UNIVERSIDAD COMPLUTENSE MADRID

UNED ETS de Ingeniería Informática

Trabajo final sobre la materia

Máster en Ingeniería de Sistemas y Control

Asignatura: Robótica Indsutrial

Curso: 2025/2026

Alumno: Javier Arambarri Calvo

17 de diciembre de 2025

Autoría

El autor de este documento es Javier Arambarri Calvo, alumno del Máster en Ingeniería de Sistemas y Control y matriculado por la Universidad Complutense de Madrid en el curso académico 2025-2026.

Para que así conste, se firma digitalmente este documento.

Nota

REVISAR ESTO!!!!!!!!!!!!! Las imágenes son propias, generadas bien con *draw.io* [1] o *Matlab* [2]. Para completar la comprensión del tema 7 se ha utilizado el siguiente vídeo [3].

Índice

1. Introducción **7**
1.1. Objetivo y motivación del proyecto . . . 7
1.2. Metodología . . . 7

2. *UFactory xArm 6* **8**
2.1. Estructura mecánica: eslabones y articulaciones . . . 8
2.2. Actuadores . . . 9
2.3. Sensores integrados y opcionales . . . 10
2.4. Efector . . . 11
2.5. Sistema de control . . . 12
2.6. *Software* . . . 13

3. Estudio cinemático **14**
3.1. Cinemática directa . . . 14
3.1.1. Parámetros de Denavit-Hartenberg . . . 14
3.1.2. Parámetros modificados de Denavit-Hartenberg . . . 17
3.2. Cinemática inversa . . . 21
3.2.1. Pseudoinversa del Jacobiano y Newton–Raphson . . . 24
3.2.2. Cálculo de cinemática inversa para *UFactory xArm6* . . . 26
3.3. Cinemática diferencial y Jacobiano . . . 30
3.4. Generación de trayectorias . . . 33
3.4.1. **Trayectorias lineales en espacio cartesiano** . . . 33
3.4.2. **Trayectorias circulares en espacio cartesiano** . . . 33
3.4.3. Corrección de discontinuidades mediante *IPTP* . . . 33
3.4.4. **Trayectorias polinómicas en espacio articular** . . . 34
3.4.5. **Trayectorias *spline* (*multi-waypoint*)** . . . 35
3.5. Cálculo de la trayectoria en *Matlab* con la *toolbox* de Peter Corke . . . 36
3.6. Control cinemático . . . 40

4. Estudio dinámico **48**

Referencias adicionales al material del Campus Virtual **50**

Índice de figuras

1. *UFactory xArm 6*. Fuente: [10]. 8
2. *Reductor armónico*. Fuente: [11]. 9
3. Ejemplo de elemento terminal. Fuente: [4]. 11
4. *Control box* analógico. Fuente: [4]. 12
5. *Control box* digital. Fuente: [4]. 12
6. *xArm Studio*. Fuente: [4]. 13
7. Parámetros clásicos Denavit-Hartenberg. Fuente: [9]. 14
8. Parámetros modificados Denavit-Hartenberg. Fuente: [9]. 18
9. Trayectoria articular realizable. 39
10. Trayectoria realizable del efector. 39
11. Trayectoria articular realizable. 40
12. Trayectoria realizable del efector. 41
13. Velocidad cartesiana del efector. 41
14. Velocidades articulares. 42
15. Aceleración cartesiana del efector. 42
16. Aceleraciones articulares. 43

Índice de cuadros

1. Características mecánicas y de actuadores del *xArm 6*. 9
2. Resumen de tipo de señal, comunicación y alimentación de los sensores del *xArm 6*. 10
3. Resumen de tipo de señal, comunicación y alimentación de los elementos terminales del efector del *xArm 6*. 11
4. Resumen de las especificaciones técnicas de las unidades de control del *xArm 6*. . 12

Índice de códigos

1. Código Matlab para la cinemática directa con parámetros DH clásicos. 16
2. Salida numérica de la cinemática directa DH clásica. 16
3. Código Matlab para la cinemática directa con parámetros DH modificados. 20
4. Salida numérica de la cinemática directa DH modificada. 20
5. Código Matlab para obtener las ecuaciones de posición y orientación con parámetros DH modificados. 23
6. Cálculo de la cinemática inversa del robot *UFactory xArm6*. 27
7. Salida del cálculo de la cinemática inversa del robot *UFactory xArm6*. 29
8. Cálculo de la cinemática diferencial y Jacobiano mediante parámetros DH modificados. 30
9. Generación de trayectoria articular realizable. 36
10. Salida de la generación de trayectoria articular realizable. 38
11. Control cinemático completo. 43
12. Salida del control cinemático completo. 47

1. Introducción

Este proyecto se enmarca en la asignatura de Robótica Industrial del Máster en Ingeniería de Sistemas y Control. De entre las opciones planteada, este responde al tercero: *"estudio sobre un tema libre."*

Para la realización del trabajo se ha utilizado como base el robot *UFactory xArm 6* [4] en simulación mediante el *framework* ROS2 [5] con el objetivo de realizar un estudio cinemático y dinámico que lo permita programar. En otras palabras, este proyecto trabaja sobre un robot real.

1.1. Objetivo y motivación del proyecto

El objetivo del proyecto es trasladar los conceptos teóricos y prácticos estudiados en la asignatura a un robot real para aprender a trabajar con máquinas reales.

Este proyecto viene motivado por el trabajo que desempeña el estudiante en la Escuela de Ingeniería de Bilbao, donde se va a comenzar a trabajar con el citado brazo robótico para proyectos de investigación.

1.2. Metodología

Dado que por el momento no se disponer del robot montado y configurado, se ha optado por trabajar en simulación empleando el *software* Gazebo [6] y la distribución *Jazzy Jalisco* [7] de ROS2. Se ha utilizado la documentación oficial del fabricante, tanto el repositorio de ROS2 [8] como el manual de usuario del robot [9].

Los ejercicios o aplicaciones se han desarrollado en *Python? C++?...*

2. *UFactory xArm 6*

El *UFactory xArm 6*, figura 1, es un robot colaborativo de seis grados de libertad diseñado para aplicaciones de investigación, formación y automatización industrial. Se trata de un manipulador compacto y versátil, capaz de realizar movimientos complejos en tres dimensiones gracias a su configuración de seis articulaciones rotacionales.

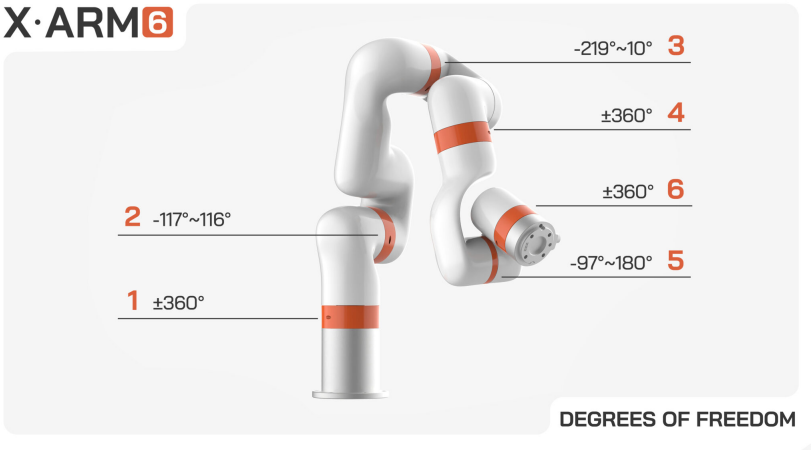


Figura 1: *UFactory xArm 6*. Fuente: [10].

2.1. Estructura mecánica: eslabones y articulaciones

La estructura mecánica del robot se compone de un cuerpo ligero fabricado en aluminio y fibra de carbono, con un peso total cercano a los 12.5 kg, lo que facilita su instalación en entornos de laboratorio o docencia. Los distintos eslabones del brazo están formados por piezas huecas unidos mediante carcasas mecanizadas que protegen los actuadores. El conjunto está diseñado para ofrecer rigidez estructural y al mismo tiempo mantener un peso reducido, lo que mejora la relación entre carga útil y masa propia del robot.

El manipulador consta de seis eslabones principales, correspondientes a las seis articulaciones rotacionales que le otorgan sus grados de libertad. Cada eslabón está conectado al siguiente mediante un eje motorizado con transmisión armónica y por tanto, cada articulación está equipada con un motor eléctrico de corriente continua y un reductor armónico de alta precisión. A pesar de tener los actuadores directamente en cada articulación, no se puede considerar de accionamiento directo debido al uso de reductores armónicos en todas las articulaciones.

En cuanto a su apariencia, el robot se presenta en un acabado de color blanco con detalles en color oscuro en las juntas y carcasas de los motores, lo que le confiere un aspecto moderno y uniforme.

El rango de movimiento de las articulaciones, tabla 1, abarca desde la base hasta la muñeca, con amplitudes que permiten cubrir un radio de trabajo de aproximadamente 700 mm. En concreto, la articulación de la base ofrece un giro completo de $\pm 360°$, mientras que las articulaciones intermedias permiten rotaciones de hasta $\pm 180°$, y las articulaciones de muñeca alcanzan rangos de $\pm 360°$, lo que proporciona gran flexibilidad para tareas de manipulación y orientación del efector final.

2.2. Actuadores

Cada articulación del *xArm 6* está equipada con un motor eléctrico de corriente continua combinado con un reductor armónico de alta precisión. Esta configuración permite alcanzar una elevada rigidez torsional y eliminar prácticamente el retroceso mecánico, garantizando movimientos suaves y precisos en tareas de manipulación y ensamblaje. Los motores, integrados directamente en cada eje, proporcionan pares máximos que varían entre 1.5 Nm en las articulaciones de la muñeca y hasta 8.4 Nm en las articulaciones de base y hombro, lo que asegura la capacidad de transportar cargas de hasta 5 kg sin comprometer la repetibilidad del sistema.

La potencia nominal del conjunto es de 150 W, distribuida entre las seis articulaciones, mientras que la velocidad máxima de giro alcanza los 180°/s. Estos valores, junto con la repetibilidad de $\pm 0{,}1$ mm en sus trayectorias, hacen del *xArm 6* un manipulador adecuado para aplicaciones de investigación, docencia y procesos industriales ligeros que requieren gran precisión. En la tabla 1 se resumen las principales características mecánicas y de actuadores, incluyendo el rango de trabajo de cada articulación, el par máximo disponible y los parámetros globales de carga útil, potencia y repetibilidad.

Los reductores armónicos [11], figura 2, se basan en la deformación elástica controlada de un componente flexible para transmitir el movimiento. Este principio permite alcanzar relaciones de reducción muy elevadas en un volumen compacto, con una rigidez torsional superior a la de otros sistemas de engranajes. En el *xArm 6*, su empleo asegura un movimiento suave y preciso, además de minimizar el retroceso mecánico (*backlash*), lo que resulta fundamental para aplicaciones de ensamblaje, manipulación de piezas y tareas de investigación que requieren gran exactitud.
Otra ventaja de los reductores armónicos es su capacidad para soportar cargas elevadas en relación con su tamaño, lo que contribuye a que el robot pueda mantener una carga útil de hasta 5 kg sin comprometer la precisión. Asimismo, su diseño compacto permite integrar el actuador y el reductor dentro de cada articulación, reduciendo el volumen total del brazo y facilitando su instalación en espacios reducidos.

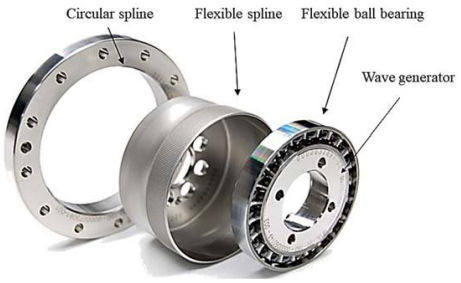


Figura 2: *Reductor armónico*. Fuente: [11].

Parámetro	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Rango de trabajo	± 360°	-117° – 116°	-219° – 10°	± 360°	-97° – 180°	± 360°
Par máximo	8.4 Nm	8.4 Nm	4.2 Nm	1.5 Nm	1.5 Nm	1.5 Nm
Vel. y acel. máxima	180 °/s, 1500 °/s²					
Carga útil máxima	5 kg					
Potencia nominal	150 W					
Repetibilidad	± 0.1 mm					

Cuadro 1: Características mecánicas y de actuadores del *xArm 6*.

2.3. Sensores integrados y opcionales

El *xArm 6* incorpora sensores básicos de posición en cada articulación mediante *encoders* absolutos digitales, que permiten conocer de forma directa y precisa la posición angular de cada eje sin necesidad de realizar un proceso de referencia tras el encendido [12]. En estos codificadores el disco transparente se divide en un número determinado de sectores, siempre potencia de 2, cada uno codificado según un código binario cíclico, normalmente *código Gray*, representado por zonas transparentes y opacas dispuestas radialmente. De este modo, cada posición queda codificada de forma única y absoluta, sin necesidad de contadores ni electrónica adicional para detectar el sentido de giro.

Además, dispone de funciones internas de detección de colisión por *software* que garantizan la seguridad durante la operación. Estas funciones monitorizan las corrientes de los motores y detienen el movimiento si se detecta un contacto inesperado, lo que generaría un aumento brusco de corriente provocado por la resistencia mecánica. Por tanto, el robot no anticipa la colisión, sino que la detecta una vez se ha producido.

El fabricante también ofrece una gama de sensores opcionales que amplían las capacidades del robot. Entre ellos destaca el sensor de fuerza o torque de seis ejes, instalado en la brida del manipulador, capaz de medir fuerzas y momentos en las tres direcciones espaciales (Fx, Fy, Fz, Mx, My, Mz). Este sensor resulta fundamental en tareas de ensamblaje, manipulación delicada y control por contacto.

Otros sensores que se pueden incorporar son los sensores de presión analógicos en el *gripper* de vacío, que permiten verificar la correcta sujeción de piezas mediante succión, así como sensores digitales de fuerza en los *gripper*s mecánicos y bio *gripper*s, que controlan la intensidad del agarre para evitar daños en los objetos manipulados, y la integración de cámaras en el extremo del brazo, destinadas a aplicaciones de visión artificial e inspección.

En el cuadro 2 se recogen las características resumidas de comunicación y alimentación de estos sensores.

Sensor	Tipo de señal	Comunicación	Alimentación
Encoder absoluto	Digital	Interna	Integrada en cada motor
Detección de colisión	Virtual	Interna	Alimentación del actuador
Fuerza o torque 6 ejes	Digital	*USB* / *Ethernet*	*Hub* del robot
Sensor de presión	Analógica / Digital	Interna	*Control box* / *gripper*
Sensor de fuerza	Digital	Interna	*Control box* / *gripper*
Cámara	Digital	*USB* / *Ethernet*	*Hub* del robot o fuente externa

Cuadro 2: Resumen de tipo de señal, comunicación y alimentación de los sensores del *xArm 6*.

2.4. Efector

El efector final del *xArm 6* corresponde a la brida situada en el extremo del manipulador, diseñada con interfaces mecánicas estándar que permiten la conexión de una amplia variedad de herramientas. Entre los efectores más habituales se encuentran las pinzas mecánicas, los *grippers* de vacío y los bio *grippers*, todos ellos disponibles como accesorios oficiales del fabricante. Asimismo, el sistema admite la instalación de cámaras en la brida para aplicaciones de visión artificial e inspección.

En la figura 6 se observa la pinza mecánica junto con la cámara *Intel RealSense D435* [13], recomendada por el fabricante.

