💻 CÓDIGO Y ALGORITMOS DE CONTROL MECATRÓNICO — MÓDULO 02

Curso: Curso Completo: Introducción a la Mecatrónica

Módulo 02: Módulo 02: Mecánica Aplicada: Engranajes, Poleas, Husillos, Torque y Velocidad

Archivos en `codigo_fuente/`:

- `01_firmware_arduino_modulo_02.ino` (Control en tiempo real sobre Arduino UNO)
- `02_supervision_mecatronica_modulo_02.py` (Supervisión HMI y registro CSV/SQLite en Raspberry Pi)
- `03_pruebas_y_calibracion_modulo_02.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 02

---

2. Firmware Completo de Tiempo Real (`01_firmware_arduino_modulo_02.ino`)

```
/*
 * ============================================================================
 * CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 02
 * Archivo: 01_firmware_arduino_modulo_02.ino
 * Prototipo: Banco de Telemetría Mecánica (Engranajes, Poleas, Torque y RPM)
 * Hardware: Arduino UNO + Sensor Óptico Ranurado LM393 (Pin 2 INT0)
 *           + Motor DC con Caja Reductora 48:1 (PWM Pin 9) + Potenciómetro (A0)
 * ============================================================================
 */

#include <Arduino.h>

const uint8_t PIN_ENCODER_OPTICO = 2;   // Interrupción 0 en Arduino UNO
const uint8_t PIN_LED_PULSO      = 8;   // Indicador visual de giro
const uint8_t PIN_PWM_MOTOR      = 9;   // Salida PWM hacia etapa de potencia
const uint8_t PIN_POT_VELOCIDAD  = A0;  // Mando de régimen de trabajo

// --- Parámetros Mecánicos del Tren de Transmisión ---
const float RANURAS_DISCO        = 20.0; // Disco encoder de 20 ranuras
const float RELACION_REDUCTORA   = 48.0; // Caja reductora 48:1 (ej. motor TT)
const float PASO_HUSILLO_MM      = 2.0;  // Avance lineal por vuelta (husillo M8 / GT2)
const float TORQUE_MOTOR_GCM     = 15.0; // Par nominal del rotor sin reducción (g·cm)
const float EFICIENCIA_MECANICA  = 0.82; // Rendimiento mecánico del tren de engranajes

volatile unsigned long contadorPulsos = 0;
unsigned long ultimoCalculoMs         = 0;
const unsigned long VENTANA_MS        = 500; // Medición cada 500 ms exactos

int pwmComandoRemoto = -1;

// Rutina de Servicio de Interrupción (ISR) ultrarrápida
void contarRanuraISR() {
  contadorPulsos++;
}

void leerComandosSerie() {
  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    cmd.toUpperCase();
    if (cmd == "AUTO") {
      pwmComandoRemoto = -1;
    } else if (cmd.startsWith("PWM:")) {
      pwmComandoRemoto = constrain(cmd.substring(4).toInt(), 0, 255);
    }
  }
}

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

  pinMode(PIN_ENCODER_OPTICO, INPUT_PULLUP);
  pinMode(PIN_LED_PULSO, OUTPUT);
  pinMode(PIN_PWM_MOTOR, OUTPUT);

  attachInterrupt(digitalPinToInterrupt(PIN_ENCODER_OPTICO), contarRanuraISR, RISING);
```

```
  Serial.println(F("{"modulo":2,"evento":"BANCO_MECANICO_INICIADO","relacion":48.0}"));
}

void loop() {
  leerComandosSerie();

  int pwmActual = 0;
  if (pwmComandoRemoto >= 0) {
    pwmActual = pwmComandoRemoto;
  } else {
    pwmActual = map(analogRead(PIN_POT_VELOCIDAD), 0, 1023, 0, 255);
  }
  analogWrite(PIN_PWM_MOTOR, pwmActual);

  unsigned long ahora = millis();
  if (ahora - ultimoCalculoMs >= VENTANA_MS) {
    unsigned long dtMs = ahora - ultimoCalculoMs;
    ultimoCalculoMs = ahora;

    // Lectura atómica del contador de interrupciones
    noInterrupts();
    unsigned long pulsosVentana = contadorPulsos;
    contadorPulsos = 0;
    interrupts();

    // Cálculos cinemáticos y dinámicos
    float vueltasSalida = (float)pulsosVentana / RANURAS_DISCO;
    float rpmSalida     = (vueltasSalida * 60000.0) / (float)dtMs;
    float rpmMotor      = rpmSalida * RELACION_REDUCTORA;
    float velLinealMmS  = (rpmSalida / 60.0) * PASO_HUSILLO_MM;
    float torqueSalGcm  = TORQUE_MOTOR_GCM * RELACION_REDUCTORA * EFICIENCIA_MECANICA * (pwmActual /
255.0);

    digitalWrite(PIN_LED_PULSO, (pulsosVentana > 0) ? HIGH : LOW);

    Serial.print(F("{"modulo":2,"pwm":"));
    Serial.print(pwmActual);
    Serial.print(F(","pulsos_500ms":"));
    Serial.print(pulsosVentana);
    Serial.print(F(","rpm_salida":"));
    Serial.print(rpmSalida, 1);
    Serial.print(F(","rpm_motor":"));
    Serial.print(rpmMotor, 0);
    Serial.print(F(","vel_lineal_mm_s":"));
    Serial.print(velLinealMmS, 2);
    Serial.print(F(","torque_est_gcm":"));
    Serial.print(torqueSalGcm, 1);
    Serial.println(F("}"));
  }
}
```

