💻 CÓDIGO Y ALGORITMOS DE CONTROL MECATRÓNICO — MÓDULO 08

Curso: Curso Completo: Introducción a la Mecatrónica

Módulo 08: Módulo 08: Proyecto Integrador Final: Cinta Clasificadora y Brazo Robótico con Supervisión

Archivos en `codigo_fuente/`:

- `01_firmware_arduino_modulo_08.ino` (Control en tiempo real sobre Arduino UNO)
- `02_supervision_mecatronica_modulo_08.py` (Supervisión HMI y registro CSV/SQLite en Raspberry Pi)
- `03_pruebas_y_calibracion_modulo_08.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 08

---

2. Firmware Completo de Tiempo Real (`01_firmware_arduino_modulo_08.ino`)

```
/*
 * ============================================================================
 * CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 08 (PROYECTO FINAL)
 * Archivo: 01_firmware_arduino_modulo_08.ino
 * Prototipo: Celda Mecatrónica Completa (Cinta Transportadora + Inspección
 *            Dimensional + Brazo Clasificador Servo + Supervisión Raspberry Pi)
 * ============================================================================
 */

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

// --- Mapa Maestro de Pines de la Celda Clasificadora ---
const uint8_t PIN_ESTOP        = 2;  // Parada de Emergencia
const uint8_t PIN_IR_ENTRADA   = 3;  // Sensor óptico al inicio de la cinta
const uint8_t PIN_SONAR_TRIG   = 4;  // Pórtico medidor de altura (TRIG)
const uint8_t PIN_SONAR_ECHO   = 5;  // Pórtico medidor de altura (ECHO)
const uint8_t PIN_CINTA_IN1    = 7;  // Puente H L298N IN1
const uint8_t PIN_CINTA_IN2    = 8;  // Puente H L298N IN2
const uint8_t PIN_CINTA_PWM    = 9;  // Puente H L298N ENA (Velocidad cinta)
const uint8_t PIN_SERVO_BRAZO  = 10; // Brazo desviador clasificador
const uint8_t PIN_BALIZA_VERDE = 11; // Luz verde: Celda operativa
const uint8_t PIN_BALIZA_ROJA  = 12; // Luz roja: Parada / Alarma

// --- Parámetros Físicos de Inspección y Clasificación ---
const float ALTURA_PORTICO_CM  = 15.0; // Distancia del sensor a la banda vacía
const float UMBRAL_TIPO_A_CM   = 5.0;  // Pieza Alta (Tipo A) >= 5.0 cm
const int ANGULO_NEUTRO        = 90;   // Posición central
const int ANGULO_BANDEJA_A     = 45;   // Desvío hacia Rampa A (Piezas Altas)
const int ANGULO_BANDEJA_B     = 135;  // Desvío hacia Rampa B (Piezas Bajas)

enum EstadoCelda {
  CELDA_ESPERA_PIEZA    = 0,
  CELDA_TRANSPORTE      = 1,
  CELDA_MEDICION_ALTURA = 2,
  CELDA_DESVIO_SERVO    = 3,
  CELDA_DESCARGA        = 4,
  CELDA_EMERGENCIA      = 5
};

Servo brazoClasificador;
EstadoCelda estadoActual        = CELDA_ESPERA_PIEZA;
unsigned long tInicioEstadoMs   = 0;
unsigned long ultimoReporteMs   = 0;

int pwmNominalCinta             = 180;
float ultimaAlturaCm            = 0.0;
char ultimoTipo                 = '-';
unsigned long contadorTipoA     = 0;
unsigned long contadorTipoB     = 0;
bool lineaActiva                = true;

void avanzarCinta(int pwm) {
  digitalWrite(PIN_CINTA_IN1, HIGH);
  digitalWrite(PIN_CINTA_IN2, LOW);
  analogWrite(PIN_CINTA_PWM, constrain(pwm, 0, 255));
}
```

```
void frenarCinta() {
  digitalWrite(PIN_CINTA_IN1, HIGH);
  digitalWrite(PIN_CINTA_IN2, HIGH);
  analogWrite(PIN_CINTA_PWM, 255);
}

void apagarCinta() {
  digitalWrite(PIN_CINTA_IN1, LOW);
  digitalWrite(PIN_CINTA_IN2, LOW);
  analogWrite(PIN_CINTA_PWM, 0);
}

float medirAlturaPiezaCm() {
  digitalWrite(PIN_SONAR_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_SONAR_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_SONAR_TRIG, LOW);

  unsigned long duracionUs = pulseIn(PIN_SONAR_ECHO, HIGH, 20000UL);
  if (duracionUs == 0) return 0.0;
  float distSensor = (duracionUs * 0.0343) / 2.0;
  float altura = ALTURA_PORTICO_CM - distSensor;
  return max(0.0f, altura);
}

void cambiarEstado(EstadoCelda nuevo) {
  estadoActual    = nuevo;
  tInicioEstadoMs = millis();
}

const char* textoEstado(EstadoCelda e) {
  switch (e) {
    case CELDA_ESPERA_PIEZA:    return "ESPERA_PIEZA";
    case CELDA_TRANSPORTE:      return "TRANSPORTE";
    case CELDA_MEDICION_ALTURA: return "MEDICION_ALTURA";