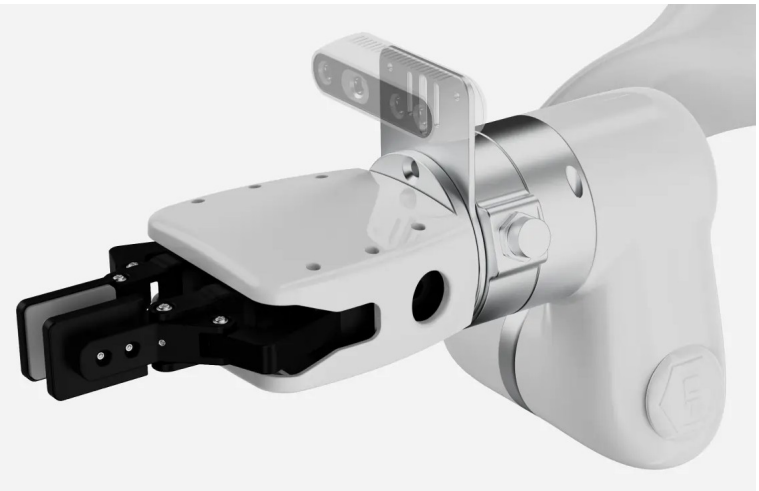


Figura 3: Ejemplo de elemento terminal. Fuente: [4].

La brida del robot cumple con el patrón de taladros indicado en la ISO 9409-1:2004 [14], lo que facilita la integración de herramientas de terceros y asegura la compatibilidad con dispositivos de medida como el sensor de fuerza o torque de seis ejes. Este sensor se instala directamente en el efector y permite medir fuerzas y momentos en las tres direcciones espaciales, ampliando las capacidades del manipulador en tareas de ensamblaje y manipulación delicada.

El efector final del *xArm 6* está diseñado para soportar una carga útil máxima de 5 kg, así como un momento máximo de 10 Nm en la muñeca, garantizando un funcionamiento seguro dentro de los límites especificados por el fabricante.

En el cuadro 3 se recogen las características resumidas de comunicación y alimentación de estos elementos terminales.

Elemento terminal	Tipo de señal	Comunicación	Alimentación
Pinza mecánica	Digital	Interna	*Control box* / *gripper*
Gripper de vacío	Analógica / Digital	Interna	*Control box* / *gripper*
Bio *gripper*	Digital	Interna	*Control box* / *gripper*

Cuadro 3: Resumen de tipo de señal, comunicación y alimentación de los elementos terminales del efector del *xArm 6*.

2.5. Sistema de control

El *xArm 6* se acompaña de dos unidades de *hardware* externas que permiten su operación e integración con accesorios: el *control box* y el *hub*.

El *control box* constituye la unidad de control principal del robot, alojando la electrónica de potencia y el controlador encargado de gestionar los actuadores y sensores y las funciones de seguridad, además de suministrar la energía necesaria al manipulador, incorporar puertos de comunicación *Ethernet* y *USB* y el botón de parada de emergencia para garantizar un uso seguro.

Se presenta en dos versiones según el tipo de alimentación eléctrica requerida. El *control box AC*, figura 4, que se conecta directamente a la red eléctrica (100–240 V AC), lo que permite un uso inmediato en entornos de laboratorio o producción ligera sin necesidad de fuentes externas adicionales, y el *control box DC*, figura 5, que está diseñado para sistemas que operan con corriente continua de 24 V, siendo más compacto y ligero, lo que facilita su integración en plataformas móviles o aplicaciones embebidas, pero requiere de una fuente externa de 24 V DC.

Por su parte, el *hub* se instala en el extremo del brazo, junto a la brida, y actúa como módulo de expansión para la conexión de accesorios que requieren estar en el efector final, como cámaras o el sensor de fuerza o torque de seis ejes. Este dispositivo está conectado por cableado interno al *control box*, del cual recibe tanto la alimentación en 24 V DC como la comunicación con el controlador del robot. De esta forma, los accesorios pueden integrarse sin necesidad de cableado externo adicional. Cabe señalar que los grippers oficiales, por defecto, no se conectan al *hub*, ya que su control está integrado en el *firmware* del *control box*.

El modo de operación habitual del *xArm 6* consiste en conectar el *control box* a la red eléctrica y establecer la comunicación con el ordenador de control a través de la interfaz *Ethernet*, utilizando una dirección IP asignada en la red local. Aunque también es posible la conexión directa por *USB*, la comunicación por *IP* resulta más versátil y constituye el modo de operación más extendido en aplicaciones industriales y colaborativas.

En la tabla 4 se recogen resumidamente las características de las unidades de control.

Figura 4: *Control box* analógico. Fuente: [4].

Figura 5: *Control box* digital. Fuente: [4].

Componente	Alimentación	Comunicación
Control Box (AC)	100–240 V AC	*Ethernet* / *USB*
Control Box (DC)	24 V DC	*Ethernet* / *USB*
Hub en la brida	24 V DC (desde el robot)	*Ethernet* / *USB*

Cuadro 4: Resumen de las especificaciones técnicas de las unidades de control del *xArm 6*.

2.6. *Software*

El robot dispone de interfaces de control que permiten su programación tanto en *Python* como en *C++*, además de una integración nativa con el *framework ROS2* a través del repositorio oficial del fabricante [8]. La comunicación con el manipulador se realiza principalmente a través de los puertos *Ethernet* y *USB* del *control box*, mediante los cuales se transmiten las órdenes de movimiento y se reciben datos de estado y retroalimentación de los sensores.

El fabricante proporciona un conjunto de librerías y *APIs* que facilitan la programación de trayectorias, el control de efectores finales y la integración con sistemas externos. Asimismo, se incluye una interfaz gráfica denominada *xArm Studio* [15], que permite la configuración inicial, la calibración y la ejecución de programas de manera intuitiva, sin necesidad de conocimientos avanzados de programación, así como la programación por bloques visuales de color. Estas herramientas convierten al *xArm 6* en una plataforma abierta y flexible, apta tanto para entornos académicos como industriales.

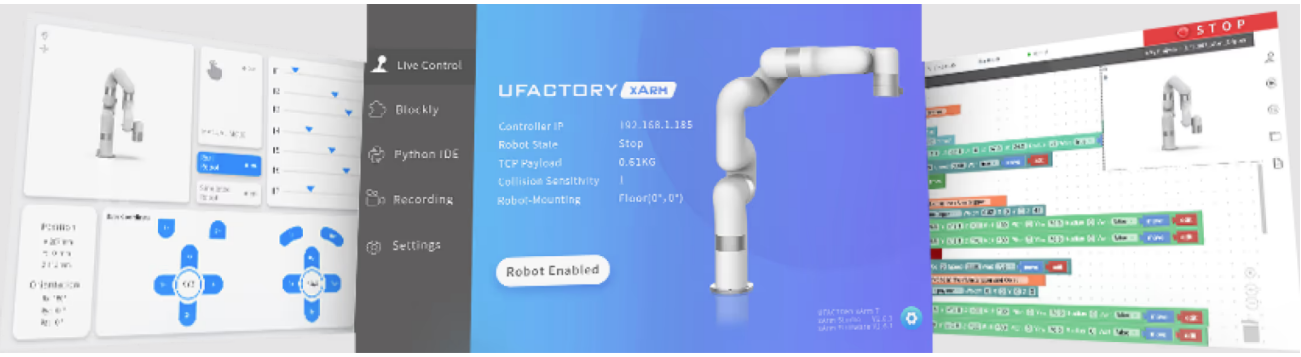


Figura 6: *xArm Studio*. Fuente: [4].

3. Estudio cinemático

El fabricante proporciona los parámetros clásicos y modificados de Denavit-Hartenberg en el manual del usuario. En las siguientes secciones se van a estudiar dichos parámetros.

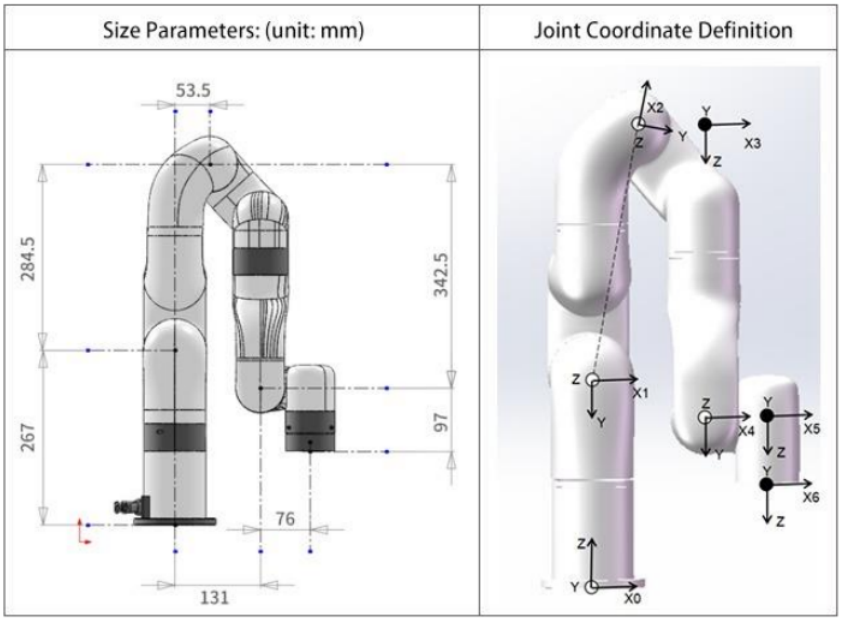
3.1. Cinemática directa

3.1.1. Parámetros de Denavit-Hartenberg

En la figura 7 se presentan los parámetros clásicos de Denavit-Hartenberg [12] proporcionados por el fabricante, con

$$a_2 = \sqrt{284{,}5^2 + 53{,}5^2} = 289{,}48866$$

$$T_{2,\text{offset}} = -\arctan\left(\frac{284{,}5}{53{,}5}\right) = -1{,}3849179\ (-79{,}34995°);\ T_{3,\text{offset}} = -T_{2,\text{offset}}.$$



Kiematics	theta (rad)	d (mm)	alpha (rad)	a (mm)	offset (rad)
Joint1	0	267	-pi/2	0	0
Joint2	0	0	0	a2	T2_offset
Joint3	0	0	-pi/2	77.5	T3_offset
Joint4	0	342.5	pi/2	0	0
Joint5	0	0	-pi/2	76	0
Joint6	0	97	0	0	0

Figura 7: Parámetros clásicos Denavit-Hartenberg. Fuente: [9].

Para alcanzar los sitemas 2 y 3 se introduce un *offset* dado que en la imagen la tercera articulación no están representada en el cero, por lo que el *offset* es el ángulo de compensación de la articulación desde la posición matemática cero hasta la posición mecánica cero que se muestra en la imagen. Esto se debe a que si se colocase en el cero, en la imagen quedaría detrás de otros eslabones y no se visualizaría. En otras palabras, el *offset* angular corrige la orientación sin necesidad de modificar el ángulo θ de la articulación y se suma directamente a θ_i.

Además, a partir del análisis de la figura 7 se observa la particularidad en la ubicación del origen de algunos sistemas, como 3 y 4. Esta elección, aunque poco convencional, facilita el cálculo de los parámetros al estar directamente fundamentada en las dimensiones geométricas del robot. De este modo, el parámetro d_4 coincide con la distancia medida que aparece en la imagen de los parámetros de tamaño. Este hecho se repite para el resto de de parámetros longitudinales d_1, a_3, a_5, d_6.

La secuencia de transformaciones [16] y la matriz de transformación homogénea [12] asociada a dichas transformaciones se presentan en las ecuaciones 1 y 2:

$$ {}^{i-1}A_i = \mathbf{R}_Z(\theta_i^* = \theta_i + \mathit{offset}_i)\,\mathbf{T}_Z(d_i)\,\mathbf{T}_X(a_i)\,\mathbf{R}_X(\alpha_i) \tag{1} $$

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i^* & -\sin\theta_i^*\cos\alpha_i & \sin\theta_i^*\sin\alpha_i & a_i\cos\theta_i^* \\ \sin\theta_i^* & \cos\theta_i\cos\alpha_i & -\cos\theta_i^*\sin\alpha_i & a_i\sin\theta_i^* \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

Se calculan a continuación las matrices de transformación del robot *UFactory xArm 6* utilizando los parámetros clásicos de Denavit-Hartenberg:

$$ A_1 = \begin{bmatrix} C(q_1) & 0 & -S(q_1) & 0 \\ S(q_1) & 0 & C(q_1) & 0 \\ 0 & -1 & 0 & 267 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ A_2 = \begin{bmatrix} C(q_2+\text{offset}_2) & -S(q_2+\text{offset}_2) & 0 & a_2\,C(q_2+\text{offset}_2) \\ S(q_2+\text{offset}_2) & C(q_2+\text{offset}_2) & 0 & a_2\,S(q_2+\text{offset}_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_2 = 289{,}48866 $$

$$ A_3 = \begin{bmatrix} C(q_3+\text{offset}_3) & 0 & -S(q_3+\text{offset}_3) & a_3\,C(q_3+\text{offset}_3) \\ S(q_3+\text{offset}_3) & 0 & C(q_3+\text{offset}_3) & a_3\,S(q_3+\text{offset}_3) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_3 = 77{,}5 $$

$$ A_4 = \begin{bmatrix} C(q_4) & 0 & S(q_4) & 0 \\ S(q_4) & 0 & -C(q_4) & 0 \\ 0 & 1 & 0 & 342{,}5 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \; A_5 = \begin{bmatrix} C(q_5) & 0 & -S(q_5) & a_5\,C(q_5) \\ S(q_5) & 0 & C(q_5) & a_5\,S(q_5) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_5 = 76 $$

$$ A_6 = \begin{bmatrix} C(q_6) & -S(q_6) & 0 & 0 \\ S(q_6) & C(q_6) & 0 & 0 \\ 0 & 0 & 1 & 97 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Siendo la transformación total:

$$T_0^6(q) = A_1(q_1)\ A_2(q_2 + \text{offset}_2)\ A_3(q_3 + \text{offset}_3)\ A_4(q_4)\ A_5(q_5)\ A_6(q_6)$$

Podemos calcular y comprobar el resultado en *Matlab* [2] y la *toolbox* de Peter Corke [17]:

```
clear; clc;

% Definir símbolos
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

% Definición de eslabones según tu tabla DH
L1_link = Revolute('d', 267, 'a', 0, 'alpha', -pi2);              % Eslabón 1
L2_link = Revolute('a', a2, 'alpha', 0, 'offset', T2_offset);     % Eslabón 2
L3_link = Revolute('a', 77.5, 'alpha', -pi2, 'offset', T3_offset);% Eslabón 3
L4_link = Revolute('d', 342.5, 'alpha', pi2);                     % Eslabón 4
L5_link = Revolute('a', 76, 'alpha', -pi2);                       % Eslabón 5
L6_link = Revolute('d', 97, 'alpha', 0);                          % Eslabón 6

% Crear el robot
robot = SerialLink([L1_link L2_link L3_link L4_link L5_link L6_link], 'name', 'MiRobot');

% Vector de articulaciones
q = [q1 q2 q3 q4 q5 q6];

% Construir la transformación total manualmente con las matrices .T
T01 = simplify(L1_link.A(q1).T);
T12 = simplify(L2_link.A(q2).T);
T23 = simplify(L3_link.A(q3).T);
T34 = simplify(L4_link.A(q4).T);
T45 = simplify(L5_link.A(q5).T);
T56 = simplify(L6_link.A(q6).T);

% Transformación total simbólica
T06_sym = simplify(T01 * T12 * T23 * T34 * T45 * T56);

disp('Transformación total T_0^6 simbólica:');
disp(T06_sym);

% Sustitución numérica (ejemplo con tus valores y q_i = 0)
S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
S.a2 = 289.48866;
S.T2_offset = deg2rad(-79.34995);
S.T3_offset = deg2rad(79.34995);

T06_num = subs(T06_sym, S);
T06_num = vpa(T06_num, 10);

disp('Transformación total T_0^6 numérica:');
disp(T06_num);
```

Código 1: Código Matlab para la cinemática directa con parámetros DH clásicos.

```
Transformación total T_0^6 simbólica:
... [muy larga para exponer]

Transformación total T_0^6 numérica:
[1.0,    0,    0, 207.0003735]
[  0, -1.0,    0,            0]
[  0,    0, -1.0, 112.0020111]
[  0,    0,    0,          1.0]
```

Código 2: Salida numérica de la cinemática directa DH clásica.

