UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

Control de fuerza

Carlos Mira López
Nicolás Miró Mira
Vittorio Alessandro Esposito Ceballos

Modelado y Control de Robots

4.º curso - Grado en Ingeniería ...

Diciembre de 2025

Índice general

1 Introducción

En este trabajo se implementa un sistema de control de fuerza para el robot manipulador de 3 GDL desarrollado en el Trabajo 1. El objetivo principal es permitir que el robot ejerza una fuerza constante de 0.5 kg sobre una superficie plana, independientemente de pequeñas variaciones en la posición o irregularidades. Para ello, se integra un sensor de presión RP-C10-LT en el efector final y se diseña un lazo de control encargado de ajustar dinámicamente el movimiento de los servomotores en función de la fuerza medida. Finalmente, se validará experimentalmente el desempeño del sistema.

2 Integración del sensor RP-C10-LT

El sensor RP-C10-LT se acopla mecánicamente al efector final del robot, de forma que pueda medir la fuerza normal ejercida sobre la superficie de contacto. Su señal analógica se conecta directamente a una de las entradas analógicas del Arduino UNO R3, utilizando una alimentación de 5V y referencia común con el resto del sistema.

En el firmware del Arduino se implementa un proceso de lectura continua del sensor, aplicando un pequeño filtrado para reducir ruido (media móvil). La lectura resultante se transforma a unidades de fuerza mediante la curva de calibración proporcionada por el fabricante o mediante calibración experimental realizada previamente.

3 Algoritmo de control

Listing 1: *Parámetros de impedancia y configuración global.*

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>
#include <math.h>

%% ---------- FSR (Sensor de Fuerza) ----------
const int pinFSR = A0;

%% ---------- PCA9685 ----------
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();  %% I2C 0x40

%% Canales en PCA9685
#define SERVO_BASE  0
#define SERVO_BRAZO 1
#define SERVO_CODO  2

%% Ángulos actuales / referencia
int   baseAngle  = 0;
int   brazoAngle = 90;
int   codoAngle  = 90;
float brazoRef   = 90.0f;

%% HOME del brazo (Posición de reposo)
int BRAZO_HOME = 90;

%% ---------- Escalado a kg ----------
const int   ADC_MIN = 100;
const int   ADC_MAX = 960;
const float KG_MIN  = 0.2f;
```

```
const float KG_MAX  = 3.0f;

%% ---------- Filtros ----------
const uint8_t N_AVG = 6;
int     bufAvg[N_AVG] = {0};
uint8_t idxAvg = 0;
bool    bufLleno = false;

const float ALPHA_KG = 0.15f; %% Coeficiente filtro (bajo = suave)
float kg_filt = 0.0f;

%% ---------- Control de Impedancia ----------
const float KG_TARGET   = 0.5f;
const float KG_DEADBAND = 0.10f; %% Histéresis de activación

%% Parámetros físicos de la impedancia
const float K_STIFF_RET  = 2.0f;   %% Rigidez [deg/kg]
const float B_DAMP_RET   = 0.5f;   %% Amortiguamiento [deg/(kg/s)]

%% Vuelta a HOME
const float KP_HOME           = 0.5f;
const float STEP_HOME_MAX     = 0.7f;
const float HOME_DEADBAND_DEG = 1.0f;

%% Límites y timing
const int   BRAZO_MIN        = 10;
const int   BRAZO_MAX        = 170;
const float STEP_LIMIT_DEG   = 1.0f;
const float MAX_RETRACT_DEG  = 200.0f;
const uint16_t UPDATE_MS     = 40;
const bool  INVERT_BRAZO_DIR = true;

unsigned long tPrevCtrl     = 0;
unsigned long tLastSampleMs = 0;
float         kg_prev       = 0.0f;

%% Máquina de estados del brazo
enum BrazoMode { MODE_HOME = 0, MODE_COMPLIANT = 1 };
BrazoMode modoBrazo = MODE_HOME;

unsigned long lastHighForceMs = 0;
```

Listing 2: *Funciones auxiliares: Conversión y Telemetría.*