---

3. Software de Supervisión y Telemetría (`02_supervision_mecatronica_modulo_02.py`)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO COMPLETO: INTRODUCCIÓN A LA MECATRÓNICA — MÓDULO 02
Archivo: 02_supervision_mecatronica_modulo_02.py
Descripción: Analizador cinemático en Raspberry Pi. Calcula relaciones de
             transmisión, velocidad tangencial/lineal y multiplicación de
             torque a partir de la telemetría del tacómetro óptico.
================================================================================
"""

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  = "telemetria_mecanica_modulo_02.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", "pwm", "pulsos_500ms", "rpm_salida",
                "rpm_motor", "vel_lineal_mm_s", "torque_est_gcm"
            ])


def mostrar_y_guardar(d: dict) -> None:
    ts  = datetime.now().strftime("%H:%M:%S.%f")[:-3]
    pwm = d.get("pwm", 0)
    pul = d.get("pulsos_500ms", 0)
     r_s = d.get("rpm_salida", 0.0)
    r_m = d.get("rpm_motor", 0.0)
    vel = d.get("vel_lineal_mm_s", 0.0)
    trq = d.get("torque_est_gcm", 0.0)

    print(
        f"[{ts}] PWM:{pwm:3d} | Eje Salida:{r_s:6.1f} RPM | "
        f"Rotor Int:{r_m:6.0f} RPM | Avance:{vel:5.2f} mm/s | Torque:{trq:6.1f} g·cm"
    )
    with open(ARCHIVO_CSV, mode="a", newline="", encoding="utf-8") as f:
        csv.writer(f).writerow([ts, pwm, pul, r_s, r_m, vel, trq])


def ejecutar_demo() -> None:
    print("ℹ️ Simulando banco de transmisión mecánica (Reducción i = 48:1, Disco = 20 ranuras)...")
```

```
    ensayos = [
        {"pwm": 80,  "pulsos_500ms": 6,  "rpm_salida": 36.0,  "rpm_motor": 1728, "vel_lineal_mm_s":
1.20, "torque_est_gcm": 185.2},
        {"pwm": 128, "pulsos_500ms": 15, "rpm_salida": 90.0,  "rpm_motor": 4320, "vel_lineal_mm_s":
3.00, "torque_est_gcm": 296.4},
        {"pwm": 200, "pulsos_500ms": 25, "rpm_salida": 150.0, "rpm_motor": 7200, "vel_lineal_mm_s":
5.00, "torque_est_gcm": 463.1},
        {"pwm": 255, "pulsos_500ms": 32, "rpm_salida": 192.0, "rpm_motor": 9216, "vel_lineal_mm_s":
6.40, "torque_est_gcm": 590.4},
    ]
    for e in ensayos:
        mostrar_y_guardar(e)
        time.sleep(0.35)


def main() -> None:
    print("=" * 78)
    print(" ANALIZADOR DE TRANSMISIÓN MECÁNICA MÓDULO 02 — ENGRANAJES, TORQUE Y RPM")
    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 "rpm_salida" in linea:
                    mostrar_y_guardar(json.loads(linea))
    except Exception as e:
        print(f"⚠️ Sin conexión física ({e}). Ejecutando simulación de banco:")
        ejecutar_demo()


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