3.1.2. Parámetros modificados de Denavit-Hartenberg

El método clásico de Denavit-Hartenberg, propuesto en 1955, presentaba ciertas limitaciones en la colocación de los sistemas de referencia, especialmente cuando dos ejes consecutivos eran paralelos. En estos casos podían aparecer ambigüedades en la definición de los parámetros y problemas numéricos en las matrices de transformación.

Para superar estas dificultades John J. Craig introdujo el método modificado de Denavit-Hartenberg o convención de Craig [18, 19], que redefine las transformaciones con respecto al sistema i en lugar de hacerlo respecto al sistema $i-1$. El método modificado mantiene los cuatro parámetros $\theta_i, d_i, a_i, \alpha_i$, pero cambia la referencia de los ejes lo que hace más intuitiva la asignación de marcos. El nuevo procedimiento para asignar los sistemas de referencia se resume en los siguientes pasos:

1. Identificar los ejes de las articulaciones.
2. Considerar dos ejes consecutivos (i y $i+1$) e identificar la perpendicular común entre ellos, o bien el punto de intersección. En dicho punto de intersección, o en el punto donde la perpendicular común corta al eje i, se asigna el origen del sistema de enlace.
3. Asignar el eje Z_i apuntando a lo largo del eje de la articulación i.
4. Asignar el eje X_i apuntando a lo largo de la perpendicular común. Si los ejes se intersectan, se asigna X_i como normal al plano que contiene ambos ejes.
5. Asignar el eje Y_i de manera que se complete un sistema de coordenadas con la regla de la mano derecha (sistema dextrógiro).
6. Asignar el sistema $\{0\}$ coincidiendo con $\{1\}$ cuando la primera variable articular sea cero. Para el sistema $\{N\}$, elegir libremente la posición del origen y la dirección de X_N, procurando que el mayor número posible de parámetros de enlace se anulen.

Consecuentemente, la secuencia de transformaciones [16] y la matriz de transformación homogénea asociada a dichas transformaciones se presentan en las ecuaciones 3 y 4:

$$^{i-1}\bar{A}_i = \mathbf{R}_X(\alpha_{i-1})\,\mathbf{T}_X(a_{i-1})\,\mathbf{R}_Z(\theta_i)\mathbf{T}_Z(d_i) \tag{3}$$

$$^{i-1}\mathbf{A}_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos\alpha_{i-1} & \cos\theta_i\cos\alpha_{i-1} & -\sin\alpha_{i-1} & -d_i\sin\alpha_{i-1} \\ \sin\theta_i\sin\alpha_{i-1} & \sin\alpha_{i-1}\cos\theta_i & \cos\alpha_{i-1} & d_i\cos\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

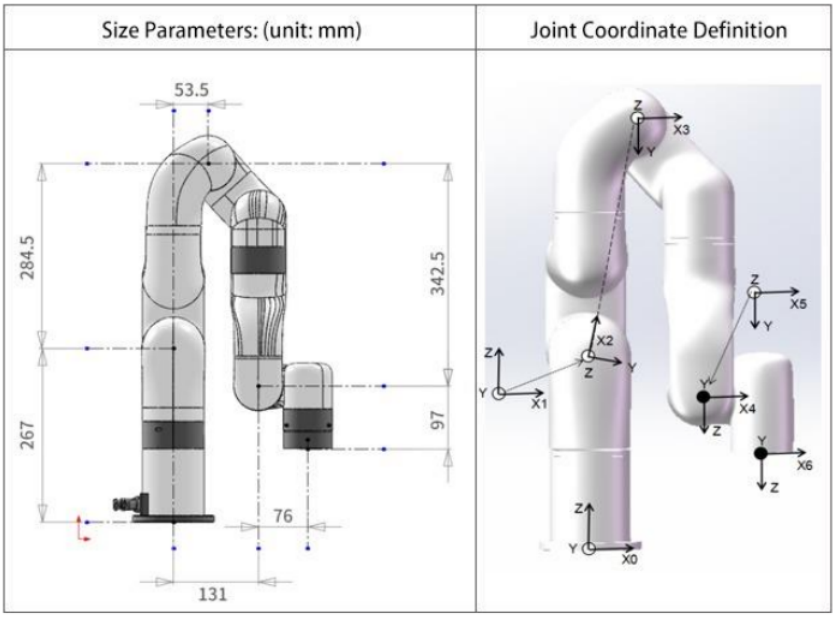
Esta convención ofrece ventajas como la asignación única y consistente de marcos a cada eslabón porque simplifica la implementación computacional y la integración con modelos de diseño asistido por computadora (CAD), donde se prefieren coordenadas locales de eslabón [16]. Además, facilita los cálculos recursivos de cinemática, Jacobianos y dinámica, evitando las ambigüedades de colocación presentes en el método estándar y es la notación más clara y transparente para el análisis mecánico.

En la figura 8 se presentan los parámetros modificados de Denavit-Hartenberg proporcionados por el fabricante, con

$$a_2 = \sqrt{284{,}5^2 + 53{,}5^2} = 289{,}48866$$

$$T_{2,\text{offset}} = -\arctan\left(\frac{284{,}5}{53{,}5}\right) = -1{,}3849179\ (-79{,}34995°)$$

$$T_{3,\text{offset}} = -T_{2,\text{offset}} = 1{,}3849179\ (79{,}34995°).$$



Kiematics	theta (rad)	d (mm)	alpha (rad)	a (mm)	offset (rad)
Joint1	0	267	0	0	0
Joint2	0	0	-pi/2	0	T2_offset
Joint3	0	0	0	a2	T3_offset
Joint4	0	342.5	-pi/2	77.5	0
Joint5	0	0	pi/2	0	0
Joint6	0	97	-pi/2	76	0

Figura 8: Parámetros modificados Denavit-Hartenberg. Fuente: [9].

Al igual que con los parámetros clásicos, sección 3.1.1, para alcanzar los sitemas 2 y 3 se introduce un *offset* dado que en la imagen la tercera articulación no están representada en el cero y se observa la particularidad en la ubicación del origen de algunos sistemas, como 3 y 4, que facilita el cálculo de los parámetros al estar directamente fundamentada en las dimensiones geométricas del robot.

Se calculan a continuación las matrices de transformación del robot *UFactory xArm 6* utilizando los parámetros modificados de Denavit-Hartenberg:

$$A_1 = \begin{bmatrix} C(q_1) & -S(q_1) & 0 & 0 \\ S(q_1) & C(q_1) & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad d_1 = 267$$

$$A_2 = \begin{bmatrix} C(q_2 + T2_{\text{offset}}) & -S(q_2 + T2_{\text{offset}}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -S(q_2 + T2_{\text{offset}}) & -C(q_2 + T2_{\text{offset}}) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} C(q_3 + T3_{\text{offset}}) & -S(q_3 + T3_{\text{offset}}) & 0 & a_2 \\ S(q_3 + T3_{\text{offset}}) & C(q_3 + T3_{\text{offset}}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_2 = 289{,}48866$$

$$A_4 = \begin{bmatrix} C(q_4) & -S(q_4) & 0 & a_3 \\ 0 & 0 & 1 & d_4 \\ -S(q_4) & -C(q_4) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_3 = 77{,}5,\ d_4 = 342{,}5$$

$$A_5 = \begin{bmatrix} C(q_5) & 0 & S(q_5) & 0 \\ S(q_5) & 0 & -C(q_5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6 = \begin{bmatrix} C(q_6) & -S(q_6) & 0 & a_5 \\ 0 & 0 & 1 & d_6 \\ -S(q_6) & -C(q_6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_5 = 76,\ d_6 = 97$$

Siendo la transformación total:

$$T_0^6(q) = A_1(q_1)\ A_2(q_2 + T2_{\text{offset}})\ A_3(q_3 + T3_{\text{offset}})\ A_4(q_4)\ A_5(q_5)\ A_6(q_6)$$

Podemos calcular y comprobar el resultado en *Matlab* [2] y la *toolbox* de Peter Corke [17]:

```
clear; clc;
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

% Link([theta d a alpha sigma offset], 'modified')
L1_link = Link([0     267    0      0        0  0], 'modified');
L2_link = Link([0     0      0     -pi2      0  T2_offset], 'modified');
L3_link = Link([0     0      a2     0        0  T3_offset], 'modified');
L4_link = Link([0     342.5  77.5  -pi2      0  0], 'modified');
L5_link = Link([0     0      0      pi2      0  0], 'modified');
L6_link = Link([0     97     76    -pi2      0  0], 'modified');

robot = SerialLink([L1_link L2_link L3_link L4_link L5_link L6_link], ...
                   'name', 'MiRobot', 'modified');

q = [q1 q2 q3 q4 q5 q6];

T01 = simplify(L1_link.A(q1).T);
T12 = simplify(L2_link.A(q2).T);
T23 = simplify(L3_link.A(q3).T);
T34 = simplify(L4_link.A(q4).T);
T45 = simplify(L5_link.A(q5).T);
T56 = simplify(L6_link.A(q6).T);

T06_sym = simplify(T01 * T12 * T23 * T34 * T45 * T56);

% Sustitución numérica (q_i=0 y offsets reales)
S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
S.a2 = 289.48866;
S.T2_offset = deg2rad(-79.34995);
S.T3_offset = deg2rad(79.34995);

T06_num = vpa(subs(T06_sym, S), 10);
disp(T06_num);
```

Código 3: Código Matlab para la cinemática directa con parámetros DH modificados.

```
[1.0,    0,    0, 207.0003735]
[  0, -1.0,    0,           0]
[  0,    0, -1.0, 112.0020111]
[  0,    0,    0,         1.0]
```

Código 4: Salida numérica de la cinemática directa DH modificada.

3.2. Cinemática inversa

Tanto para los parámetros clásicos como para los modificados de Denavit-Hartenberg las ecuaciones obtenidas a resolver son geométricamente iguales, y obtienen los mismos resultados, pero las ventajas de la convención modificada suelen facilitar su resolución. Por ello, se estudia la cinematica inversa sobre los parámetros modificados.

Cuando la muñeca es esférica, es decir, los ejes de las articulaciones 4, 5 y 6 se cortan en único punto o $a_4 = a_5 = a_6 = 0, d_5 = d_6 = 0$, se puede aplicar el método de desacoplo cinemático [12]. Sin embargo, esta propiedad no se cumple en el robot *UFactory xArm6*, ya que la posición y orientación depende de todas las articulaciones. Por tanto, no existe separación natural entre el brazo y la muñeca y la cinemática inversa se vuelve mucho más compleja porque no existe en forma cerrada.

Consecuentemente, este problema no puede resolverse analíticamente y tiene que ser resuelto con métodos numéricos iterativos como el inverso, pseudo-inverso y transpuesto del Jacobiano, Newton–Raphson o Levenberg–Marquardt, entre otros, utilizando las 12 ecuaciones de la matriz de transformación T_0^6. Estos métodos aproximan la solución mediante iteraciones sucesivas, minimizando el error entre la posición y orientación deseada y la alcanzada alcanzada por el robot.

El repositorio oficial del fabricante [8] utiliza *MoveIt 2* [20], que resuelve la cinemática inversa mediante Jacobiano pseudo-inverso y métodos iterativos de tipo Newton–Raphson.

De la matriz de transformación obtenida en el código 3, $T_0^6 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$, se obtienen las siguientes ecuaciones de posición y orientación:

$$
\begin{aligned}
p_x = {} & \cos(q_1)\cos(T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3)\left(76\cos q_4\cos q_5 - 97\cos q_4\sin q_5 + \frac{155}{2}\right) \\
& + a_2\cos(q_1)\cos(T2_{\text{offset}} + q_2) \\
& - \cos(q_1)\sin(T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3)\left(97\cos q_5 + 76\sin q_5 + \frac{685}{2}\right) \\
& + \sin(q_1)\sin(q_4)\left(76\cos q_5 - 97\sin q_5\right),
\end{aligned}
$$

$$
\begin{aligned}
p_y = {} & \sin(q_1)\cos(T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3)\left(76\cos q_4\cos q_5 - 97\cos q_4\sin q_5 + \frac{155}{2}\right) \\
& + a_2\sin(q_1)\cos(T2_{\text{offset}} + q_2) \\
& - \sin(q_1)\sin(T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3)\left(97\cos q_5 + 76\sin q_5 + \frac{685}{2}\right) \\
& - \cos(q_1)\sin(q_4)\left(76\cos q_5 - 97\sin q_5\right),
\end{aligned}
$$

$$
\begin{aligned}
p_z = 267 &- a_2 \sin(T2_{\text{offset}} + q_2) \\
&- \cos(T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3)\left(97\cos q_5 + 76\sin q_5 + \frac{685}{2}\right) \\
&- \sin(T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3)\left(76\cos q_4\cos q_5 - 97\cos q_4\sin q_5 + \frac{155}{2}\right).
\end{aligned}
$$

Sea $\phi = T2_{\text{offset}} + T3_{\text{offset}} + q_2 + q_3$, de modo que las expresiones de la matriz de rotación se simplifican.

$$
\begin{aligned}
r_{11} = &\sin q_1\,(\cos q_4\sin q_6 + \cos q_5\cos q_6\sin q_4) - \cos q_1\cos\phi\,(\sin q_4\sin q_6 - \cos q_4\cos q_5\cos q_6) \\
&- \cos q_1\sin\phi\,\cos q_6\sin q_5
\end{aligned}
$$

$$
\begin{aligned}
r_{12} = &\sin q_1\,(\cos q_4\cos q_6 - \cos q_5\sin q_4\sin q_6) - \cos q_1\cos\phi\,(\cos q_6\sin q_4 + \cos q_4\cos q_5\sin q_6) \\
&+ \cos q_1\sin\phi\,\sin q_5\sin q_6
\end{aligned}
$$

$$
r_{13} = -\sin q_1\sin q_4\sin q_5 - \cos q_1\sin\phi\,\cos q_5 - \cos q_1\cos\phi\,\cos q_4\sin q_5
$$

$$
\begin{aligned}
r_{21} = &-\cos q_1\,(\cos q_4\sin q_6 + \cos q_5\cos q_6\sin q_4) - \sin q_1\cos\phi\,(\sin q_4\sin q_6 - \cos q_4\cos q_5\cos q_6) \\
&- \sin q_1\sin\phi\,\cos q_6\sin q_5
\end{aligned}
$$

$$
\begin{aligned}
r_{22} = &\sin q_1\sin\phi\,\sin q_5\sin q_6 - \sin q_1\cos\phi\,(\cos q_6\sin q_4 + \cos q_4\cos q_5\sin q_6) \\
&- \cos q_1\,(\cos q_4\cos q_6 - \cos q_5\sin q_4\sin q_6)
\end{aligned}
$$

$$
r_{23} = \cos q_1\sin q_4\sin q_5 - \sin q_1\sin\phi\,\cos q_5 - \sin q_1\cos\phi\,\cos q_4\sin q_5
$$

$$
r_{31} = \sin\phi\,(\sin q_4\sin q_6 - \cos q_4\cos q_5\cos q_6) - \cos\phi\,\cos q_6\sin q_5
$$

$$
r_{32} = \sin\phi\,(\cos q_6\sin q_4 + \cos q_4\cos q_5\sin q_6) + \cos\phi\,\sin q_5\sin q_6
$$

$$
r_{33} = \sin\phi\,\cos q_4\sin q_5 - \cos\phi\,\cos q_5
$$

Estas ecuaciones se han obtenido con la ayuda de *Matlab* [2] y la *toolbox* de Peter Corke [17]:

```
clear; clc;
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

% --- Definición de eslabones DH MODIFICADO ---
L1 = Link([0     267    0      0        0  0], 'modified');
L2 = Link([0     0      0     -pi2      0  T2_offset], 'modified');
L3 = Link([0     0      a2     0        0  T3_offset], 'modified');
L4 = Link([0     342.5  77.5  -pi2      0  0], 'modified');
L5 = Link([0     0      0      pi2      0  0], 'modified');
L6 = Link([0     97     76    -pi2      0  0], 'modified');

% --- Matrices parciales ---
T01 = L1.A(q1).T;
T12 = L2.A(q2).T;
T23 = L3.A(q3).T;
T34 = L4.A(q4).T;
T45 = L5.A(q5).T;
T56 = L6.A(q6).T;

% --- Transformación completa ---
T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
R06 = T06(1:3,1:3);
p06 = T06(1:3,4);

% --- IMPRESIÓN DE ECUACIONES ---
disp('=== ECUACIONES DE POSICIÓN p06 ===');
disp('p_x ='); pretty(p06(1))
disp('p_y ='); pretty(p06(2))
disp('p_z ='); pretty(p06(3))

disp(' ');
disp('=== ECUACIONES DE ORIENTACIÓN R06 ===');
pretty(R06)
```

Código 5: Código Matlab para obtener las ecuaciones de posición y orientación con parámetros DH modificados.

