💻 CÓDIGO Y ALGORITMOS DE CONTROL MECATRÓNICO — MÓDULO 06

Curso: Curso Completo: Introducción a la Mecatrónica

Módulo 06: Módulo 06: Automatización Secuencial, Neumática, Electroválvulas y Máquinas de Estados

Archivos en `codigo_fuente/`:

- `01_firmware_arduino_modulo_06.ino` (Control en tiempo real sobre Arduino UNO)
- `02_supervision_mecatronica_modulo_06.py` (Supervisión HMI y registro CSV/SQLite en Raspberry Pi)
- `03_pruebas_y_calibracion_modulo_06.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 06

---

2. Firmware Completo de Tiempo Real (`01_firmware_arduino_modulo_06.ino`)

```
/*
 * ============================================================================
 * CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 06
 * Archivo: 01_firmware_arduino_modulo_06.ino
 * Prototipo: Automatización Secuencial Neumática con Máquina de Estados (FSM)
 * Hardware: Arduino UNO + 2 Relés de Solenoide/Válvula (Pines 7,8)
 *           + E-Stop (Pin 2) + Start (Pin 3) + Sensores (Pines 4,5)
 * ============================================================================
 */

#include <Arduino.h>

const uint8_t PIN_ESTOP          = 2;  // Parada de Emergencia
const uint8_t PIN_START          = 3;  // Botón de Inicio de Ciclo
const uint8_t PIN_SENSOR_PIEZA   = 4;  // Pieza posicionada en el molde
const uint8_t PIN_SENSOR_PISTON  = 5;  // Cilindro neumático extendido
const uint8_t PIN_RELE_ALIMENTA  = 7;  // Relé 1: Actuador de alimentación
const uint8_t PIN_RELE_VALVULA   = 8;  // Relé 2: Solenoide Electroválvula 12V
const uint8_t PIN_BALIZA_VERDE   = 9;  // Indicador de ciclo normal
const uint8_t PIN_BALIZA_ROJA    = 10; // Indicador de Emergencia / Timeout

enum EstadoFSM {
  ESTADO_REPOSO      = 0,
  ESTADO_ALIMENTANDO = 1,
  ESTADO_PRENSANDO   = 2,
  ESTADO_RETORNANDO  = 3,
  ESTADO_EMERGENCIA  = 4
};

EstadoFSM estadoActual             = ESTADO_REPOSO;
unsigned long tInicioEstadoMs      = 0;
unsigned long ultimoReporteMs      = 0;
unsigned long lotesProducidos      = 0;
bool comandoStartRemoto            = false;

const unsigned long TIMEOUT_ATASCO_MS  = 5000; // Watchdog mecánico de 5.0 s
const unsigned long TIEMPO_PRENSADO_MS = 1500; // Mantenimiento de presión: 1.5 s
const unsigned long TIEMPO_RETORNO_MS  = 800;  // Retorno del vástago: 0.8 s

const char* nombreEstado(EstadoFSM e) {
  switch (e) {
    case ESTADO_REPOSO:      return "REPOSO";
    case ESTADO_ALIMENTANDO: return "ALIMENTANDO";
    case ESTADO_PRENSANDO:   return "PRENSANDO";
    case ESTADO_RETORNANDO:  return "RETORNANDO";
    default:                 return "FALLA_EMERGENCIA";
  }
}

void apagarActuadoresSeguridad() {
  digitalWrite(PIN_RELE_ALIMENTA, LOW);
  digitalWrite(PIN_RELE_VALVULA, LOW);
}

void cambiarEstado(EstadoFSM nuevo) {
  estadoActual    = nuevo;
  tInicioEstadoMs = millis();
```

```
}

void leerComandosSerie() {
  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    cmd.toUpperCase();
    if (cmd == "START") {
      comandoStartRemoto = true;
    } else if (cmd == "STOP") {
      apagarActuadoresSeguridad();
      cambiarEstado(ESTADO_EMERGENCIA);
    } else if (cmd == "RESET" && digitalRead(PIN_ESTOP) == HIGH) {
      apagarActuadoresSeguridad();
      cambiarEstado(ESTADO_REPOSO);
    }
  }
}

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

  pinMode(PIN_ESTOP, INPUT_PULLUP);
  pinMode(PIN_START, INPUT_PULLUP);
  pinMode(PIN_SENSOR_PIEZA, INPUT_PULLUP);
  pinMode(PIN_SENSOR_PISTON, INPUT_PULLUP);

  pinMode(PIN_RELE_ALIMENTA, OUTPUT);
  pinMode(PIN_RELE_VALVULA, OUTPUT);
  pinMode(PIN_BALIZA_VERDE, OUTPUT);
  pinMode(PIN_BALIZA_ROJA, OUTPUT);

  apagarActuadoresSeguridad();
  cambiarEstado(ESTADO_REPOSO);
}

void loop() {
  leerComandosSerie();
  unsigned long ahora = millis();
  unsigned long tEnEstado = ahora - tInicioEstadoMs;

  // 1. VERIFICACIÓN PRIORITARIA DE PARADA DE EMERGENCIA (Cero bloqueos)
  if (digitalRead(PIN_ESTOP) == LOW && estadoActual != ESTADO_EMERGENCIA) {
    apagarActuadoresSeguridad();
    cambiarEstado(ESTADO_EMERGENCIA);
  }

  // 2. MÁQUINA DE ESTADOS FINITOS (FSM)
  switch (estadoActual) {
    case ESTADO_REPOSO:
      apagarActuadoresSeguridad();
      digitalWrite(PIN_BALIZA_VERDE, HIGH);
      digitalWrite(PIN_BALIZA_ROJA, LOW);
      if (digitalRead(PIN_START) == LOW || comandoStartRemoto) {
        comandoStartRemoto = false;
        cambiarEstado(ESTADO_ALIMENTANDO);
      }
      break;
```

```
    case ESTADO_ALIMENTANDO:
      digitalWrite(PIN_RELE_ALIMENTA, HIGH);
      digitalWrite(PIN_RELE_VALVULA, LOW);
      if (digitalRead(PIN_SENSOR_PIEZA) == LOW) {
        digitalWrite(PIN_RELE_ALIMENTA, LOW);
        cambiarEstado(ESTADO_PRENSANDO);
      } else if (tEnEstado > TIMEOUT_ATASCO_MS) {
        // Watchdog mecánico: la pieza no llegó en 5 segundos
        apagarActuadoresSeguridad();
        cambiarEstado(ESTADO_EMERGENCIA);
      }
      break;

    case ESTADO_PRENSANDO:
      digitalWrite(PIN_RELE_ALIMENTA, LOW);
      digitalWrite(PIN_RELE_VALVULA, HIGH); // Energiza solenoide electroválvula
      if (digitalRead(PIN_SENSOR_PISTON) == LOW && tEnEstado >= TIEMPO_PRENSADO_MS) {
        digitalWrite(PIN_RELE_VALVULA, LOW);
        cambiarEstado(ESTADO_RETORNANDO);
      } else if (tEnEstado > TIMEOUT_ATASCO_MS) {
        apagarActuadoresSeguridad();
        cambiarEstado(ESTADO_EMERGENCIA);
      }
      break;

    case ESTADO_RETORNANDO:
      apagarActuadoresSeguridad();
      if (tEnEstado >= TIEMPO_RETORNO_MS) {
        lotesProducidos++;
        cambiarEstado(ESTADO_REPOSO);
      }
      break;

    case ESTADO_EMERGENCIA:
      apagarActuadoresSeguridad();
      digitalWrite(PIN_BALIZA_VERDE, LOW);
      digitalWrite(PIN_BALIZA_ROJA, (ahora / 250) % 2); // Parpadeo de alarma
      break;
  }

  // 3. REPORTE DE TELEMETRÍA INDUSTRIAL CADA 200 MS
  if (ahora - ultimoReporteMs >= 200) {
    ultimoReporteMs = ahora;
    Serial.print(F("{"modulo":6,"estado":""));
    Serial.print(nombreEstado(estadoActual));
    Serial.print(F("","t_estado_ms":"));
    Serial.print(tEnEstado);
    Serial.print(F(","rele_alim":"));
    Serial.print(digitalRead(PIN_RELE_ALIMENTA));
    Serial.print(F(","valvula_neum":"));
    Serial.print(digitalRead(PIN_RELE_VALVULA));
    Serial.print(F(","lotes":"));
    Serial.print(lotesProducidos);
    Serial.println(F("}"));
  }
}
```

