



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

CAMPUS D'ALCOI

# Modelado y control cinemático

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

<b>1</b>	<b>Introducción</b>	<b>1</b>
<b>2</b>	<b>Descripción del diseño mecánico y electrónico</b>	<b>1</b>
<b>3</b>	<b>Modelo matemático</b>	<b>2</b>
3.1	Cinemática directa . . . . .	2
3.2	Cinemática inversa . . . . .	3
<b>4</b>	<b>Código e interfaz</b>	<b>3</b>
4.1	Código implementado en Arduino . . . . .	3
4.2	Interfaz de usuario . . . . .	8
<b>5</b>	<b>Pruebas y demostración</b>	<b>9</b>

# 1 Introducción

En este proyecto se diseña y construye un robot manipulador de al menos tres grados de libertad, controlado con servomotores y una placa Arduino UNO. El objetivo es que el robot pueda mover sus articulaciones de manera precisa, calcular la posición del efector final mediante cinemática directa, y determinar los ángulos necesarios para llegar a posiciones deseadas mediante cinemática inversa. Además, se desarrolla una interfaz de usuario que permite controlar el robot, ver su posición en coordenadas articulares y cartesianas, y enviar comandos de movimiento de forma sencilla.

## 2 Descripción del diseño mecánico y electrónico

Hemos optado por el diseño y ensamblaje de un robot angular con 3 grados de libertad, los cuales aparecen como una rotación en la base en el eje x, un movimiento horizontal en el eje z con respecto a la base gracias a un rodamiento que hemos implementado y un movimiento en el eje x del codo del robot, también gracias a un rodamiento implementado.

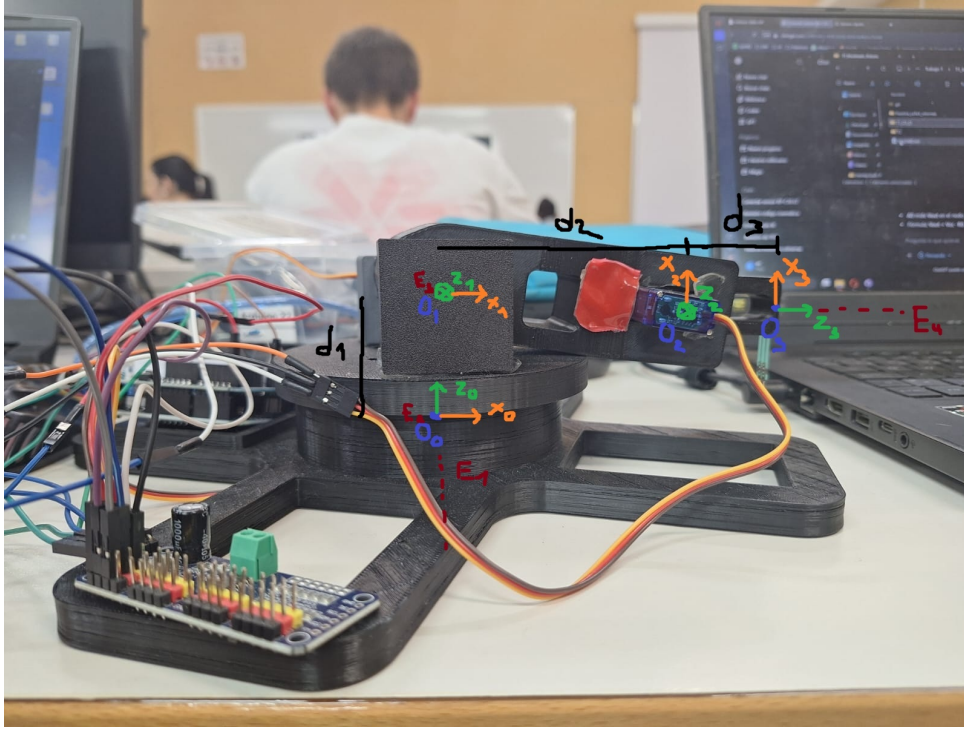
Utilizamos 3 servos, uno bajo la base para realizar el giro, otro alineado con el rodamiento del brazo para llevar a cabo el movimiento y otro alineado con el rodamiento del codo, también para su movimiento. En cuanto al cableado cada servomotor se conecta al Arduino mediante tres cables: alimentación (+5V), tierra (GND) y señal. Los pines de señal de cada servo se han conectado a pines digitales del Arduino, mientras que todas las tierras se unifican en común para garantizar un funcionamiento estable.