3.2.1. Pseudoinversa del Jacobiano y Newton–Raphson

Como se expone en [12], la relación entre las velocidades articulares y la velocidad del efector final viene dada por

$$\dot{x} = J(q)\,\dot{q},$$

donde $J(q)$ es el Jacobiano geométrico del manipulador. Para invertir esta relación y obtener un incremento articular a partir de un desplazamiento cartesiano, se emplea la pseudoinversa del Jacobiano,

$$\dot{q} = J^{\dagger}(q)\,\dot{x}, \qquad J^{\dagger} = (J^{\top}J)^{-1}J^{\top},$$

válida cuando el Jacobiano es de rango completo. Este operador permite calcular un incremento articular que minimiza en norma cuadrática el error cartesiano, incluso en configuraciones próximas a singularidades o en robots redundantes.
Este mismo principio se aplica a la resolución numérica de la cinemática inversa. El objetivo consiste en encontrar un vector articular q tal que

$$f(q) = x_d,$$

siendo $f(q)$ la cinemática directa y x_d la posición y orientación deseadas del efector final. Definiendo el error cartesiano como

$$e = x_d - f(q),$$

y linealizando la cinemática directa alrededor de la configuración actual,

$$f(q + \Delta q) \approx f(q) + J(q)\,\Delta q,$$

se obtiene la actualización iterativa característica del método de Newton–Raphson [21]:

$$q_{k+1} = q_k + J^{\dagger}(q_k)\,(x_d - f(q_k)).$$

En esta expresión, $J^{\dagger}(q_k)$ actúa como una inversa generalizada del Jacobiano, permitiendo calcular un incremento articular coherente incluso cuando $J(q_k)$ no es cuadrado o se encuentra próximo a una singularidad. Este procedimiento converge de forma eficiente siempre que la estimación inicial sea adecuada, y constituye uno de los métodos numéricos más empleados para resolver la cinemática inversa en manipuladores sin solución analítica cerrada.

Ejemplo sencillo de Newton–Raphson a mano

Para ilustrar el método de Newton–Raphson, se considera un manipulador unidimensional con una sola articulación $q \in \mathbb{R}$, cuya cinemática directa viene dada por

$$f(q) = q^2.$$

El objetivo es encontrar el valor de q tal que la posición cartesiana del efector final coincida con una posición deseada x_d. En este ejemplo se fija

$$x_d = 4.$$

El problema de cinemática inversa se escribe entonces como

$$f(q) = x_d \quad \Longleftrightarrow \quad q^2 = 4.$$

Siguiendo la estructura anterior, se define el error cartesiano como

$$e(q) = x_d - f(q) = 4 - q^2.$$

En este caso, como se trata de un problema unidimensional, el Jacobiano geométrico se reduce a la derivada de $f(q)$ con respecto a q:

$$J(q) = \frac{\partial f}{\partial q} = 2q.$$

Para $J(q) \neq 0$, la pseudoinversa de un escalar coincide con su inversa

$$J^{\dagger}(q) = \frac{1}{J(q)} = \frac{1}{2q}.$$

Aplicando el esquema iterativo de Newton–Raphson para la cinemática inversa, se obtiene

$$q_{k+1} = q_k + J^{\dagger}(q_k)\left(x_d - f(q_k)\right).$$

Sustituyendo las expresiones de $f(q)$, x_d y $J^{\dagger}(q)$,

$$q_{k+1} = q_k + \frac{1}{2q_k}\left(4 - q_k^2\right).$$

Cálculo iterativo a mano

Se elige, por ejemplo, una estimación inicial

$$q_0 = 1.$$

A partir de esta estimación, se calcula paso a paso:

Iteración 1

$$f(q_0) = f(1) = 1^2 = 1,$$
$$x_d - f(q_0) = 4 - 1 = 3,$$
$$J(q_0) = 2q_0 = 2 \cdot 1 = 2, \qquad J^{\dagger}(q_0) = \frac{1}{2}.$$

Por tanto,

$$q_1 = q_0 + J^{\dagger}(q_0)\left(x_d - f(q_0)\right) = 1 + \frac{1}{2} \cdot 3 = 1 + 1{,}5 = 2{,}5.$$

Iteración 2

$$f(q_1) = f(2{,}5) = (2{,}5)^2 = 6{,}25,$$
$$x_d - f(q_1) = 4 - 6{,}25 = -2{,}25,$$
$$J(q_1) = 2q_1 = 2 \cdot 2{,}5 = 5, \qquad J^{\dagger}(q_1) = \frac{1}{5}.$$

Así,

$$q_2 = q_1 + J^{\dagger}(q_1)\left(x_d - f(q_1)\right) = 2{,}5 + \frac{1}{5} \cdot (-2{,}25) = 2{,}5 - 0{,}45 = 2{,}05.$$

Iteración 3

$$f(q_2) = f(2{,}05) = (2{,}05)^2 \approx 4{,}2025,$$
$$x_d - f(q_2) \approx 4 - 4{,}2025 = -0{,}2025,$$
$$J(q_2) = 2q_2 = 2 \cdot 2{,}05 = 4{,}10, \qquad J^\dagger(q_2) = \frac{1}{4{,}10}.$$

La siguiente actualización es

$$q_3 = q_2 + J^\dagger(q_2)\left(x_d - f(q_2)\right) \approx 2{,}05 + \frac{1}{4{,}10} \cdot (-0{,}2025) \approx 2{,}05 - 0{,}0494 \approx 2{,}0006.$$

Después de unas pocas iteraciones, el valor de q_k converge rápidamente hacia

$$q^\star \approx 2,$$

que es una de las soluciones exactas de la ecuación $q^2 = 4$. En este ejemplo tan simple, el Jacobiano es un escalar y su pseudoinversa coincide con la inversa, de modo que la actualización de Newton–Raphson se interpreta exactamente como en el caso general:

$$q_{k+1} = q_k + J^\dagger(q_k)\,(x_d - f(q_k)),$$

pero aplicada a una cinemática directa escalar y sencilla.

3.2.2. Cálculo de cinemática inversa para *UFactory xArm6*

De la tabla 1 se conoce que las articulaciones del robot presentan límites físicos de movimiento, por lo que el método iterativo de Newton–Raphson debe modificarse para garantizar que las soluciones obtenidas se mantengan dentro de dichos rangos. Este problema es bien conocido en robótica y suele abordarse mediante técnicas de *joint limit avoidance* [21, 22], que introducen un término adicional en la actualización de las articulaciones para alejar la solución de los límites articulares.

En lugar de aplicar un recorte directo de las articulaciones (*clamping*), que puede bloquear la convergencia cuando la solución se encuentra cerca de un límite, se define una función de coste que penaliza la proximidad a los extremos del rango articular:

$$H(q) = \sum_{i=1}^{6} \left(\frac{q_i - q_{i,\text{mid}}}{q_{i,\text{max}} - q_{i,\text{min}}} \right)^2,$$

donde $q_{i,\text{min}}$ y $q_{i,\text{max}}$ son los límites articulares y $q_{i,\text{mid}} = (q_{i,\text{min}} + q_{i,\text{max}})/2$ es el punto medio del rango. El gradiente de esta función,

$$\nabla H_i = 2\,\frac{q_i - q_{i,\text{mid}}}{(q_{i,\text{max}} - q_{i,\text{min}})^2},$$

indica la dirección en la que cada articulación debe desplazarse para alejarse de los límites. Este gradiente se incorpora directamente en la ecuación iterativa del método de Newton–Raphson, modificando la actualización clásica $\Delta q = J^+(q_k)\,e(q_k)$, para incluir un término de penalización:

$$\Delta q = J^+(q_k)\,e(q_k) \;-\; K\,\nabla H(q_k),$$

donde K es un coeficiente positivo que controla la intensidad de la penalización. Este enfoque permite que el algoritmo reduzca el error de posición garantizando que las posiciones articulares

obtenidas son factibles y se utiliza ampliamente en control redundante y cinemática inversa con restricciones [23].

Además, para mejorar la estabilidad numérica cerca de singularidades, se emplea la variante amortiguada del método, conocida como *Damped Least Squares* (DLS) o método de Levenberg–Marquardt [24, 22]. En este caso, la pseudoinversa se sustituye por:

$$\Delta q = J^\top (JJ^\top + \lambda^2 I)^{-1} e(q_k) \; - \; K \nabla H(q_k),$$

donde λ es un factor de amortiguación que evita que el Jacobiano se acerque a una singularidad o configuración geométricamente desfavorable.

La combinación de penalización suave y amortiguación permite obtener soluciones de cinemática inversa que respetan los límites articulares, evitan configuraciones cercanas a singularidades y mantienen la convergencia del método incluso en zonas estrechas del espacio de trabajo.

A continuación se expone la porción de código de *Matlab* [2] y la *Robotics Toolbox* de Peter Corke [17] con la que se ha programado la cinemática inversa para el *UFactory xArm6*:

```
clear; clc;
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

%% ================================================================
%   DEFINICIÓN DE ESLABONES (DH MODIFICADO)
% ================================================================
L1 = Link([0     267    0      0        0  0],       'modified');
L2 = Link([0     0      0     -pi2      0  T2_offset], 'modified');
L3 = Link([0     0      a2     0        0  T3_offset], 'modified');
L4 = Link([0     342.5  77.5  -pi2      0  0],       'modified');
L5 = Link([0     0      0      pi2      0  0],       'modified');
L6 = Link([0     97     76    -pi2      0  0],       'modified');

%% ================================================================
%   MATRICES DE TRANSFORMACIÓN PARCIALES
% ================================================================
T01 = L1.A(q1).T;
T12 = L2.A(q2).T;
T23 = L3.A(q3).T;
T34 = L4.A(q4).T;
T45 = L5.A(q5).T;
T56 = L6.A(q6).T;

%% ================================================================
%   TRANSFORMACIÓN COMPLETA T_0^6
% ================================================================
T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
p06 = T06(1:3,4);

%% ================================================================
%   JACOBIANO LINEAL (para cinemática inversa)
% ================================================================
Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);

%% ================================================================
%   CONVERTIR A FUNCIONES NUMÉRICAS
% ================================================================
p_fun = matlabFunction(p06, 'Vars', ...
    {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});

J_fun = matlabFunction(Jv, 'Vars', ...
    {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});

T_fun = matlabFunction(T06, 'Vars', ...
    {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});

%% ================================================================
%   PARÁMETROS NUMÉRICOS DEL ROBOT
```

```
% ================================================================
a2_val        = 289.48866;
T2_offset_val = deg2rad(-79.34995);
T3_offset_val = deg2rad(79.34995);


%% ================================================================
%   CINEMÁTICA INVERSA NUMÉRICA (-NEWTONRAPHSON, SOLO POSICIÓN)
% ================================================================
% Posición deseada (se sabe que corresponde a q_i = 0 con esos offsets)
p_d = [207; 0; 112];

% Configuración inicial
qk = zeros(6,1);   % punto de partida

tol     = 1e-6;
maxIter = 50;

% Límites matemáticos de las articulaciones (incluyendo offsets en 2 y 3)
% Rangos físicos (tabla):
% J1:  ±360º
% J2:  -117º -- 116º  (físicos)
% J3:  -219º -- 10º   (físicos)
% J4:  ±360º
% J5:  -97º -- 180º
% J6:  ±360º
%
% Convertimos a límites "matemáticos" para q2 y q3 (sin offset):
% q2_mat   [q2_fis_min - offset2, q2_fis_max - offset2]
% q3_mat   [q3_fis_min - offset3, q3_fis_max - offset3]
q1_min = -360;     q1_max =  360;
q2_min = -117 - rad2deg(T2_offset_val);   % grados matemáticos
q2_max =  116 - rad2deg(T2_offset_val);
q3_min = -219 - rad2deg(T3_offset_val);
q3_max =   10 - rad2deg(T3_offset_val);
q4_min = -360;     q4_max =  360;
q5_min = -97;      q5_max =  180;
q6_min = -360;     q6_max =  360;

q_min = deg2rad([q1_min, q2_min, q3_min, q4_min, q5_min, q6_min])';
q_max = deg2rad([q1_max, q2_max, q3_max, q4_max, q5_max, q6_max])';

% Centro del rango (para penalización suave)
q_mid = (q_min + q_max) / 2;

% Ganancia del término de penalización (joint limit avoidance)
K = 0.005;

for k = 1:maxIter

    % Evaluar posición y Jacobiano
    % (incluyen offsets vía a2_val, T2_offset_val, T3_offset_val)
    p_now = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), ...
                  a2_val, T2_offset_val, T3_offset_val);   % 3x1
    J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), ...
                  a2_val, T2_offset_val, T3_offset_val);   % 3x6

    % Error
    e = p_d - p_now;

    if norm(e) < tol
        break
    end

    % === GRADIENTE DE LA FUNCIÓN DE EVITACIÓN DE LÍMITES ===
    gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);

    % Paso -NewtonRaphson (pseudoinversa) + penalización suave
    dq = pinv(J_now) * e - K * gradH;

    % Actualizar
    qk = qk + dq;
end

% Redondear solución para limpiar ruido numérico
```

```
q_solution = round(qk, 12);

%% ================================================================
%   VERIFICACIÓN: EVALUAR T06(q*)
% ================================================================
T_check = T_fun(q_solution(1), q_solution(2), q_solution(3), ...
                q_solution(4), q_solution(5), q_solution(6), ...
                a2_val, T2_offset_val, T3_offset_val);
p_check = T_check(1:3,4);

%% ================================================================
%   MOSTRAR RESULTADOS
% ================================================================
disp('=== SOLUCIÓN DE CINEMÁTICA INVERSA (en radianes) ===')
disp(q_solution.')

disp('=== SOLUCIÓN DE CINEMÁTICA INVERSA (en grados) ===')
disp(rad2deg(q_solution).')

% Si se quieren ver los ángulos FÍSICOS de las articulaciones 2 y 3:
q_phys = q_solution;
q_phys(2) = q_phys(2) + T2_offset_val;
q_phys(3) = q_phys(3) + T3_offset_val;

disp('=== ÁNGULOS FÍSICOS (q + offset) EN GRADOS ===')
disp(rad2deg(q_phys).')

disp('=== POSICIÓN OBTENIDA ===')
disp(p_check)

disp('=== POSICIÓN DESEADA ===')
disp(p_d)

disp('=== ERROR VECTORIAL ===')
disp(e)

final_error = norm(e);
disp('=== ERROR FINAL ===')
disp(final_error)
```

Código 6: Cálculo de la cinemática inversa del robot *UFactory xArm6*.

```
=== SOLUCIÓN DE CINEMÁTICA INVERSA (en radianes) ===
         0   -0.0032   -0.0039         0    0.0154         0

=== SOLUCIÓN DE CINEMÁTICA INVERSA (en grados) ===
         0   -0.1856   -0.2235         0    0.8796         0

=== ÁNGULOS FÍSICOS (q + offset) EN GRADOS ===
         0  -79.5355   79.1264         0    0.8796         0

=== POSICIÓN OBTENIDA ===
  207.7231
         0
  112.1151

=== POSICIÓN DESEADA ===
   207
     0
   112

=== ERROR VECTORIAL ===
   -0.7231
    0.0000
   -0.1151

=== ERROR FINAL ===
    0.7322
```

Código 7: Salida del cálculo de la cinemática inversa del robot *UFactory xArm6*.

