💻 CÓDIGO Y ALGORITMOS DE CONTROL MECATRÓNICO — MÓDULO 04

Curso: Curso Completo: Introducción a la Mecatrónica

Módulo 04: Módulo 04: Motores DC y Electrónica de Potencia con Puente H (L298N / TB6612FNG)

Archivos en `codigo_fuente/`:

- `01_firmware_arduino_modulo_04.ino` (Control en tiempo real sobre Arduino UNO)
- `02_supervision_mecatronica_modulo_04.py` (Supervisión HMI y registro CSV/SQLite en Raspberry Pi)
- `03_pruebas_y_calibracion_modulo_04.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 04

---

2. Firmware Completo de Tiempo Real (`01_firmware_arduino_modulo_04.ino`)

```
/*
 * ============================================================================
 * CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 04
 * Archivo: 01_firmware_arduino_modulo_04.ino
 * Prototipo: Control Bidireccional de Motor DC con Puente H L298N y Rampa PWM
 * Hardware: Arduino UNO + Puente H L298N (ENA=9, IN1=7, IN2=8)
 *           + Finales de Carrera Izq/Der (Pines 2,3) + Potenciómetro (A0)
 * ============================================================================
 */

#include <Arduino.h>

const uint8_t PIN_TOPE_IZQ = 2;  // Final de carrera extremo izquierdo
const uint8_t PIN_TOPE_DER = 3;  // Final de carrera extremo derecho
const uint8_t PIN_IN1      = 7;  // Dirección 1 del Puente H L298N
const uint8_t PIN_IN2      = 8;  // Dirección 2 del Puente H L298N
const uint8_t PIN_ENA_PWM  = 9;  // Velocidad PWM (0..255)
const uint8_t PIN_POT_VEL  = A0; // Perilla de ajuste de velocidad

enum SentidoGiro { GIRO_PARADO = 0, GIRO_ADELANTE = 1, GIRO_ATRAS = 2, GIRO_FRENO = 3 };

SentidoGiro sentidoActual = GIRO_ADELANTE;
int pwmObjetivo           = 0;
int pwmSuavizado          = 0;
int pwmRemoto             = -1;

unsigned long ultimaRampaMs = 0;
unsigned long ultimoEnvioMs = 0;

void aplicarPuenteH(SentidoGiro s, int pwm) {
  pwm = constrain(pwm, 0, 255);
  switch (s) {
    case GIRO_ADELANTE:
      digitalWrite(PIN_IN1, HIGH);
      digitalWrite(PIN_IN2, LOW);
      analogWrite(PIN_ENA_PWM, pwm);
      break;
    case GIRO_ATRAS:
      digitalWrite(PIN_IN1, LOW);
      digitalWrite(PIN_IN2, HIGH);
      analogWrite(PIN_ENA_PWM, pwm);
      break;
    case GIRO_FRENO:
      // Freno dinámico electromagnético (cortocircuita devanado por el puente)
      digitalWrite(PIN_IN1, HIGH);
      digitalWrite(PIN_IN2, HIGH);
      analogWrite(PIN_ENA_PWM, 255);
      break;
    default:
      // Parada por rueda libre
      digitalWrite(PIN_IN1, LOW);
      digitalWrite(PIN_IN2, LOW);
      analogWrite(PIN_ENA_PWM, 0);
      break;
  }
}
```

```
void procesarComandosSerie() {
  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    cmd.toUpperCase();
    if (cmd == "FWD") {
      sentidoActual = GIRO_ADELANTE;
      pwmSuavizado  = 0;
    } else if (cmd == "REV") {
      sentidoActual = GIRO_ATRAS;
      pwmSuavizado  = 0;
    } else if (cmd == "STOP") {
      sentidoActual = GIRO_FRENO;
      pwmSuavizado  = 0;
    } else if (cmd == "AUTO") {
      pwmRemoto = -1;
    } else if (cmd.startsWith("PWM:")) {
      pwmRemoto = constrain(cmd.substring(4).toInt(), 0, 255);
    }
  }
}

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

  pinMode(PIN_TOPE_IZQ, INPUT_PULLUP);
  pinMode(PIN_TOPE_DER, INPUT_PULLUP);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);
  pinMode(PIN_ENA_PWM, OUTPUT);

  aplicarPuenteH(GIRO_PARADO, 0);
  Serial.println(F("{"modulo":4,"evento":"PUENTE_H_L298N_LISTO"}"));
}

void loop() {
  procesarComandosSerie();

  bool topeIzq = (digitalRead(PIN_TOPE_IZQ) == LOW);
  bool topeDer = (digitalRead(PIN_TOPE_DER) == LOW);

  // Protección e inversión automática al alcanzar los extremos mecánicos
  if (topeDer && sentidoActual == GIRO_ADELANTE) {
    aplicarPuenteH(GIRO_FRENO, 255);
    delay(250);
    sentidoActual = GIRO_ATRAS;
    pwmSuavizado  = 0;
  } else if (topeIzq && sentidoActual == GIRO_ATRAS) {
    aplicarPuenteH(GIRO_FRENO, 255);
    delay(250);
    sentidoActual = GIRO_ADELANTE;
    pwmSuavizado  = 0;
  }

  // Selección de velocidad objetivo
  if (pwmRemoto >= 0) {
    pwmObjetivo = pwmRemoto;
  } else {
```

```
    pwmObjetivo = map(analogRead(PIN_POT_VEL), 0, 1023, 0, 255);
  }

  // Rampa progresiva de aceleración / desaceleración (Soft-Start cada 20 ms)
  unsigned long ahora = millis();
  if (ahora - ultimaRampaMs >= 20) {
    ultimaRampaMs = ahora;
    if (pwmSuavizado < pwmObjetivo) {
      pwmSuavizado = min(pwmSuavizado + 5, pwmObjetivo);
    } else if (pwmSuavizado > pwmObjetivo) {
      pwmSuavizado = max(pwmSuavizado - 5, pwmObjetivo);
    }
    aplicarPuenteH(sentidoActual, pwmSuavizado);
  }

  // Telemetría de potencia cada 200 ms
  if (ahora - ultimoEnvioMs >= 200) {
    ultimoEnvioMs = ahora;
    const char* txtSentido = (sentidoActual == GIRO_ADELANTE) ? "FWD" :
                             (sentidoActual == GIRO_ATRAS)    ? "REV" :
                             (sentidoActual == GIRO_FRENO)    ? "BRAKE" : "COAST";
    int pctPotencia = (pwmSuavizado * 100) / 255;

    Serial.print(F("{"modulo":4,"sentido":""));
    Serial.print(txtSentido);
    Serial.print(F(","pwm":"));
    Serial.print(pwmSuavizado);
    Serial.print(F(","potencia_pct":"));
    Serial.print(pctPotencia);
    Serial.print(F(","tope_izq":"));
    Serial.print(topeIzq ? 1 : 0);
    Serial.print(F(","tope_der":"));
    Serial.print(topeDer ? 1 : 0);
    Serial.println(F("}"));
  }
}
```