*Figura 1: robot en plenitud*

Hay varias maneras de definir la posición inicial, nosotros hemos optado por definir una posición conocida como es la  $[0,0,0]$ , donde la posición del brazo y del codo son horizontales como se muestra en la primera figura, pudiendo calcular así las distancias y facilitar la calibración y el movimiento del robot.

### 3 Modelo matemático



*Figura 2: Ejes del robot*

$i$	$d_i$	$\theta_i$	$\alpha_i$	$a_i$
1	$d_1$	0	$\frac{\pi}{2}$	0
2	0	0	0	$d_2$
3	0	0	0	$d_3$

#### 3.1 Cinemática directa

La cinemática directa permite calcular la posición y orientación del efector final (la pinza) a partir de los ángulos y desplazamientos de las articulaciones del robot.

En nuestro caso, el robot tiene 3 grados de libertad:

- Rotación en la base (eje X): permite girar todo el brazo alrededor de la base.
- Movimiento horizontal en el eje Z: conseguido mediante un rodamiento implementado en la base, que desplaza el brazo a lo largo del eje Z.
- Movimiento en el eje X del codo: logrado mediante otro rodamiento, que permite extender o retraer el brazo a lo largo del eje X desde el codo.

La posición del efector final  $(x, y, z)$  se calcula combinando estas tres transformaciones mediante matrices de transformación homogénea. Multiplicando las matrices correspondientes a cada articulación se obtiene la matriz global  $T_0^3$ , de la que se extraen las coordenadas de la pinza.

### 3.2 Cinemática inversa

La cinemática inversa permite determinar los ángulos y desplazamientos de las articulaciones necesarios para colocar la pinza en una posición y orientación deseadas.

Para nuestro robot:

- Debemos calcular la rotación de la base para orientar el brazo hacia el punto deseado.
- Ajustar el desplazamiento horizontal en Z para posicionar la altura correcta.
- Ajustar el movimiento en X del codo para llegar a la distancia correcta desde la base.

Se deben considerar posibles múltiples soluciones, así como restricciones físicas de los rodillos y límites de las articulaciones. La cinemática inversa asegura que el robot pueda mover su efector a cualquier posición alcanzable dentro de su espacio de trabajo.

## 4 Código e interfaz

### 4.1 Código implementado en Arduino

*Listing 1: Configuración inicial y variables globales.*

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

%% Controlador PCA9685
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

%% Canales para cada servo en el PCA9685
#define SERVO_BASE 0
#define SERVO_BRAZO 1
#define SERVO_CODO 2

%% Matriz DH cinemática directa [alpha, a, d, theta]
float Matriz_DH_CD[3][4];
const int N = 3;
const int N1 = 4;

%% Matriz resultado
float res_CN[N1][N1];

%% Angulos actuales
int baseAngle = 0;
int brazoAngle = 0;
int codoAngle = 0;

%% Distancia entre ejes
float d1 = 0.075;
float d2 = 0.082;
```

```

float d3 = 0.050;

%% Resultados de la cinemática inversa
float theta1, theta2, theta3;
const int step = 5;
int modo = 0; %% 0 = directa, 1 = inversa

%% Mapear ángulo a pulso
int angleToPulse(int ang) {
    int pulsoMin = 150; %% 0 grados
    int pulsoMax = 600; %% 180 grados
    return map(ang, 0, 180, pulsoMin, pulsoMax);
}

%% Wrapper para mover servo
void moverServo(int canal, int ángulo) {
    int pulso = angleToPulse(ángulo);
    pwm.setPWM(canal, 0, pulso);
}

```

*Listing 2: Función de Cinemática Inversa (ik\_RRR).*

```

%% Función de cinemática inversa RRR
bool ik_RRR(float X, float Y, float Z) {
    const float a2 = d2;
    const float a3 = d3;
    const float h = d1;

    float th1 = atan2f(Y, X);

    float r = sqrtf(X*X + Y*Y);
    float z = Z - h;

    float D = (r*r + z*z - a2*a2 - a3*a3) / (2.0f * a2 * a3);

    %% Verificación de dominio matemático
    if (D > 1.0f) D = 1.0f;
    if (D < -1.0f) D = -1.0f;
    if (isnan(D)) return false;

    float s3 = sqrtf(fmaxf(0.0f, 1.0f - D*D));
    float th3_geom = atan2f(-s3, D); %% codo abajo

    float num = a3 * sinf(th3_geom);
    float den = a2 + a3 * cosf(th3_geom);
    float th2_geom = atan2f(z, r) - atan2f(num, den);

    const float RAD2DEG = 180.0f / PI;
    float g1 = th1 * RAD2DEG;
    float g2 = th2_geom * RAD2DEG;
    float g3 = th3_geom * RAD2DEG;

