EN TIEMPO REAL")
    print("=" * 78)
    inicializar_csv(ARCHIVO_CSV)
    if "--demo" in sys.argv or serial is None:
        ejecutar_demo()
        return
    try:
        with serial.Serial(PUERTO_SERIE, BAUDIOS, timeout=1.0) as ser:
            while True:
                linea = ser.readline().decode("utf-8", errors="ignore").strip()
                if linea.startswith("{") and "err_cm" in linea:
                    procesar_muestra_pid(json.loads(linea))
    except Exception as e:
        print(f"⚠️ Sin conexión serie ({e}). Ejecutando simulación de lazo PID:")
        ejecutar_demo()


if __name__ == "__main__":
    main()
```