3.3. Cinemática diferencial y Jacobiano

Tal y como se expone en el apartado anterior, la cinemática diferencial describe la relación entre las velocidades articulares $\dot{q}$ y la velocidad del efector final $\dot{x}$. Esta relación viene dada por

$$\dot{x} = J(q)\,\dot{q},$$

donde $J(q)$ es el Jacobiano geométrico del manipulador. El Jacobiano recoge cómo las variaciones infinitesimales de las articulaciones se traducen en variaciones infinitesimales en el espacio cartesiano, y constituye la herramienta fundamental para analizar velocidades, singularidades y la cinemática inversa diferencial. Cuando se desea obtener un incremento articular a partir de un desplazamiento cartesiano, es necesario invertir esta relación. Sin embargo, el Jacobiano no siempre es cuadrado ni invertible, por lo que se recurre a una inversa generalizada, como la pseudoinversa, que permite obtener soluciones en mínimos cuadrados incluso en configuraciones próximas a singularidades.

A partir de la transformación homogénea total

$$T_0^6(q) = A_1(q_1)\; A_2(q_2 + T2_{\text{offset}})\; A_3(q_3 + T3_{\text{offset}})\; A_4(q_4)\; A_5(q_5)\; A_6(q_6),$$

se pueden calcular las matrices de rotación y vectores de posición necesarios para construir el Jacobiano y para calcular las velocidades lineales y angulares de las articulaciones y del efector. Dado que la matriz de transformación resultante es una expresión grande, su manejo manual simbólico resulta muy tedioso, por lo que se ha calculado con ayuda de *Matlab* [2] y la *Robotics Toolbox* de Peter Corke [17]:

```
clear; clc;
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

%% ================================================================
%   DEFINICIÓN DE ESLABONES (DH MODIFICADO)
% ================================================================
L1 = Link([0     267    0      0        0  0], 'modified');
L2 = Link([0     0      0     -pi2      0  T2_offset], 'modified');
L3 = Link([0     0      a2     0        0  T3_offset], 'modified');
L4 = Link([0     342.5  77.5  -pi2      0  0], 'modified');
L5 = Link([0     0      0      pi2      0  0], 'modified');
L6 = Link([0     97     76    -pi2      0  0], 'modified');

%% ================================================================
%   MATRICES DE TRANSFORMACIÓN PARCIALES
% ================================================================
T01 = L1.A(q1).T;
T12 = L2.A(q2).T;
T23 = L3.A(q3).T;
T34 = L4.A(q4).T;
T45 = L5.A(q5).T;
T56 = L6.A(q6).T;

%% ================================================================
%   TRANSFORMACIÓN COMPLETA T_0^6
% ================================================================
T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
R06 = T06(1:3,1:3);
p06 = T06(1:3,4);

%% ================================================================
%   IMPRESIÓN DE ECUACIONES SIMBÓLICAS
% ================================================================
disp('=== ECUACIONES DE POSICIÓN p06 ===');
disp('p_x ='); pretty(p06(1))
disp('p_y ='); pretty(p06(2))
disp('p_z ='); pretty(p06(3))
```

```
39
40 disp(' ');
41 disp('=== ECUACIONES DE ORIENTACIÓN R06 ===');
42 pretty(R06)
43
44 %% ================================================================
45 %   JACOBIANO GEOMÉTRICO (Jv + Jw)
46 % ================================================================
47 % Jacobiano lineal usando jacobian
48 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
49
50 % Ejes z_i en el marco base
51 z0 = [0;0;1];
52 T01_ = T01;
53 T02_ = T01*T12;
54 T03_ = T01*T12*T23;
55 T04_ = T01*T12*T23*T34;
56 T05_ = T01*T12*T23*T34*T45;
57
58 z1 = T01_(1:3,3);
59 z2 = T02_(1:3,3);
60 z3 = T03_(1:3,3);
61 z4 = T04_(1:3,3);
62 z5 = T05_(1:3,3);
63
64 % Jacobiano angular
65 Jw = [z0 z1 z2 z3 z4 z5];
66
67 % Jacobiano geométrico completo
68 J = simplify([Jv; Jw]);
69
70 disp('=== JACOBIANO GEOMÉTRICO ===');
71 pretty(J)
72
73 %% ================================================================
74 %   VELOCIDADES SIMBÓLICAS
75 % ================================================================
76 syms dq1 dq2 dq3 dq4 dq5 dq6 real
77 dq = [dq1; dq2; dq3; dq4; dq5; dq6];
78
79 % Contribución angular de cada articulación
80 omega1 = simplify(Jw(:,1) * dq1);
81 omega2 = simplify(Jw(:,2) * dq2);
82 omega3 = simplify(Jw(:,3) * dq3);
83 omega4 = simplify(Jw(:,4) * dq4);
84 omega5 = simplify(Jw(:,5) * dq5);
85 omega6 = simplify(Jw(:,6) * dq6);
86
87 % Velocidades del efector
88 vel = simplify(J * dq);
89 v_linear  = vel(1:3);
90 v_angular = vel(4:6);
91
92 disp('=== VELOCIDAD ANGULAR APORTADA POR CADA ARTICULACIÓN ===');
93 disp('omega_1 ='); pretty(omega1)
94 disp(' '); disp('omega_2 ='); pretty(omega2)
95 disp(' '); disp('omega_3 ='); pretty(omega3)
96 disp(' '); disp('omega_4 ='); pretty(omega4)
97 disp(' '); disp('omega_5 ='); pretty(omega5)
98 disp(' '); disp('omega_6 ='); pretty(omega6)
99
100 disp(' ');
101 disp('=== VELOCIDAD LINEAL DEL EFECTOR ===');
102 pretty(v_linear)
103
104 disp(' ');
105 disp('=== VELOCIDAD ANGULAR DEL EFECTOR ===');
106 pretty(v_angular)
107
108 %% ================================================================
109 %   EVALUACIÓN NUMÉRICA DE T06, J Y VELOCIDADES
110 % ================================================================
111 disp(' ');
112 disp('================================================================');
113 disp('===   EVALUACIÓN NUMÉRICA DE T06, J Y VELOCIDADES          ===');
```

```
disp('================================================================');

% Valores numéricos
S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
S.a2 = 289.48866;
S.T2_offset = deg2rad(-79.34995);
S.T3_offset = deg2rad(79.34995);

% Imprimir valores usados
disp(' ');
disp('=== VALORES USADOS PARA LA EVALUACIÓN NUMÉRICA ===');
fprintf('q1 = %.4f rad\n', S.q1);
fprintf('q2 = %.4f rad\n', S.q2);
fprintf('q3 = %.4f rad\n', S.q3);
fprintf('q4 = %.4f rad\n', S.q4);
fprintf('q5 = %.4f rad\n', S.q5);
fprintf('q6 = %.4f rad\n', S.q6);
fprintf('a2 = %.5f mm\n', S.a2);
fprintf('T2_offset = %.4f rad (%.2f°)\n', S.T2_offset, rad2deg(S.T2_offset));
fprintf('T3_offset = %.4f rad (%.2f°)\n', S.T3_offset, rad2deg(S.T3_offset));

% Transformación numérica
T06_num = vpa(subs(T06, S), 10);
disp(' ');
disp('=== TRANSFORMACIÓN HOMOGÉNEA T_0^6 NUMÉRICA ===');
disp(T06_num);

% Jacobiano numérico
J_num = vpa(subs(J, S), 10);
disp(' ');
disp('=== JACOBIANO GEOMÉTRICO NUMÉRICO ===');
disp(J_num);

% Velocidades articulares numéricas
dq_val = [0.2; 0.1; -0.15; 0.05; 0.1; -0.05];
vel_num = double(J_num * dq_val);

v_linear_num  = vel_num(1:3);
v_angular_num = vel_num(4:6);

disp(' ');
disp('=== VELOCIDAD LINEAL DEL EFECTOR (numérica) ===');
disp(v_linear_num.');

disp(' ');
disp('=== VELOCIDAD ANGULAR DEL EFECTOR (numérica) ===');
disp(v_angular_num.');
```

Código 8: Cálculo de la cinemática diferencial y Jacobiano mediante parámetros DH modificados.

3.4. Generación de trayectorias

La generación de trayectorias se implementa mediante el planificador de *MoveIt 2* y el paquete *xarm_planner*, integrados en el repositorio oficial [8]. Este planificador permite definir trayectorias en espacio articular y cartesiano, que son posteriormente ejecutadas en tiempo real a través de *ros2_control* [25].

Los tipos de trayectorias que se pueden generar son [12]: lineales y circulares en espacio cartesiano, polinómicas en espacio articular y multipunto (*multi-waypoint*).

3.4.1. Trayectorias lineales en espacio cartesiano

el efector final se desplaza siguiendo una línea recta entre dos poses definidas. Matemáticamente,

$$x(t) = x_0 + \frac{t}{T}(x_f - x_0), \qquad t \in [0, T],$$

con condiciones de contorno

$$x(0) = x_0, \qquad x(T) = x_f.$$

Aunque la interpolación cartesiana es lineal, *MoveIt* aplica una parametrización temporal que suaviza la velocidad y aceleración, evitando discontinuidades. En particular, se emplea el algoritmo *Iterative Parabolic Time Parameterization (IPTP)*, que ajusta los perfiles de velocidad y aceleración de cada articulación para cumplir los límites dinámicos del robot.

3.4.2. Trayectorias circulares en espacio cartesiano

se definen mediante un arco de circunferencia, especificando centro c, radio r y ángulo barrido $\theta(t)$:

$$x(t) = c + r \begin{bmatrix} \cos\theta(t) \\ \sin\theta(t) \\ 0 \end{bmatrix}, \qquad \theta(t) = \theta_0 + \frac{t}{T}(\theta_f - \theta_0).$$

Estas trayectorias se utilizan para movimientos de arco, por ejemplo en operaciones de pulido o soldadura. En este caso, también pueden aparecer discontinuidades en la velocidad y aceleración, siendo necesario aplicar el algoritmo *IPTP*.

3.4.3. Corrección de discontinuidades mediante *IPTP*

El procedimiento consiste en recorrer iterativamente la trayectoria planificada y asignar tiempos de paso a cada segmento de manera que:

$$\dot{q}_i(t) \leq \dot{q}_{i,\max}, \qquad \ddot{q}_i(t) \leq \ddot{q}_{i,\max},$$

para todas las articulaciones i, garantizando que las velocidades y aceleraciones nunca superen los valores máximos permitidos.

El algoritmo *IPTP* utiliza perfiles de velocidad parabólicos para cada segmento, de modo que la aceleración se mantiene constante dentro de cada tramo y se ajusta en los puntos de unión para asegurar continuidad. Esto permite que las trayectorias lineales o circulares en espacio

cartesiano se conviertan en trayectorias articulares suaves y físicamente realizables.

Para ilustrar el problema, supóngase una trayectoria lineal cartesiana definida por dos segmentos consecutivos en el espacio articular de una sola articulación $q(t)$:

$$q(t) = \begin{cases} q_0 + \dfrac{t}{T_1}(q_1 - q_0), & t \in [0, T_1], \\ q_1 + \dfrac{t - T_1}{T_2}(q_2 - q_1), & t \in [T_1, T_1 + T_2]. \end{cases}$$

Las condiciones de contorno son:

$$q(0) = q_0, \quad q(T_1) = q_1, \quad q(T_1 + T_2) = q_2.$$

La velocidad resultante en cada tramo es constante:

$$\dot{q}(t) = \begin{cases} \dfrac{q_1 - q_0}{T_1}, & t \in [0, T_1], \\ \dfrac{q_2 - q_1}{T_2}, & t \in [T_1, T_1 + T_2]. \end{cases}$$

Esto implica una discontinuidad en la velocidad en el punto de unión $t = T_1$, ya que:

$$\dot{q}(T_1^-) = \frac{q_1 - q_0}{T_1} \quad \neq \quad \dot{q}(T_1^+) = \frac{q_2 - q_1}{T_2}.$$

Si los valores difieren, se produce un salto brusco en la velocidad, y por tanto una aceleración infinita en ese instante:

$$\ddot{q}(t) = \delta(t - T_1)\left(\dot{q}(T_1^+) - \dot{q}(T_1^-)\right),$$

donde $\delta(t - T_1)$ representa la función delta de Dirac, que permite modelar esa variación brusca.

El algoritmo *Iterative Parabolic Time Parameterization (IPTP)* corrige este problema introduciendo perfiles de velocidad parabólicos en cada segmento. En lugar de mantener velocidad constante, se define un perfil con aceleración constante en la fase inicial y final de cada tramo y velocidad constante en la fase intermedia:

$$\dot{q}(t) = \begin{cases} \dot{q}_0 + \ddot{q}\, t, & \text{fase de aceleración}, \\ \dot{q}_{\text{const}}, & \text{fase de velocidad constante}, \\ \dot{q}_{\text{const}} - \ddot{q}\,(t - t_f), & \text{fase de deceleración}. \end{cases}$$

De este modo, la aceleración $\ddot{q}$ se mantiene finita y constante en cada tramo, y la velocidad se ajusta suavemente en los puntos de unión, garantizando continuidad:

$$\dot{q}(T_1^-) = \dot{q}(T_1^+).$$

3.4.4. Trayectorias polinómicas en espacio articular

Cuando se especifican directamente configuraciones articulares, *MoveIt* genera trayectorias suaves mediante interpolación polinómica que garantizan continuidad en posición y velocidad evitando saltos bruscos. Sin embargo, con el objetivo de asegurar que la trayectoria sea

físicamente realizable y que se respeten los límites dinámicos del robot, es necesario realizar una parametrización temporal mediante el algoritmo *Iterative Parabolic Time Parameterization (IPTP)*.

Las trayectorias polinómicas más habituales son el polinomio cúbico:

$$q_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, \qquad t \in [0, T].$$

con condiciones de contorno en posición y velocidad:

$$q_i(0) = q_{i,0}, \quad \dot{q}_i(0) = \dot{q}_{i,0}, \quad q_i(T) = q_{i,f}, \quad \dot{q}_i(T) = \dot{q}_{i,f}.$$

$$a_0 = q_{i,0}, \quad a_1 = \dot{q}_{i,0},$$
$$a_2 = \frac{3\,(q_{i,f} - q_{i,0}) - (2\,\dot{q}_{i,0} + \dot{q}_{i,f})\,T}{T^2}, \quad a_3 = \frac{2\,(q_{i,0} - q_{i,f}) + (\dot{q}_{i,0} + \dot{q}_{i,f})\,T}{T^3}.$$

Y el polinomio quíntico:

$$q_i(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5, \qquad t \in [0, T].$$

$$q_i(0) = q_{i,0}, \quad \dot{q}_i(0) = \dot{q}_{i,0}, \quad \ddot{q}_i(0) = \ddot{q}_{i,0},$$
$$q_i(T) = q_{i,f}, \quad \dot{q}_i(T) = \dot{q}_{i,f}, \quad \ddot{q}_i(T) = \ddot{q}_{i,f}.$$

$$\begin{aligned}
a_0 &= q_{i,0},\\
a_1 &= \dot{q}_{i,0},\\
a_2 &= \tfrac{1}{2}\,\ddot{q}_{i,0},\\
a_3 &= \frac{20(q_{i,f} - q_{i,0}) - (8\dot{q}_{i,f} + 12\dot{q}_{i,0})T - (3\ddot{q}_{i,0} - \ddot{q}_{i,f})T^2}{2T^3},\\
a_4 &= \frac{30(q_{i,0} - q_{i,f}) + (14\dot{q}_{i,f} + 16\dot{q}_{i,0})T + (3\ddot{q}_{i,0} - 2\ddot{q}_{i,f})T^2}{2T^4},\\
a_5 &= \frac{12(q_{i,f} - q_{i,0}) - (6\dot{q}_{i,f} + 6\dot{q}_{i,0})T - (\ddot{q}_{i,0} - \ddot{q}_{i,f})T^2}{2T^5}.
\end{aligned}$$

3.4.5. Trayectorias *spline* (*multi-waypoint*)

Cuando se definen múltiples puntos intermedios en espacio cartesiano, *MoveIt* interpola mediante *splines* cúbicos, asegurando continuidad en posición y velocidad a lo largo de toda la trayectoria. Sin embargo, esta interpolación geométrica no garantiza por sí sola que se respeten los límites dinámicos del robot. Por ello, el algoritmo de parametrización temporal *Iterative Spline Parameterization (ISP)* ajusta los tiempos de paso de cada segmento para evitar oscilaciones y asegurar un movimiento fluido y físicamente realizable.