    %% Normalización de ángulos
    if (g1 < 0.0f) g1 += 360.0f;
    if (g1 > 180.0f) g1 = 360.0f - g1;

    float servo_base = g1;
    float servo_brazo = g2;
    float servo_codo = -g3;

    %% Restricción de rango físico
    servo_base = constrain(servo_base, 0.0f, 180.0f);
    servo_brazo = constrain(servo_brazo, 0.0f, 180.0f);
}

```

```

servo_codo = constrain(servo_codo, 0.0f, 180.0f);

theta1 = servo_base;
theta2 = servo_brazo;
theta3 = servo_codo;

return true;
}

```

*Listing 3: Funciones para manejo de matrices y DH.*

```

%% Inicializar matriz DH
void inicializar_matriz(){
    Matriz_DH_CD[0][0] = M_PI/2;
    Matriz_DH_CD[0][1] = 0.0;
    Matriz_DH_CD[0][2] = d1;
    Matriz_DH_CD[0][3] = 0.0;

    Matriz_DH_CD[1][0] = 0.0;
    Matriz_DH_CD[1][1] = d2;
    Matriz_DH_CD[1][2] = 0.0;
    Matriz_DH_CD[1][3] = 0.0;

    Matriz_DH_CD[2][0] = 0.0;
    Matriz_DH_CD[2][1] = d3;
    Matriz_DH_CD[2][2] = 0.0;
    Matriz_DH_CD[2][3] = 0.0;
}

%% Convertir parámetros DH en matriz homogénea
void dh_to_T(float DH[4][4], float alpha, float a, float d, float theta){
    float ca = cos(alpha);
    float sa = sin(alpha);
    float ct = cos(theta);
    float st = sin(theta);

    DH[0][0] = ct;      DH[0][1] = -st * ca;    DH[0][2] = st * sa;    DH[0][3] = a * ct;
    DH[1][0] = st;      DH[1][1] = ct * ca;    DH[1][2] = -ct * sa;  DH[1][3] = a * st;
    DH[2][0] = 0.0;     DH[2][1] = sa;         DH[2][2] = ca;        DH[2][3] = d;
    DH[3][0] = 0.0;     DH[3][1] = 0.0;        DH[3][2] = 0.0;       DH[3][3] = 1.0;
}

%% Multiplicar matrices 4x4
void multiplicarMatrices(float A[N1][N1], float B[N1][N1], float C[N1][N1]) {
    for (int i = 0; i < N1; i++) {
        for (int j = 0; j < N1; j++) {
            C[i][j] = 0;
            for (int k = 0; k < N1; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

```

*Listing 4: Cálculo de la Cinemática Directa.*

```

%% Calcular cinemática directa
void calcular_directa(){
    float alpha[N], a[N], d[N], theta[N];
    float A1[N1][N1], A2[N1][N1], A3[N1][N1];
    float T0[N1][N1], T1[N1][N1];
}

```

```

for (int i = 0; i < N; i++) {
    alpha[i] = Matriz_DH_CD[i][0];
    a[i]      = Matriz_DH_CD[i][1];
    d[i]      = Matriz_DH_CD[i][2];
    theta[i] = Matriz_DH_CD[i][3];
}

theta[0] += baseAngle * M_PI / 180.0;
theta[1] += brazoAngle * M_PI / 180.0;
theta[2] += -codoAngle * M_PI / 180.0;

dh_to_T(A1, alpha[0], a[0], d[0], theta[0]);
dh_to_T(A2, alpha[1], a[1], d[1], theta[1]);
dh_to_T(A3, alpha[2], a[2], d[2], theta[2]);

float identidad[N1][N1] = {
    {1.0, 0.0, 0.0, 0.0},
    {0.0, 1.0, 0.0, 0.0},
    {0.0, 0.0, 1.0, 0.0},
    {0.0, 0.0, 0.0, 1.0}
};

multiplicarMatrices(identidad, A1, T0);
multiplicarMatrices(T0, A2, T1);
multiplicarMatrices(T1, A3, res_CN);
}

```

*Listing 5: Configuración (Setup) y Bucle Principal (Loop).*

```

%% Configuración inicial
void setup() {
    Serial.begin(9600);
    pwm.begin();
    pwm.setPWMFreq(60);

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

    inicializar_matriz();
}

%% Bucle principal
void loop() {
    if (Serial.available()) {
        String cmd = Serial.readStringUntil('\n');
        cmd.trim();

        if (cmd == "0") {
            modo = 0;
        }
        else if (cmd == "1") {
            modo = 1;
        }
        %% Modo inverso: SET,X,Y,Z
        else if (cmd.startsWith("SET,") && modo == 1) {
            cmd.remove(0, 4);
            int coma1 = cmd.indexOf(',');
            int coma2 = cmd.lastIndexOf(',');

            if (coma1 > 0 && coma2 > coma1) {
                float b = cmd.substring(0, coma1).toFloat();
            }
        }
    }
}

```