```
%% ---------------- Utilidades ----------------
int angleToPulse(int ang) {
    int pulsoMin = 150;  %% 0 grados
    int pulsoMax = 600;  %% 180 grados
    return map(ang, 0, 180, pulsoMin, pulsoMax);
}

float fmap(float x, float in_min, float in_max, float out_min, float out_max) {
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

float rawToKg(int raw) {
    if (raw <= ADC_MIN) return KG_MIN;
    if (raw >= ADC_MAX) return KG_MAX;

    float kg = fmap((float)raw, (float)ADC_MIN, (float)ADC_MAX, KG_MIN, KG_MAX);

    if (kg < KG_MIN) kg = KG_MIN;
    if (kg > KG_MAX) kg = KG_MAX;
```

```
    return kg;
}

void moverServo(int canal, int angulo) {
    int pulso = angleToPulse(angulo);
    pwm.setPWM(canal, 0, pulso);
}

void sendTelemetry(unsigned long timestampMs, int raw, int rawFiltrado,
                   float kgEscalado, float kgFiltrado, float kgReferencia,
                   int brazoPosDeg, int modo) {
    Serial.print("DATA,");
    Serial.print(timestampMs);
    Serial.print(',');
    Serial.print(raw);
    Serial.print(',');
    Serial.print(rawFiltrado);
    Serial.print(',');
    Serial.print(kgEscalado, 4);
    Serial.print(',');
    Serial.print(kgFiltrado, 4);
    Serial.print(',');
    Serial.print(kgReferencia, 4);
    Serial.print(',');
    Serial.print(brazoPosDeg);
    Serial.print(',');
    Serial.println(modo);
}
```

Listing 3: *Control Parte 1: Filtrado y Máquina de Estados.*

```
%% ----------- Bucle de control -----------
bool controlBrazoImpedancia(float kg_meas) {
    unsigned long now = millis();
    if ((now - tPrevCtrl) < UPDATE_MS) return false;

    float dt = (now - tPrevCtrl) / 1000.0f;
    if (dt <= 0.0f) dt = 1e-3f;
    tPrevCtrl = now;

    %% Filtrado de fuerza (Paso Bajo)
    kg_filt = ALPHA_KG * kg_meas + (1.0f - ALPHA_KG) * kg_filt;

    %% Derivada de fuerza (para el término amortiguador B)
    float dkg = (kg_filt - kg_prev) / dt;
    const float ALPHA_D = 0.3f;
    static float dkg_filt = 0.0f;
    dkg_filt = ALPHA_D * dkg + (1.0f - ALPHA_D) * dkg_filt;
    kg_prev = kg_filt;

    %% Registrar si hay fuerza alta (para mantener COMPLIANT un rato)
    if (kg_filt > (KG_TARGET + 0.5f * KG_DEADBAND)) {
        lastHighForceMs = now;
    }

    %% -------- Máquina de estados --------
    if (modoBrazo == MODE_HOME) {
        %% Estamos en HOME / volviendo a HOME
        if (kg_filt > (KG_TARGET + KG_DEADBAND)) {
            %% Entramos en modo "cede" sólo si fuerza supera umbral
            modoBrazo = MODE_COMPLIANT;
        }
    } else { %% MODE_COMPLIANT
```

```
        %% Salimos cuando fuerza baja bastante y se mantiene baja un rato
        if ((kg_filt < (KG_TARGET - KG_DEADBAND)) && (now - lastHighForceMs > 250)) {
            modoBrazo = MODE_HOME;
        }
    }
```

Listing 4: *Control Parte 2: Ecuación de Impedancia y Movimiento.*

```
    float dtheta = 0.0f;

    if (modoBrazo == MODE_COMPLIANT) {
        %% -------- Modo COMPLIANT: cede ante la fuerza --------
        float eF = kg_filt - KG_TARGET;
        if (eF < 0.0f) eF = 0.0f;

        float dkg_pos = (dkg_filt > 0.0f) ? dkg_filt : 0.0f;

        %% Ecuación de Impedancia: Rigidez + Amortiguamiento
        float dtheta_mag = K_STIFF_RET * eF + B_DAMP_RET * dkg_pos;
        if (dtheta_mag > STEP_LIMIT_DEG) dtheta_mag = STEP_LIMIT_DEG;

        float retractSign = INVERT_BRAZO_DIR ? +1.0f : -1.0f;
        dtheta = retractSign * dtheta_mag;