Un *spline* cúbico se define en cada intervalo $[t_k, t_{k+1}]$ como:

$$q_i(t) = b_0 + b_1(t - t_k) + b_2(t - t_k)^2 + b_3(t - t_k)^3,$$

donde los coeficientes b_j se calculan imponiendo continuidad en posición, velocidad y aceleración en los puntos de unión:

$$q_i(t_k^-) = q_i(t_k^+), \quad \dot{q}_i(t_k^-) = \dot{q}_i(t_k^+), \quad \ddot{q}_i(t_k^-) = \ddot{q}_i(t_k^+).$$

$$q_i(t_k) = q_{i,k}, \quad \dot{q}_i(t_k) = v_{i,k}, \quad q_i(t_{k+1}) = q_{i,k+1}, \quad \dot{q}_i(t_{k+1}) = v_{i,k+1}.$$

$$\begin{aligned} b_0 &= q_{i,k}, \\ b_1 &= v_{i,k}, \\ b_2 &= \frac{3(q_{i,k+1} - q_{i,k})}{h^2} - \frac{2v_{i,k} + v_{i,k+1}}{h}, \\ b_3 &= \frac{2(q_{i,k} - q_{i,k+1})}{h^3} + \frac{v_{i,k} + v_{i,k+1}}{h^2}, \end{aligned}$$

donde $h = t_{k+1} - t_k$ es la duración del intervalo. De este modo, la trayectoria completa es continua en posición, velocidad y aceleración, evitando saltos bruscos y oscilaciones no deseadas. El algoritmo *ISP* recorre iterativamente todos los segmentos y ajusta los tiempos de paso para cumplir los límites dinámicos del robot:

$$\dot{q}_i(t) \leq \dot{q}_{i,\max}, \qquad \ddot{q}_i(t) \leq \ddot{q}_{i,\max}.$$

Cabe destacar que los coeficientes b_j en cada intervalo se obtienen a partir de las condiciones de posición y velocidad en los extremos, mientras que la continuidad en aceleración se garantiza al resolver el sistema *spline* completo para todos los *waypoints* o puntos de paso.

3.5. Cálculo de la trayectoria en *Matlab* con la *toolbox* de Peter Corke

Se ha programado una trayectoria articular quíntica para el robot en estudio, garantizando que las posiciones obtenidas en durante la trayectoria son realizables. Es importante tener en cuenta que previo al cálculo de esta trayectoria es necesario obtener las posiciones articulares finales deseadas mediante cinemática inversa.

Las imágenes 9 y 10 muestran la trayectoria calculada mediante el código 9 para las posiciones indicadas en el código 10.

```
clear; clc; close all;
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

%% ================================
%  Definición de eslabones (DH modificado)
% ================================
L1 = Link([0     267    0      0        0  0], 'modified');
L2 = Link([0     0      0     -pi2      0  T2_offset], 'modified');
L3 = Link([0     0      a2     0        0  T3_offset], 'modified');
L4 = Link([0     342.5  77.5  -pi2      0  0], 'modified');
L5 = Link([0     0      0      pi2      0  0], 'modified');
L6 = Link([0     97     76    -pi2      0  0], 'modified');

%% ================================
%  Transformación completa T0^6
% ================================
T01 = L1.A(q1).T; T12 = L2.A(q2).T; T23 = L3.A(q3).T;
T34 = L4.A(q4).T; T45 = L5.A(q5).T; T56 = L6.A(q6).T;
T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
```

```
p06 = T06(1:3,4);

% Funciones numéricas
T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});

%% ================================
%  Parámetros numéricos del robot
% ================================
a2_val        = 289.48866;
T2_offset_val = deg2rad(-79.34995);
T3_offset_val = deg2rad(79.34995);

% Límites articulares (matemáticos, en grados)
q1_min = -360;     q1_max =  360;
q2_min = -117 - rad2deg(T2_offset_val);
q2_max =  116 - rad2deg(T2_offset_val);
q3_min = -219 - rad2deg(T3_offset_val);
q3_max =   10 - rad2deg(T3_offset_val);
q4_min = -360;     q4_max =  360;
q5_min = -97;      q5_max =  180;
q6_min = -360;     q6_max =  360;

% Convertir a radianes
q_min = deg2rad([q1_min q2_min q3_min q4_min q5_min q6_min]);
q_max = deg2rad([q1_max q2_max q3_max q4_max q5_max q6_max]);

%% ================================
%  Trayectoria articular (quintic)
% ================================
q0 = zeros(1,6); % inicio en 0
% finales aleatorios dentro de rango
qf = q_min + rand(1,6).*(q_max - q_min);

T = 5; % duración nominal inicial
n = 200; t = linspace(0,T,n);

% Polinomio quíntico (vel y acc inicial/final = 0)
Q = zeros(n,6);
for i=1:6
    a0 = q0(i); a1 = 0; a2 = 0;
    a3 = 10*(qf(i)-q0(i))/T^3;
    a4 = -15*(qf(i)-q0(i))/T^4;
    a5 = 6*(qf(i)-q0(i))/T^5;
    Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
end

%% ================================
%  IPTP: ajuste temporal
% ================================
% Límites dinámicos (ejemplo, ajusta según datasheet)
vel_max = deg2rad([180 180 180 180 180 180]);   % rad/s
acc_max = deg2rad([1500 1500 1500 1500 1500 1500]);   % rad/s^2

dt = t(2)-t(1);
dQ = diff(Q)/dt;          % velocidades
ddQ = diff(dQ)/dt;        % aceleraciones

% Escalado temporal si excede límites
scale_v = max(max(abs(dQ)./vel_max));
scale_a = max(max(abs(ddQ)./acc_max));
scale = max([1, scale_v, sqrt(scale_a)]);

if scale > 1
    T = T*scale; % estira el tiempo total
    t = linspace(0,T,n);
    for i=1:6
        a0 = q0(i); a1 = 0; a2 = 0;
        a3 = 10*(qf(i)-q0(i))/T^3;
        a4 = -15*(qf(i)-q0(i))/T^4;
        a5 = 6*(qf(i)-q0(i))/T^5;
        Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
    end
end

%% ================================
```

```
%  Visualización de articulaciones
% ================================
figure;
plot(t, rad2deg(Q));
xlabel('Tiempo [s]'); ylabel('Ángulo [°]');
legend('q1','q2','q3','q4','q5','q6');
title('Trayectoria articular (quíntica con IPTP)');

%% ================================
%  Trayectoria del efector final
% ================================
P = zeros(n,3);
for k=1:n
    Tnow = T_fun(Q(k,1),Q(k,2),Q(k,3),Q(k,4),Q(k,5),Q(k,6), ...
                 a2_val,T2_offset_val,T3_offset_val);
    P(k,:) = Tnow(1:3,4).';
end

figure;
plot3(P(:,1),P(:,2),P(:,3),'b-','LineWidth',2);
grid on; xlabel('X'); ylabel('Y'); zlabel('Z');
title('Trayectoria del efector final (quíntica con IPTP)');

%% ================================
%  Imprimir posiciones inicial y final
% ================================
% Valores físicos (sumando offset en q2 y q3)
q0_phys = q0;
q0_phys(2) = q0_phys(2) + T2_offset_val;
q0_phys(3) = q0_phys(3) + T3_offset_val;

qf_phys = qf;
qf_phys(2) = qf_phys(2) + T2_offset_val;
qf_phys(3) = qf_phys(3) + T3_offset_val;

disp('=== Valores iniciales físicos (grados) ===');
disp(rad2deg(q0_phys));

disp('=== Valores finales físicos (grados) ===');
disp(rad2deg(qf_phys));
```

Código 9: Generación de trayectoria articular realizable.

```
=== Valores iniciales físicos (grados) ===
         0  -79.3500   79.3500         0         0         0

=== Valores finales físicos (grados) ===
   33.9952  -84.7005 -184.8117 -174.5941  135.8787 -176.9168
```

Código 10: Salida de la generación de trayectoria articular realizable.

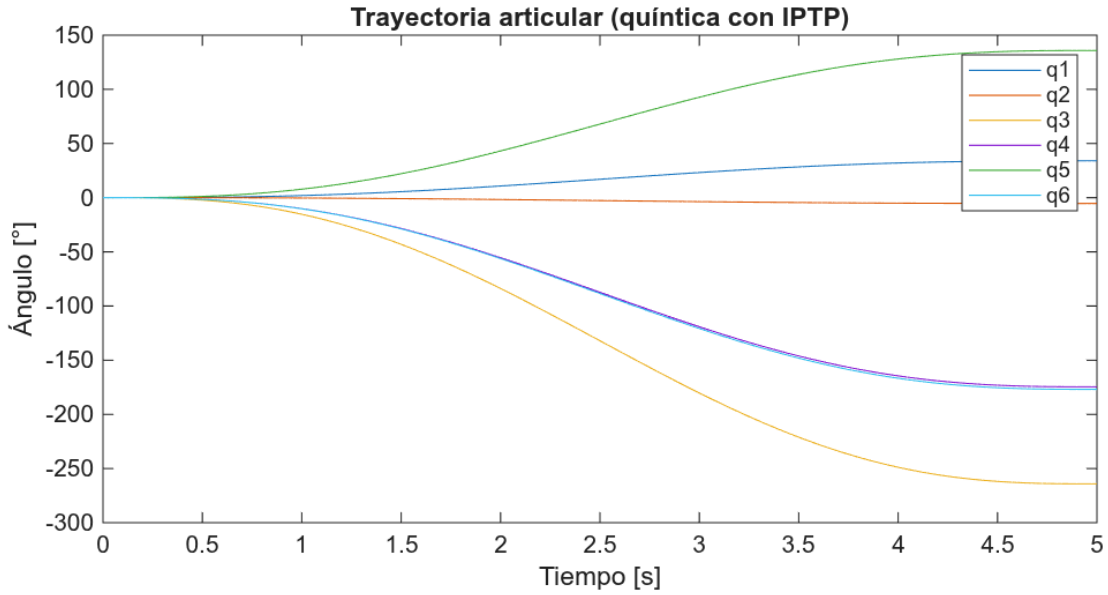


Figura 9: Trayectoria articular realizable.

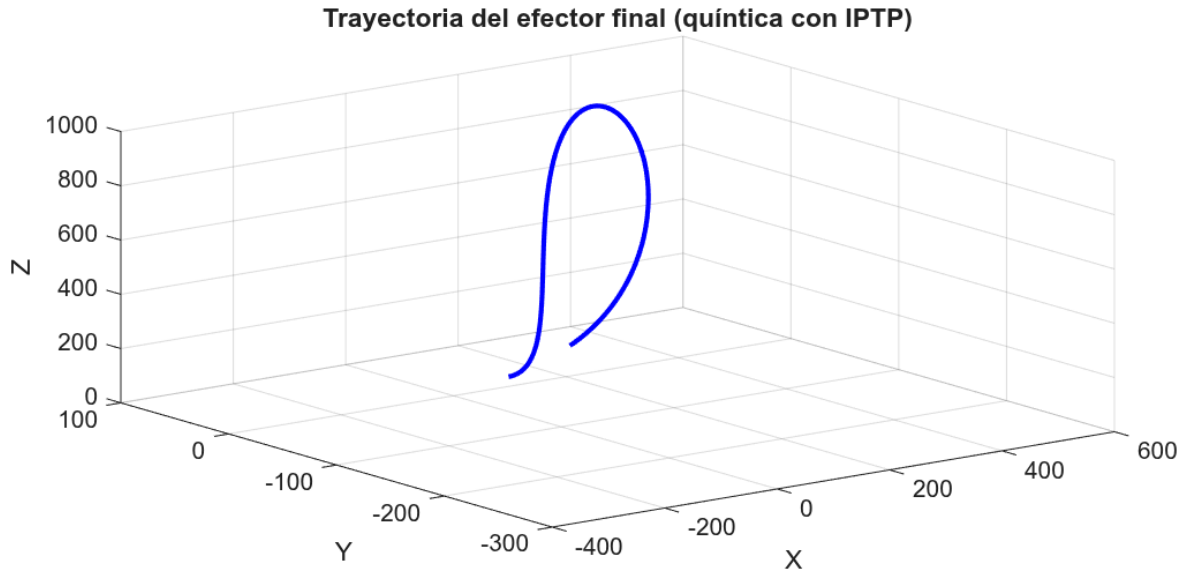


Figura 10: Trayectoria realizable del efector.

3.6. Control cinemático

Para llevar a cabo el control cinemático del brazo robótico es necesario combinar todo el estudio cinemático presentado, tal que los pasos a seguir son:

1. Calcular la cinemática directa.
2. Calcular la cinemática diferencial.
3. Calcular la cinemática inversa.
4. Generar las trayectorias de movimiento.

En este sentido, se ha programado en *Matlab*, junto con la *toolbox* de Peter Corke, el control cinemático completo del robot *UFactory xArm6*. Al inicio, el programa solicita la posición articular de origen, dado que el robot conoce con exactitud las posiciones de sus articulaciones; la posición cartesiana a alcanzar, ya que es de interés del usuario mover el robot a una posición cartesiana, y la duración en segundos de la trayectoria. La aplicación comprueba que la posición final es alcanzable según el rango de trabajo del robot de 700 mm, calcula las posiciones, velocidades y aceleraciones articulares necesarias para alcanzar la posición cartesiana final mediante cinemática inversa y genera la trayectoria correspondiente.

Las imágenes 11, 12, 13, 14, 15 y 16 muestran la trayectoria articular y del efector y las velocidades y aceleraciones articulares y cartesianas, y los códigos 11 y 12 la implementación y la salida por terminal.

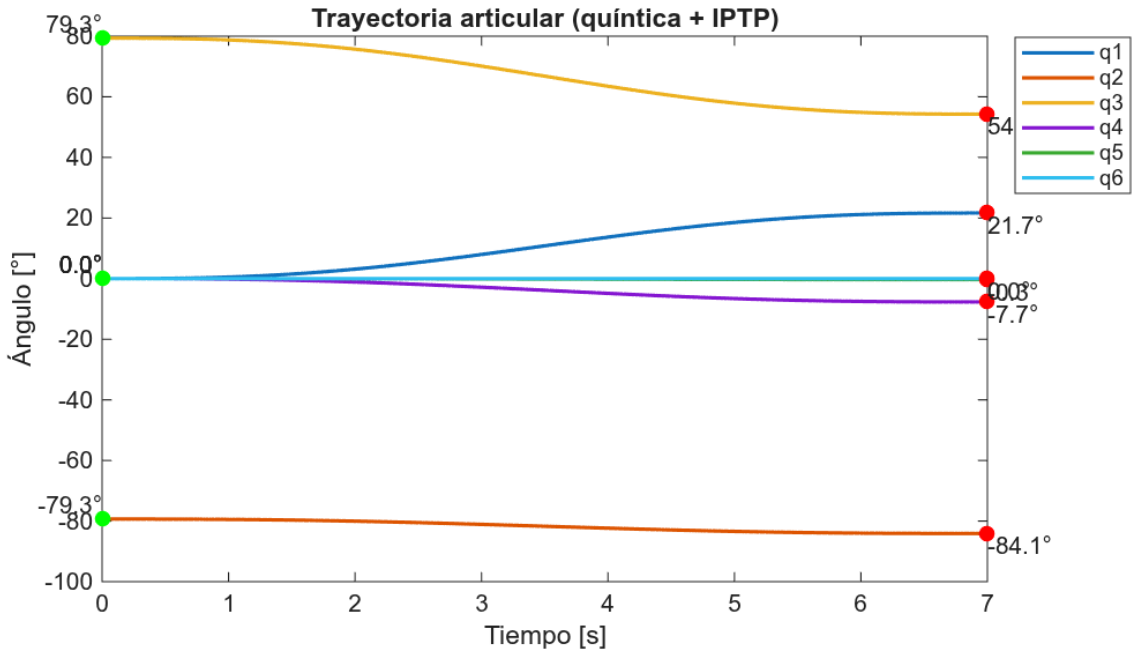


Figura 11: Trayectoria articular realizable.