```

        float r = cmd.substring(coma1 + 1, coma2).toFloat();
        float c = cmd.substring(coma2 + 1).toFloat();

        bool func = ik_RRR(b, r, c);

        if (func == true){
            moverServo(SERVO_BASE, theta1);
            moverServo(SERVO_BRAZO, theta2);
            moverServo(SERVO_CODO, theta3);
        }
        baseAngle = theta1;
        brazoAngle = theta2;
        codoAngle = theta3;
    }
}

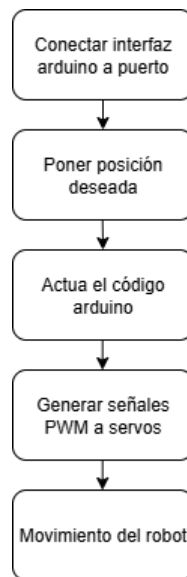
%% Controles manuales
else if (cmd == "Base+" && modo == 0) {
    baseAngle = constrain(baseAngle + step, 0, 180);
    moverServo(SERVO_BASE, baseAngle);
}
else if (cmd == "Base-" && modo == 0) {
    baseAngle = constrain(baseAngle - step, 0, 180);
    moverServo(SERVO_BASE, baseAngle);
}
else if (cmd == "Brazo+" && modo == 0) {
    brazoAngle = constrain(brazoAngle + step, 0, 180);
    moverServo(SERVO_BRAZO, brazoAngle);
}
else if (cmd == "Brazo-" && modo == 0) {
    brazoAngle = constrain(brazoAngle - step, 0, 180);
    moverServo(SERVO_BRAZO, brazoAngle);
}
else if (cmd == "Codo+" && modo == 0) {
    codoAngle = constrain(codoAngle + step, 0, 180);
    moverServo(SERVO_CODO, codoAngle);
}
else if (cmd == "Codo-" && modo == 0) {
    codoAngle = constrain(codoAngle - step, 0, 180);
    moverServo(SERVO_CODO, codoAngle);
}
}

calcular_directa();

Serial.print(baseAngle);
Serial.print(",");
Serial.print(brazoAngle);
Serial.print(",");
Serial.print(codoAngle);
Serial.print(",");
Serial.print(res_CN[0][3]);
Serial.print(",");
Serial.print(res_CN[1][3]);
Serial.print(",");
Serial.print(res_CN[2][3]);
Serial.print("\n");

delay(500);
}

```



*Figura 3: Diagrama de funcionamiento*

## 4.2 Interfaz de usuario

### Control y Lectura desde Arduino para el robot

#### Conectar con Arduino

Estado: ●

#### Selección de Cinemática

☐ Cinemática directa

Base:  + -

Brazo:  + -

Codo:  + -

Posición X con Cinemática Directa:  Posición Y con Cinemática Directa:  Posición Z con Cinemática Directa:

#### Enviar ángulos cinemática inversa

X en mm:  Y en mm:  Z en mm:  Enviar

#### Ángulos para la posición en cinemática inversa

Base:  Brazo:  Codo:  Resultado:

*Figura 4: Interfaz de usuario*

Esta es la interfaz de usuario que hemos diseñado, tenemos en primera parte un botón para conectar con el robot, junto con un icono de color que nos indica visualmente si la conexión ha sido exitosa o no. debajo de eso elegimos el tipo de cinemática, si la opción está pulsada es cinemática directa, en caso contrario es cinemática inversa; debajo de eso tenemos cajas de texto ajustables para base, brazo y codo del robot, todas junto a iconos de + y -, donde podemos mover paso a paso individualmente cada parte en incrementos o decrementos de 5 grados, acompañado con eso debajo cajas de texto que indican las posición del robot en los ejes X Y y Z.

En cuanto a la parte de la cinemática inversa tenemos cajas de texto donde modificamos el valor de las posiciones de X,Y y Z medidas en milímetros, a lo que después presionamos el botón

enviar, lo que mueve el robot a las posiciones elegidas, mostrando en las cajas de abajo el ángulo en la base el brazo y el codo, además del resultado del cálculo de la cinemática inversa

## 5 Pruebas y demostración

En esta sección presentaremos pruebas del funcionamiento del robot

[Link a video de cinemática directa](#)

[Link a video de cinemática inversa](#)