    } else {
        %% -------- Modo HOME: volver despacio a la posición inicial --------
        float posErr = (float)BRAZO_HOME - brazoRef;

        if (fabs(posErr) < HOME_DEADBAND_DEG) {
            dtheta = 0.0f;
        } else {
            float step = KP_HOME * posErr;

            if (step >  STEP_HOME_MAX) step =  STEP_HOME_MAX;
            if (step < -STEP_HOME_MAX) step = -STEP_HOME_MAX;

            %% No avanzar por delante de HOME
            if ((posErr <= 0.0f) && (step > 0.0f)) step = 0.0f;

            dtheta = step;
        }
    }

    %% Clamp de paso por ciclo (seguridad)
    if (dtheta >  STEP_LIMIT_DEG) dtheta =  STEP_LIMIT_DEG;
    if (dtheta < -STEP_LIMIT_DEG) dtheta = -STEP_LIMIT_DEG;

    %% Aplica sobre referencia flotante
    brazoRef += dtheta;
    float nuevo = brazoRef;

    %% Limita rango respecto a HOME para evitar viajes grandes
    float homeMin = (float)BRAZO_HOME - MAX_RETRACT_DEG;
    float homeMax = (float)BRAZO_HOME + MAX_RETRACT_DEG;
    if (nuevo < homeMin) nuevo = homeMin;
    if (nuevo > homeMax) nuevo = homeMax;

    %% Límites mecánicos absolutos
    if (nuevo < BRAZO_MIN) nuevo = BRAZO_MIN;
    if (nuevo > BRAZO_MAX) nuevo = BRAZO_MAX;

    %% Actualiza Servo
    brazoRef = nuevo;
```

```
    int nuevoInt = (int)roundf(nuevo);

    if (nuevoInt != brazoAngle) {
        brazoAngle = nuevoInt;
        moverServo(SERVO_BRAZO, brazoAngle);
    }

    tLastSampleMs = now;
    return true;
}
```

Listing 5: *Setup y Bucle Principal con media móvil.*

```
%% ---------------- Setup / Loop ----------------
void setup() {
    Serial.begin(115200);
    Serial.println("# Force control telemetry v2");

    pwm.begin();
    pwm.setPWMFreq(60);
    delay(10);

    moverServo(SERVO_BASE, baseAngle);
    moverServo(SERVO_BRAZO, brazoAngle);
    moverServo(SERVO_CODO, codoAngle);

    %% Establecer posición actual como HOME
    BRAZO_HOME = brazoAngle;
    brazoRef   = (float)brazoAngle;

    %% Pre-llenado del buffer de media móvil
    int raw = analogRead(pinFSR);
    for (uint8_t i = 0; i < N_AVG; i++) bufAvg[i] = raw;
    bufLleno = true;

    kg_filt = rawToKg(raw);
    kg_prev = kg_filt;
    tPrevCtrl = millis();
    lastHighForceMs = millis();
}

void loop() {
    %% Media móvil del ADC para reducir ruido
    int raw = analogRead(pinFSR);
    bufAvg[idxAvg++] = raw;

    if (idxAvg >= N_AVG) {
        idxAvg = 0;
        bufLleno = true;
    }

    long suma = 0;
    uint8_t n = bufLleno ? N_AVG : idxAvg;
    for (uint8_t i = 0; i < n; i++) suma += bufAvg[i];
    int rawFiltrado = (int)(suma / n);

    float kg = rawToKg(rawFiltrado);

    %% Ejecución del control
    if (controlBrazoImpedancia(kg)) {
        sendTelemetry(tLastSampleMs,
                      raw,
                      rawFiltrado,
```

```
                        kg,
                        kg_filt,
                        KG_TARGET,
                        brazoAngle,
                        (int)modoBrazo);
    }

    %% Para monitorizar rápido la fuerza en kg
    Serial.print("SETA,");
    Serial.print(kg);
    Serial.print("\n");

    delay(3);
}
```

4 Resultados experimentales

5 limitaciones y posibles mejoras

Aunque el sistema cumple con los requisitos, presenta algunas limitaciones. El uso de servomotores 9G introduce cierta holgura mecánica y ruido en la medida, lo que afecta a la precisión del control. El control proporcional puede generar oscilaciones si la ganancia no está ajustada correctamente. Además, la lectura analógica del sensor presenta variabilidad que podría reducirse con mejor filtrado.

Como mejoras futuras, se podría implementar un control PID, añadir un sensor de referencia independiente, mejorar la rigidez de la estructura mecánica o sustituir los servos por actuadores con retroalimentación interna.