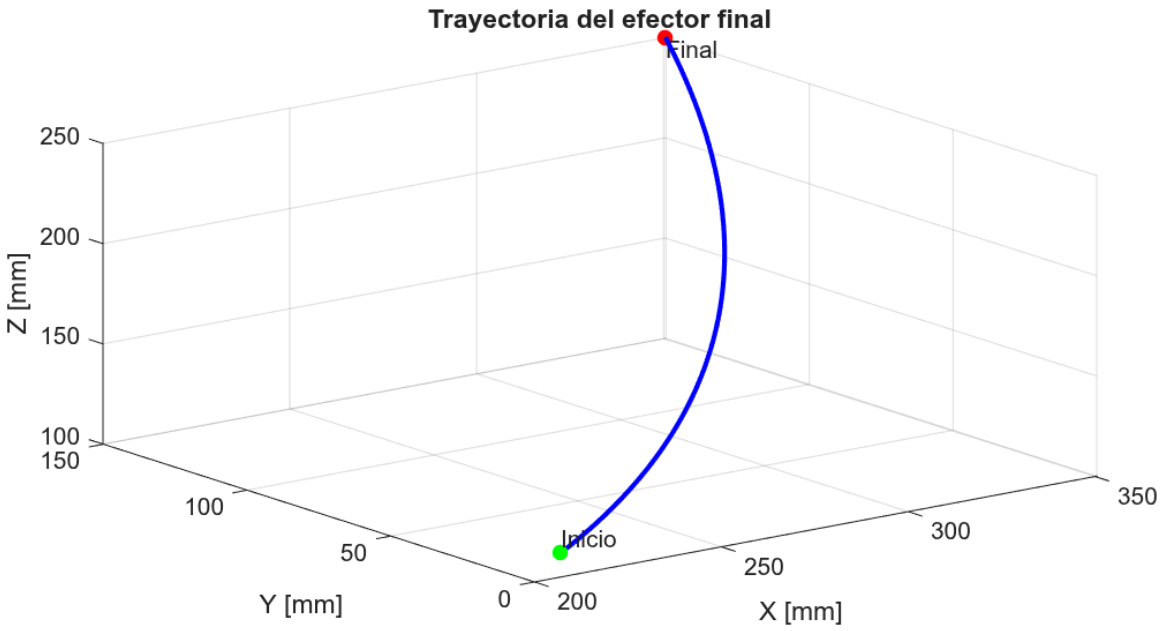


Figura 12: Trayectoria realizable del efector.

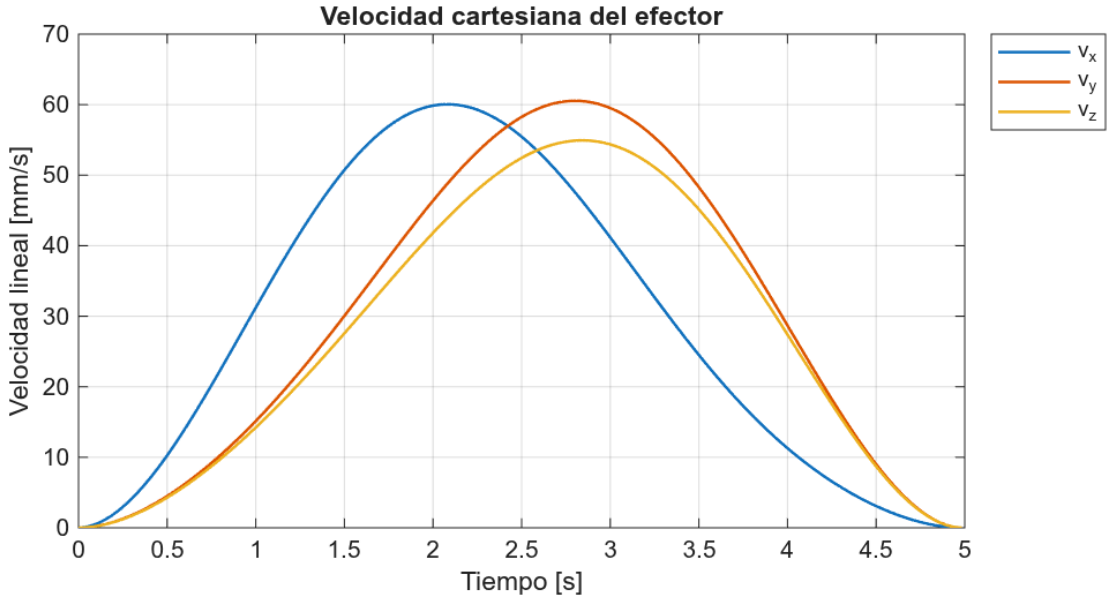


Figura 13: Velocidad cartesiana del efector.

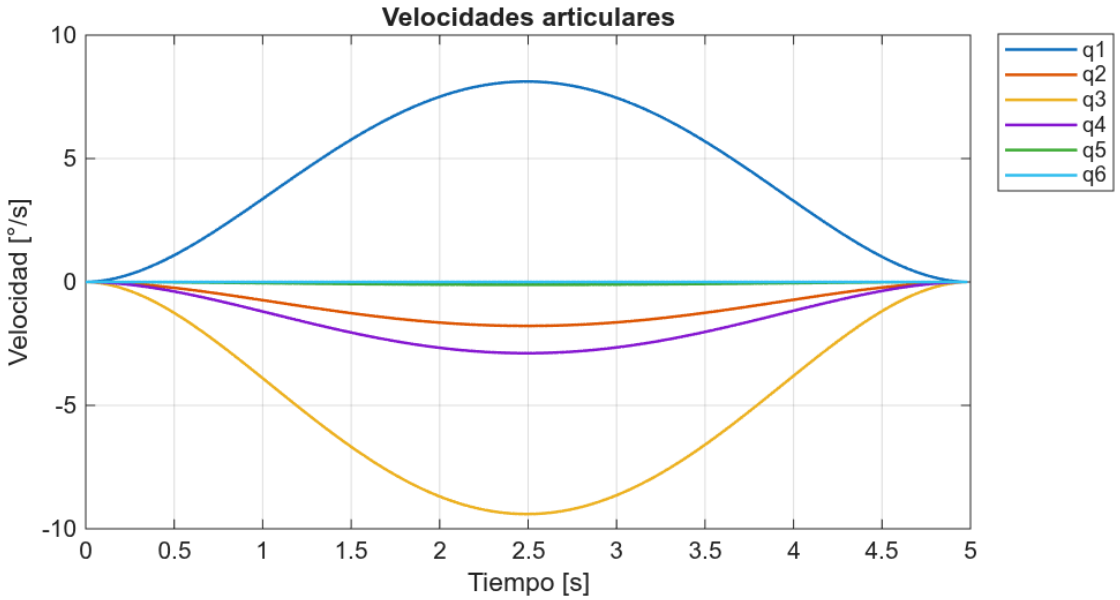


Figura 14: Velocidades articulares.

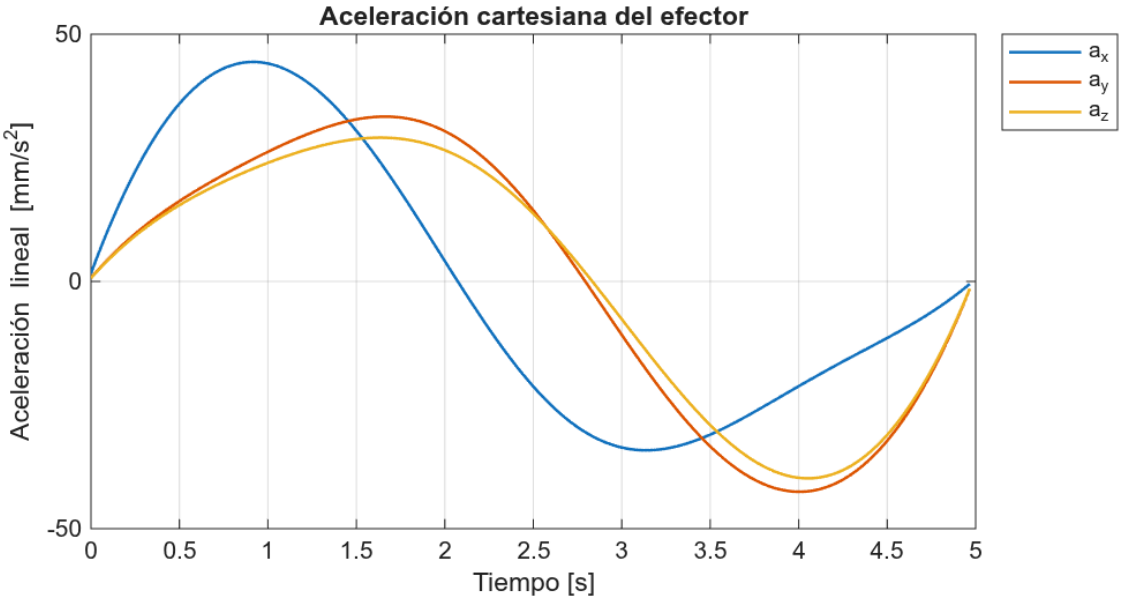


Figura 15: Aceleración cartesiana del efector.