3. Software de Supervisión y Telemetría (`02_supervision_mecatronica_modulo_06.py`)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 06
Archivo: 02_supervision_mecatronica_modulo_06.py
Descripción: Supervisor de Máquina de Estados Finitos (FSM) y estación
             neumática en Raspberry Pi. Audita transiciones de estado,
             tiempos de ciclo, accionamiento de electroválvulas y paradas.
================================================================================
"""

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  = "secuencia_fsm_modulo_06.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", "estado", "t_estado_ms", "rele_alim",
"valvula_neum", "lotes"])


def mostrar_estado(d: dict) -> None:
    ts  = datetime.now().strftime("%H:%M:%S.%f")[:-3]
    est = d.get("estado", "REPOSO")
    tms = d.get("t_estado_ms", 0)
    r1  = d.get("rele_alim", 0)
    v2  = d.get("valvula_neum", 0)
    lot = d.get("lotes", 0)

    print(
        f"[{ts}] Estado FSM: {est:16s} ({tms:4d} ms) | "
        f"Alimentador:{r1} | Electroválvula 12V:{v2} | Lotes:{lot:3d}"
    )
    with open(ARCHIVO_CSV, mode="a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow([ts, est, tms, r1, v2, lot])


def ejecutar_demo() -> None:
    print("ℹ️ Simulando ciclo industrial completo de Máquina de Estados (FSM) + Evento E-Stop...")
    pasos = [
        {"estado": "REPOSO",           "t_estado_ms": 120,  "rele_alim": 0, "valvula_neum": 0,
"lotes": 0},
```

```
        {"estado": "ALIMENTANDO",      "t_estado_ms": 840,  "rele_alim": 1, "valvula_neum": 0,
"lotes": 0},
        {"estado": "PRENSANDO",        "t_estado_ms": 1500, "rele_alim": 0, "valvula_neum": 1,
"lotes": 0},
        {"estado": "RETORNANDO",       "t_estado_ms": 800,  "rele_alim": 0, "valvula_neum": 0,
"lotes": 1},
        {"estado": "FALLA_EMERGENCIA", "t_estado_ms": 50,   "rele_alim": 0, "valvula_neum": 0,
"lotes": 1},
    ]
    for p in pasos:
        mostrar_estado(p)
        time.sleep(0.35)


def main() -> None:
    print("=" * 78)
    print(" SUPERVISOR SECUENCIAL Y NEUMÁTICO MÓDULO 06 — MÁQUINA DE ESTADOS (FSM)")
    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 "valvula_neum" in linea:
                    mostrar_estado(json.loads(linea))
    except Exception as e:
        print(f"⚠️ Sin puerto serie ({e}). Ejecutando simulación secuencial:")
        ejecutar_demo()


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