    case CELDA_DESVIO_SERVO:    return "DESVIO_SERVO";
    case CELDA_DESCARGA:        return "DESCARGA";
    default:                    return "EMERGENCIA";
  }
}

void procesarComandosHMI() {
  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    cmd.toUpperCase();
    if (cmd == "STOP") {
      lineaActiva = false;
      apagarCinta();
    } else if (cmd == "START") {
      lineaActiva = true;
      if (estadoActual == CELDA_EMERGENCIA && digitalRead(PIN_ESTOP) == HIGH) {
        cambiarEstado(CELDA_ESPERA_PIEZA);
      }
    } else if (cmd == "RESET") {
      if (digitalRead(PIN_ESTOP) == HIGH) {
```

```
        lineaActiva = true;
        brazoClasificador.write(ANGULO_NEUTRO);
        cambiarEstado(CELDA_ESPERA_PIEZA);
      }
    } else if (cmd.startsWith("VEL:")) {
      pwmNominalCinta = constrain(cmd.substring(4).toInt(), 80, 255);
    }
  }
}

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

  pinMode(PIN_ESTOP, INPUT_PULLUP);
  pinMode(PIN_IR_ENTRADA, INPUT_PULLUP);
  pinMode(PIN_SONAR_TRIG, OUTPUT);
  pinMode(PIN_SONAR_ECHO, INPUT);

  pinMode(PIN_CINTA_IN1, OUTPUT);
  pinMode(PIN_CINTA_IN2, OUTPUT);
  pinMode(PIN_CINTA_PWM, OUTPUT);
  pinMode(PIN_BALIZA_VERDE, OUTPUT);
  pinMode(PIN_BALIZA_ROJA, OUTPUT);

  brazoClasificador.attach(PIN_SERVO_BRAZO);
  brazoClasificador.write(ANGULO_NEUTRO);
  apagarCinta();

  Serial.println(F("{"modulo":8,"evento":"CELDA_CLASIFICADORA_LISTA"}"));
}

void loop() {
  procesarComandosHMI();
  unsigned long ahora = millis();
  unsigned long tEstado = ahora - tInicioEstadoMs;

  // Vigilancia instantánea de Parada de Emergencia
  if (digitalRead(PIN_ESTOP) == LOW && estadoActual != CELDA_EMERGENCIA) {
    apagarCinta();
    brazoClasificador.write(ANGULO_NEUTRO);
    cambiarEstado(CELDA_EMERGENCIA);
  }

  switch (estadoActual) {
    case CELDA_ESPERA_PIEZA:
      apagarCinta();
      digitalWrite(PIN_BALIZA_VERDE, lineaActiva ? HIGH : LOW);
      digitalWrite(PIN_BALIZA_ROJA, LOW);
      if (lineaActiva && digitalRead(PIN_IR_ENTRADA) == LOW) {
        cambiarEstado(CELDA_TRANSPORTE);
      }
      break;

    case CELDA_TRANSPORTE:
      avanzarCinta(pwmNominalCinta);
      if (tEstado >= 1200) { // Tiempo de llegada bajo el pórtico ultrasónico
        frenarCinta();
        cambiarEstado(CELDA_MEDICION_ALTURA);
```

```
      }
      break;

    case CELDA_MEDICION_ALTURA:
      if (tEstado >= 300) { // Estabilización mecánica de 300 ms antes de medir
        ultimaAlturaCm = medirAlturaPiezaCm();
        if (ultimaAlturaCm >= UMBRAL_TIPO_A_CM) {
          ultimoTipo = 'A';
          contadorTipoA++;
        } else {
          ultimoTipo = 'B';
          contadorTipoB++;
        }
        cambiarEstado(CELDA_DESVIO_SERVO);
      }
      break;

    case CELDA_DESVIO_SERVO:
      if (ultimoTipo == 'A') {
        brazoClasificador.write(ANGULO_BANDEJA_A);
      } else {
        brazoClasificador.write(ANGULO_BANDEJA_B);
      }
      if (tEstado >= 350) {
        cambiarEstado(CELDA_DESCARGA);
      }
      break;

    case CELDA_DESCARGA:
      avanzarCinta(pwmNominalCinta);
      if (tEstado >= 1500) {
        apagarCinta();
        brazoClasificador.write(ANGULO_NEUTRO);
        cambiarEstado(CELDA_ESPERA_PIEZA);
      }
      break;

    case CELDA_EMERGENCIA:
      apagarCinta();
      digitalWrite(PIN_BALIZA_VERDE, LOW);
      digitalWrite(PIN_BALIZA_ROJA, (ahora / 250) % 2);
      break;
  }

  if (ahora - ultimoReporteMs >= 250) {
    ultimoReporteMs = ahora;
    Serial.print(F("{"modulo":8,"estado":""));
    Serial.print(textoEstado(estadoActual));
    Serial.print(F(","altura_cm":"));
    Serial.print(ultimaAlturaCm, 2);
    Serial.print(F(","tipo":""));
    Serial.print(ultimoTipo);
    Serial.print(F(","total_a":"));
    Serial.print(contadorTipoA);
    Serial.print(F(","total_b":"));
    Serial.print(contadorTipoB);
    Serial.print(F(","pwm_cinta":"));
    Serial.print(pwmNominalCinta);
```

```
    Serial.println(F("}"));
  }
}
```