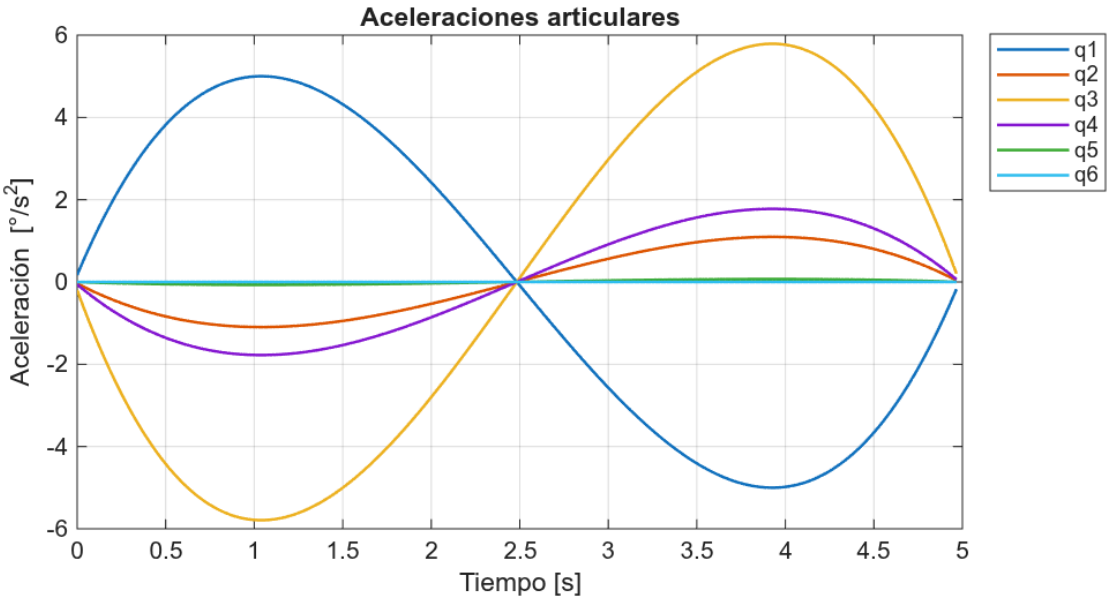


Figura 16: Aceleraciones articulares.

```m
clear; clc; close all;
syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
pi2 = sym(pi)/2;

%% ================================
%  Modelo xArm6 (DH modificado)
% ================================
L1 = Link([0     267    0      0        0  0], 'modified');
L2 = Link([0     0      0     -pi2      0  T2_offset], 'modified');
L3 = Link([0     0      a2     0        0  T3_offset], 'modified');
L4 = Link([0     342.5  77.5  -pi2      0  0], 'modified');
L5 = Link([0     0      0      pi2      0  0], 'modified');
L6 = Link([0     97     76    -pi2      0  0], 'modified');

%% ================================
%  Transformaciones y Jacobiano
% ================================
T01 = L1.A(q1).T; T12 = L2.A(q2).T; T23 = L3.A(q3).T;
T34 = L4.A(q4).T; T45 = L5.A(q5).T; T56 = L6.A(q6).T;
T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
p06 = T06(1:3,4);
Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);

% Funciones numéricas
p_fun = matlabFunction(p06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
J_fun = matlabFunction(Jv, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});

%% ================================
%  Parámetros del robot
% ================================
a2_val        = 289.48866;
T2_offset_val = deg2rad(-79.34995);
T3_offset_val = deg2rad(79.34995);

% Límites físicos → matemáticos (deg)
q1_min = -360;     q1_max =  360;
q2_min = -117 - rad2deg(T2_offset_val);
q2_max =  116 - rad2deg(T2_offset_val);
q3_min = -219 - rad2deg(T3_offset_val);
q3_max =   10 - rad2deg(T3_offset_val);
q4_min = -360;     q4_max =  360;
q5_min = -97;      q5_max =  180;
q6_min = -360;     q6_max =  360;

q_min = deg2rad([q1_min, q2_min, q3_min, q4_min, q5_min, q6_min])';
q_max = deg2rad([q1_max, q2_max, q3_max, q4_max, q5_max, q6_max])';
```

```
48
49 %% ================================
50 %  Entrada: articulaciones actuales, pose final y duración
51 % ================================
52 dlg = inputdlg( ...
53     {'Articulaciones actuales [q1 q2 q3 q4 q5 q6] en grados (matemático):', ...
54      'Posición final del efector [x y z] en mm:', ...
55      'Duración nominal de la trayectoria [s]:'}, ...
56     'Entrada de movimiento', [1 70], {'0 0 0 0 0 0', '350 150 250', '5'});
57
58 if isempty(dlg)
59     q0_deg = input('Introduce [q1 q2 q3 q4 q5 q6] en grados (matemático): ');
60     p_goal = input('Introduce [x y z] en mm para la posición final del efector: ');
61     T      = input('Introduce la duración nominal de la trayectoria [s]: ');
62 else
63     q0_deg = str2num(dlg{1});
64     p_goal = str2num(dlg{2});
65     T      = str2double(dlg{3});
66 end
67
68 assert(numel(q0_deg)==6, 'Debes introducir 6 valores articulares.');
69 assert(numel(p_goal)==3, 'Debes introducir 3 valores cartesianas [x y z].');
70
71 % Convertir a radianes y columna
72 q_init = deg2rad(q0_deg(:));
73 p_goal = p_goal(:);
74
75 % Pose inicial cartesiana derivada de las articulaciones actuales
76 T_init = T_fun(q_init(1), q_init(2), q_init(3), q_init(4), q_init(5), q_init(6), ...
77                a2_val, T2_offset_val, T3_offset_val);
78 p_init = T_init(1:3,4);
79
80 %% ================================
81 %  Verificación de alcanzabilidad (radio de acción)
82 % ================================
83 reach_max = 700; % mm (xArm6)
84 d_goal = norm(p_goal);
85
86 if d_goal > reach_max
87     fprintf('Objetivo a %.1f mm, fuera del alcance máximo de %.1f mm -> NO realizable\n', ...
88         d_goal, reach_max);
89     return;   % detener el programa aquí
90 end
91
92 fprintf('Objetivo a %.1f mm, dentro del alcance máximo %.1f mm -> Realizable\n', ...
93     d_goal, reach_max);
94
95 %% ================================
96 %  IK numérica (solo posición) con penalización de límites
97 %  (SIN clamping: se mantiene tu diseño original)
98 % ================================
99 tol = 1e-6; maxIter = 80; K = 0.005; % penalización
100 q_mid = (q_min + q_max)/2;
101
102 % Semilla: las articulaciones actuales
103 q_goal_seed = q_init;
104 q_goal = newtonIK(p_goal, q_goal_seed, p_fun, J_fun, ...
105                   a2_val, T2_offset_val, T3_offset_val, ...
106                   q_min, q_max, q_mid, K, tol, maxIter);
107
108 %% ================================
109 %  Trayectoria quíntica + IPTP
110 % ================================
111 n = 300; t = linspace(0,T,n);
112 Q = quinticTraj(q_init.', q_goal.', T, t);
113
114 % Límites dinámicos del xArm6 (usa valores del manual)
115 vel_max = deg2rad([180 180 180 180 180 180]);       % rad/s
116 acc_max = deg2rad([1500 1500 1500 1500 1500 1500]); % rad/s^2
117
118 % IPTP: escalar tiempo si excede límites (recalcula la quintic con T escalado)
119 [Q, t, T_scaled] = iptp_scale_time(Q, t, vel_max, acc_max);
120
121 %% ================================
122 %  Cálculo de velocidades y aceleraciones
```

```
% ================================
dt = t(2) - t(1);           % intervalo de muestreo
dQ = diff(Q) / dt;          % velocidades articulares (rad/s), tamaño (n-1) x 6
ddQ = diff(dQ) / dt;        % aceleraciones articulares (rad/s^2), tamaño (n-2) x 6

% Velocidad cartesiana del efector usando Jacobiano lineal
v_cart = zeros(n-1,3);
for k = 1:(n-1)
    J_now = J_fun(Q(k,1),Q(k,2),Q(k,3),Q(k,4),Q(k,5),Q(k,6), a2_val, T2_offset_val, T3_offset_val);
    v_cart(k,:) = (J_now * dQ(k,:)').'; % [vx vy vz] en mm/s si p_fun/T_fun están en mm
end

%% ================================
%  Impresión de resultados (físicos y matemáticos)
% ================================
q0      = q_init(:).';
qf      = q_goal(:).';

% Valores físicos (sumando offsets en q2 y q3)
q0_phys = q0; q0_phys(2) = q0_phys(2) + T2_offset_val; q0_phys(3) = q0_phys(3) + T3_offset_val;
qf_phys = qf; qf_phys(2) = qf_phys(2) + T2_offset_val; qf_phys(3) = qf_phys(3) + T3_offset_val;

fprintf('\n=== Estado ===\n');
fprintf('Duración planificada (IPTP): %.2f s\n', T_scaled);

disp('\n=== Articulaciones iniciales (matemático) [deg] ==='); disp(rad2deg(q0));
disp('=== Articulaciones finales (matemático) [deg]   ==='); disp(rad2deg(qf));
disp('=== Articulaciones iniciales (físico) [deg]     ==='); disp(rad2deg(q0_phys));
disp('=== Articulaciones finales (físico) [deg]       ==='); disp(rad2deg(qf_phys));
disp('=== Velocidades articulares iniciales [deg/s] ==='); disp(rad2deg(dQ(1,:)));
disp('=== Velocidades articulares finales [deg/s] ==='); disp(rad2deg(dQ(end,:)));
disp('=== Aceleraciones articulares iniciales [deg/s^2] ==='); disp(rad2deg(ddQ(1,:)));
disp('=== Aceleraciones articulares finales [deg/s^2] ==='); disp(rad2deg(ddQ(end,:)));

%% ================================
%  Trayectoria del efector final
% ================================
P = zeros(n,3);
for k=1:n
    Tnow = T_fun(Q(k,1),Q(k,2),Q(k,3),Q(k,4),Q(k,5),Q(k,6), a2_val, T2_offset_val, T3_offset_val);
    P(k,:) = Tnow(1:3,4).';
end

%% ================================
%  Gráficas con anotaciones (posiciones físicas)
% ================================
% Añadir offsets a J2 y J3 para graficar valores físicos
Q_phys = Q;
Q_phys(:,2) = Q_phys(:,2) + T2_offset_val; % offset J2
Q_phys(:,3) = Q_phys(:,3) + T3_offset_val; % offset J3

figure;
plot(t, rad2deg(Q_phys), 'LineWidth',1.5); hold on;
xlabel('Tiempo [s]'); ylabel('Ángulo [°]');
legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
title('Trayectoria articular (quíntica + IPTP, valores físicos)');

% Puntos inicio/fin visibles pero EXCLUIDOS de la leyenda
for i=1:6
    plot(t(1), rad2deg(Q_phys(1,i)), 'go', 'MarkerFaceColor','g', 'HandleVisibility','off'); % inicio
    text(t(1), rad2deg(Q_phys(1,i)), sprintf('%.1f°', rad2deg(q0_phys(i))), ...
        'VerticalAlignment','bottom','HorizontalAlignment','right');
    plot(t(end), rad2deg(Q_phys(end,i)), 'ro', 'MarkerFaceColor','r', 'HandleVisibility','off'); % final
    text(t(end), rad2deg(Q_phys(end,i)), sprintf('%.1f°', rad2deg(qf_phys(i))), ...
        'VerticalAlignment','top','HorizontalAlignment','left');
end

%% ================================
%  Gráficas de velocidades y aceleraciones articulares
% ================================
% Nota: dQ es de longitud n-1 y ddQ de n-2
figure;
plot(t(1:end-1), rad2deg(dQ), 'LineWidth',1.2); grid on;
xlabel('Tiempo [s]'); ylabel('Velocidad [°/s]');
legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
```

```
title('Velocidades articulares');

figure;
plot(t(1:end-2), rad2deg(ddQ), 'LineWidth',1.2); grid on;
xlabel('Tiempo [s]'); ylabel('Aceleración [°/s^2]');
legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
title('Aceleraciones articulares');

%% ================================
%  Gráfica de velocidad y aceleración cartesiana del efector
% ================================
figure;
plot(t(1:end-1), v_cart, 'LineWidth',1.2); grid on;
xlabel('Tiempo [s]'); ylabel('Velocidad lineal [mm/s]');
legend('v_x','v_y','v_z','Location','bestoutside');
title('Velocidad cartesiana del efector');

a_cart = diff(v_cart) / dt;       % aceleración cartesiana (mm/s^2), ...
%     tamaño (n-2) x 3
figure;
plot(t(1:end-2), a_cart, 'LineWidth',1.2); grid on;
xlabel('Tiempo [s]'); ylabel('Aceleración lineal [mm/s^2]');
legend('a_x','a_y','a_z','Location','bestoutside');
title('Aceleración cartesiana del efector');

%% ================================
%  Trayectoria del efector final (posición)
% ================================
figure;
plot3(P(:,1),P(:,2),P(:,3),'b-','LineWidth',2); hold on;
plot3(P(1,1),P(1,2),P(1,3),'go','MarkerFaceColor','g'); text(P(1,1),P(1,2),P(1,3),'Inicio', ...
    'VerticalAlignment','bottom');
plot3(P(end,1),P(end,2),P(end,3),'ro','MarkerFaceColor','r'); text(P(end,1),P(end,2),P(end,3), ...
    'Final','VerticalAlignment','top');
grid on; xlabel('X [mm]'); ylabel('Y [mm]'); zlabel('Z [mm]');
title('Trayectoria del efector final');

%% ================================
%  Funciones auxiliares
% ================================
function q_sol = newtonIK(p_d, q_seed, p_fun, J_fun, a2_val, T2_offset_val, T3_offset_val, q_min, ...
    q_max, q_mid, K, tol, maxIter)
    % -NewtonRaphson SIN clamping (solo penalización suave de límites)
    qk = q_seed(:);
    for it = 1:maxIter
        p_now = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), a2_val, T2_offset_val, T3_offset_val);
        J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), a2_val, T2_offset_val, T3_offset_val);
        e = p_d - p_now;
        if norm(e) < tol, break; end
        gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);
        dq = pinv(J_now) * e - K * gradH;
        qk = qk + dq;
    end
    q_sol = qk;
end

function Q = quinticTraj(q0, qf, T, t)
    % Polinomio quíntico con vel y acc cero en extremos
    n = numel(t); Q = zeros(n, numel(q0));
    for i=1:numel(q0)
        a0 = q0(i); a1 = 0; a2 = 0;
        a3 = 10*(qf(i)-q0(i))/T^3;
        a4 = -15*(qf(i)-q0(i))/T^4;
        a5 = 6*(qf(i)-q0(i))/T^5;
        Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
    end
end

function [Q_out, t_out, T_scaled] = iptp_scale_time(Q, t, vel_max, acc_max)
    % Escalado temporal para respetar límites de velocidad y aceleración
    dt = t(2) - t(1);
    dQ  = diff(Q)/dt;
    ddQ = diff(dQ)/dt;
    scale_v = max(max(abs(dQ) ./ vel_max));
    scale_a = max(max(abs(ddQ) ./ acc_max));
```

```
    scale = max([1, scale_v, sqrt(scale_a)]);
    if scale > 1
        T_scaled = t(end) * scale;
        t_out = linspace(0, T_scaled, numel(t));
        % Recalcular la quintic con el nuevo T para mantener vel/acc cero en extremos
        Q_out = zeros(size(Q));
        q0 = Q(1,:); qf = Q(end,:);
        for i=1:size(Q,2)
            a0 = q0(i); a1 = 0; a2 = 0;
            a3 = 10*(qf(i)-q0(i))/T_scaled^3;
            a4 = -15*(qf(i)-q0(i))/T_scaled^4;
            a5 = 6*(qf(i)-q0(i))/T_scaled^5;
            Q_out(:,i) = a0 + a1*t_out + a2*t_out.^2 + a3*t_out.^3 + a4*t_out.^4 + a5*t_out.^5;
        end
    else
        Q_out = Q; t_out = t; T_scaled = t(end);
    end
end
```

Código 11: Control cinemático completo.

```
Objetivo a 455.5 mm, dentro del alcance máximo 700.0 mm -> Realizable

=== Estado ===
Duración planificada (IPTP): 5.00 s
\n=== Articulaciones iniciales (matemático) [deg] ===
     0     0     0     0     0     0

=== Articulaciones finales (matemático) [deg]   ===
   21.6552   -4.7464  -25.0741   -7.6934   -0.3000         0

=== Articulaciones iniciales (físico) [deg]     ===
         0  -79.3500   79.3500         0         0         0

=== Articulaciones finales (físico) [deg]       ===
   21.6552  -84.0963   54.2758   -7.6934   -0.3000         0

=== Velocidades articulares iniciales [deg/s] ===
   1.0e-03 *

    0.4820   -0.1057   -0.5581   -0.1712   -0.0067         0

=== Velocidades articulares finales [deg/s] ===
   1.0e-03 *

    0.4820   -0.1057   -0.5581   -0.1712   -0.0067         0

=== Aceleraciones articulares iniciales [deg/s^2] ===
    0.1718   -0.0377   -0.1989   -0.0610   -0.0024         0

=== Aceleraciones articulares finales [deg/s^2] ===
   -0.1718    0.0377    0.1989    0.0610    0.0024
```

Código 12: Salida del control cinemático completo.

4. Estudio dinámico

El fabricante proporciona los parámetros clásicos y modificados de Denavit-Hartenberg en el manual del usuario. En las siguientes secciones se van a estudiar dichos parámetros.

Referencias adicionales al material del Campus Virtual

[1] diagrams.net. *draw.io – Herramienta de diagramación en línea*. Accedido: 2 de diciembre de 2025. 2025. URL: https://www.drawio.com/.

[2] Inc. The MathWorks. *MATLAB Online*. https://matlab.mathworks.com/. Versión en línea de MATLAB accesible vía navegador. 2025.

[3] zDynamics. *ROBOT DELTA: CONTROL DE LAS ARTICULACIONES*. https://www.youtube.com/watch?v=m0zzhz-36Bw. Accedido: 2 de diciembre de 2025. 2023.

[4] UFactory. *xArm 6 Collaborative Robot*. https://www.ufactory.us/product/ufactory-xarm-6. Accedido: 8 de diciembre de 2025. 2025.

[5] Steven Macenski et al. «Robot Operating System 2: Design, architecture, and uses in the wild». En: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[6] Open Robotics y Gazebo Community. *Gazebo Harmonic*. https://gazebosim.org/docs/harmonic. Accedido: 8 de diciembre de 2025. 2025.

[7] Open Robotics. *ROS 2 Jazzy Jalisco*. https://docs.ros.org/en/jazzy/. Accedido: 8 de diciembre de 2025. 2024.

[8] xArm-Developer. *xArm ROS 2*. https://github.com/xArm-Developer/xarm_ros2. Repositorio en GitHub. Accedido: 8 de diciembre de 2025. 2025.

[9] UFactory. *UFactory Docs, xArm User Manual V2.0.0*. Accedido: 8 de diciembre de 2025. 2023.

[10] UFactory. *xArm Collaborative Robot*. https://www.ufactory.cc/xarm-collaborative-robot/. Accedido: 8 de diciembre de 2025. 2025.

[11] Maroš Majchrák et al. «Analysis of harmonic gearbox tooth contact pressure». En: *IOP Conference Series: Materials Science and Engineering* 659 (oct. de 2019), pág. 012068. DOI: 10.1088/1757-899X/659/1/012068.

[12] Antonio Barrientos et al. *Fundamentos de robótica*. spa. 2.ª ed. Madrid: McGraw-Hill Interamericana de España, 2007. ISBN: 9788448156367.

[13] Intel Corporation. *Intel RealSense Depth Camera D435*. Datasheet y documentación técnica. 2018. URL: https://www.intelrealsense.com/depth-camera-d435/.

[14] International Organization for Standardization. *ISO 9409-1:2004, Mechanical interfaces for industrial robots – Part 1: Plates for connection of end-effectors*. Norma internacional sobre interfaces mecánicas de robots industriales. 2004. URL: https://www.iso.org/standard/35495.html.

[15] UFactory. *xArm Studio User Manual*. https://www.ufactory.us/ufactory-studio. Accedido: 9 de diciembre de 2025. 2025.

[16] Peter I. Corke. «A Simple and Systematic Approach to Assigning Denavit–Hartenberg Parameters». En: *Proceedings of the IEEE International Conference on Robotics and Automation* (1996), págs. 1834-1839. URL: https://petercorke.com/doc/simple_systematic.pdf.

[17] Peter Corke. *Robotics Toolbox*. 2025. URL: https://petercorke.com/toolboxes/robotics-toolbox/.

[18] John J. Craig. *Introduction to Robotics: Mechanics and Control*. 1st. Boston, MA: Addison-Wesley, 1986.

[19] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control — The Computer Control of Robot Manipulators*. Inf. téc. NASA Technical Paper 1800. Washington, D.C.: NASA, 1986. URL: https://ntrs.nasa.gov/api/citations/19860018481/downloads/19860018481.pdf.

[20] MoveIt Contributors. *MoveIt 2*. https://moveit.ros.org. Accessed: 2025-01-15. 2024.

[21] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. London: Springer, 2009. ISBN: 978-1-84628-641-4.

[22] Yoshihiko Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.

[23] Samuel R. Buss y Jin-Su Kim. «Selectively damped least squares for inverse kinematics». En: *Journal of Graphics Tools* 10.3 (2004), págs. 37-49.

[24] Charles W. Wampler. «Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods». En: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1 (1986), págs. 93-101.

[25] ros-controls. *ros2_control: A framework for robot control in ROS 2*. https://github.com/ros-controls/ros2_control. Accessed: December 17, 2025.