---

3. Software de Supervisión y Telemetría (`02_supervision_mecatronica_modulo_04.py`)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 04
Archivo: 02_supervision_mecatronica_modulo_04.py
Descripción: Supervisor de etapa de potencia Puente H (L298N / TB6612FNG).
             Registra sentido de giro, rampa PWM, porcentaje de potencia y
             eventos de inversión en finales de carrera.
================================================================================
"""

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  = "potencia_puente_h_modulo_04.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", "sentido", "pwm", "potencia_pct", "tope_izq",
"tope_der"])


def registrar_estado(d: dict) -> None:
    ts  = datetime.now().strftime("%H:%M:%S.%f")[:-3]
    sen = d.get("sentido", "FWD")
    pwm = d.get("pwm", 0)
    pct = d.get("potencia_pct", 0)
    ti  = d.get("tope_izq", 0)
    td  = d.get("tope_der", 0)

    print(
        f"[{ts}] Sentido: {sen:5s} | PWM: {pwm:3d}/255 ({pct:3d}%) | "
        f"Tope Izq: {ti} | Tope Der: {td}"
    )
    with open(ARCHIVO_CSV, mode="a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow([ts, sen, pwm, pct, ti, td])


def ejecutar_demo() -> None:
    print("ℹ️ Simulando rampa de arranque, avance, detección de tope e inversión con Puente H...")
    ciclo = [
        {"sentido": "FWD",   "pwm": 90,  "potencia_pct": 35, "tope_izq": 0, "tope_der": 0},
        {"sentido": "FWD",   "pwm": 190, "potencia_pct": 74, "tope_izq": 0, "tope_der": 0},
```

```
        {"sentido": "BRAKE", "pwm": 255, "potencia_pct": 100,"tope_izq": 0, "tope_der": 1},
        {"sentido": "REV",   "pwm": 160, "potencia_pct": 62, "tope_izq": 0, "tope_der": 0},
    ]
    for c in ciclo:
        registrar_estado(c)
        time.sleep(0.35)


def main() -> None:
    print("=" * 78)
    print(" SUPERVISOR DE POTENCIA MÓDULO 04 — MOTORES DC Y PUENTE H L298N")
    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 "potencia_pct" in linea:
                    registrar_estado(json.loads(linea))
    except Exception as e:
        print(f"⚠️ Sin conexión serie ({e}). Ejecutando simulación de potencia:")
        ejecutar_demo()


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