---

3. Software de Supervisión y Telemetría (`02_supervision_mecatronica_modulo_08.py`)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 08 (PROYECTO FINAL)
Archivo: 02_supervision_mecatronica_modulo_08.py
Descripción: Centro de Supervisión HMI / SCADA en Raspberry Pi para la Celda
             Clasificadora y Brazo Robótico. Almacena la trazabilidad de cada
             pieza en base de datos SQLite y archivo CSV.
================================================================================
"""

import csv
import json
import os
import sqlite3
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_DB   = "produccion_mecatronica_modulo_08.db"
ARCHIVO_CSV  = "produccion_celda_modulo_08.csv"


def inicializar_almacenamiento() -> None:
    with sqlite3.connect(ARCHIVO_DB) as conn:
        conn.execute(
            """
            CREATE TABLE IF NOT EXISTS produccion (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                timestamp TEXT NOT NULL,
                estado TEXT NOT NULL,
                altura_cm REAL NOT NULL,
                tipo TEXT NOT NULL,
                total_a INTEGER NOT NULL,
                total_b INTEGER NOT NULL,
                pwm_cinta INTEGER NOT NULL
            )
            """
        )
        conn.commit()

    if not os.path.exists(ARCHIVO_CSV):
        with open(ARCHIVO_CSV, mode="w", newline="", encoding="utf-8") as f:
            csv.writer(f).writerow([
                "timestamp", "estado", "altura_cm", "tipo",
                "total_a", "total_b", "pwm_cinta"
            ])
```

```
def registrar_evento_celda(d: dict) -> None:
    ts  = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    est = d.get("estado", "ESPERA_PIEZA")
    alt = float(d.get("altura_cm", 0.0))
    tip = str(d.get("tipo", "-"))
    ta  = int(d.get("total_a", 0))
    tb  = int(d.get("total_b", 0))
    pwm = int(d.get("pwm_cinta", 180))

    print(
        f"[{ts}] Estado:{est:15s} | Altura:{alt:5.2f} cm | Tipo:{tip} | "
        f"Lote A:{ta:3d} | Lote B:{tb:3d} | Total:{ta+tb:3d}"
    )

    with sqlite3.connect(ARCHIVO_DB) as conn:
        conn.execute(
            "INSERT INTO produccion (timestamp, estado, altura_cm, tipo, total_a, total_b,
pwm_cinta) VALUES (?, ?, ?, ?, ?, ?, ?)",
            (ts, est, alt, tip, ta, tb, pwm)
        )
        conn.commit()

    with open(ARCHIVO_CSV, mode="a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow([ts, est, alt, tip, ta, tb, pwm])


def ejecutar_demo() -> None:
    print("ℹ️ Simulando operación completa de la Celda Mecatrónica Clasificadora (Módulo 08)...")
    lote_prueba = [
        {"estado": "TRANSPORTE",      "altura_cm": 0.00, "tipo": "-", "total_a": 0, "total_b": 0,
"pwm_cinta": 180},
        {"estado": "DESVIO_SERVO",    "altura_cm": 6.40, "tipo": "A", "total_a": 1, "total_b": 0,
"pwm_cinta": 180},
        {"estado": "DESVIO_SERVO",    "altura_cm": 3.15, "tipo": "B", "total_a": 1, "total_b": 1,
"pwm_cinta": 180},
        {"estado": "DESCARGA",        "altura_cm": 7.10, "tipo": "A", "total_a": 2, "total_b": 1,
"pwm_cinta": 180},
    ]
    for ev in lote_prueba:
        registrar_evento_celda(ev)
        time.sleep(0.35)


def main() -> None:
    print("=" * 80)
    print(" CENTRO DE SUPERVISIÓN SCADA MÓDULO 08 — CELDA CLASIFICADORA Y BRAZO ROBÓTICO")
    print("=" * 80)
    inicializar_almacenamiento()
    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 "total_a" in linea:
                    registrar_evento_celda(json.loads(linea))
    except Exception as e:
```

```
        print(f"⚠️ Sin puerto serie ({e}). Ejecutando simulación integradora:")
        ejecutar_demo()


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