



UNIVERSIDAD  
**COMPLUTENSE**  
M A D R I D



# Trabajo final sobre la materia

**Máster en Ingeniería de Sistemas y Control**

**Asignatura:** Robótica Industrial

**Curso:** 2025/2026

**Alumno:** Javier Arambarri Calvo

**18 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* [[drawio](#)] o *Matlab* [[matlabonline2025](#)]. Para completar la comprensión del tema 7 se ha utilizado el siguiente vídeo [[controlDinamico](#)].

## Índice

<b>1. Introducción</b>	<b>7</b>
1.1. Objetivo y motivación del proyecto . . . . .	7
1.2. Metodología . . . . .	7
<b>2. <i>UFactory xArm 6</i></b>	<b>8</b>
2.1. Estructura mecánica: eslabones y articulaciones . . . . .	8
2.2. Actuadores . . . . .	9
2.3. Sensores integrados y opcionales . . . . .	9
2.4. Efecto . . . . .	11
2.5. Sistema de control . . . . .	12
2.6. Software . . . . .	13
<b>3. Estudio cinemático</b>	<b>14</b>
3.1. Cinemática directa . . . . .	14
3.1.1. Parámetros de Denavit-Hartenberg . . . . .	14
3.1.2. Parámetros modificados de Denavit-Hartenberg . . . . .	18
3.2. Cinemática inversa . . . . .	22
3.2.1. Pseudoinversa del Jacobiano y Newton-Raphson . . . . .	25
3.2.2. Cálculo de cinemática inversa para <i>UFactory xArm6</i> . . . . .	27
3.3. Cinemática diferencial y Jacobiano . . . . .	31
3.4. Generación de trayectorias . . . . .	34
3.4.1. <b>Trayectorias lineales en espacio cartesiano</b> . . . . .	34
3.4.2. <b>Trayectorias circulares en espacio cartesiano</b> . . . . .	34
3.4.3. Corrección de discontinuidades mediante <i>IPTP</i> . . . . .	34
3.4.4. <b>Trayectorias polinómicas en espacio articular</b> . . . . .	35
3.4.5. <b>Trayectorias spline (multi-waypoint)</b> . . . . .	36
3.5. Cálculo de la trayectoria en <i>Matlab</i> con la toolbox de Peter Corke . . . . .	37
3.6. Control cinemático . . . . .	41
<b>4. Estudio dinámico</b>	<b>49</b>
4.1. Ecuaciones de movimiento . . . . .	51
4.1.1. Enfoque energético (Lagrangiano) . . . . .	51
4.1.2. Método recursivo de Newton-Euler . . . . .	52
4.1.3. Ejemplo de una iteración para el eslabón 3 del <i>UFactory xArm6</i> . . . . .	53
<b>Referencias adicionales al material del Campus Virtual</b>	<b>56</b>

# Índice de figuras

1.	<i>UFactory xArm 6</i> . Fuente: [ufactory_xarm]. . . . .	8
2.	<i>Reductor armónico</i> . Fuente: [articleReductor]. . . . .	9
3.	Ejemplo de elemento terminal. Fuente: [ufactory_xarm6]. . . . .	11
4.	<i>Control box analógico</i> . Fuente: [ufactory_xarm6]. . . . .	12
5.	<i>Control box digital</i> . Fuente: [ufactory_xarm6]. . . . .	12
6.	<i>xArm Studio</i> . Fuente: [ufactory_xarm6]. . . . .	13
7.	Parámetros clásicos Denavit-Hartenberg. Fuente: [ufactory_xarm_manual]. . . . .	15
8.	Parámetros modificados Denavit-Hartenberg. Fuente: [ufactory_xarm_manual]. . . . .	19
9.	Trayectoria articular realizable. . . . .	40
10.	Trayectoria realizable del efecto. . . . .	40
11.	Trayectoria articular realizable. . . . .	41
12.	Trayectoria realizable del efecto. . . . .	42
13.	Velocidad cartesiana del efecto. . . . .	42
14.	Velocidades articulares. . . . .	43
15.	Aceleración cartesiana del efecto. . . . .	43
16.	Aceleraciones articulares. . . . .	44
17.	Sistemas de referencia para el estudio dinámico. Fuente: [ufactory_xarm_manual]. . . . .	49
18.	Parámetros de la dinámica del robot. Fuente: [ufactory_xarm_manual]. . . . .	49

# Índice de cuadros

1.	Características mecánicas y de actuadores del <i>xArm 6</i> . . . . .	10
2.	Resumen de tipo de señal, comunicación y alimentación de los sensores del <i>xArm 6</i> . .	10
3.	Resumen de tipo de señal, comunicación y alimentación de los elementos terminales del efecto del <i>xArm 6</i> . . . . .	11
4.	Resumen de las especificaciones técnicas de las unidades de control del <i>xArm 6</i> . .	13
5.	Parámetros DH modificados del <i>UFactory xArm6</i> para el estudio dinámico. . . . .	49

# Índice de códigos

1.	Código Matlab para la cinemática directa con parámetros DH clásicos. . . . .	16
2.	Salida numérica de la cinemática directa DH clásica. . . . .	17
3.	Código Matlab para la cinemática directa con parámetros DH modificados. . . . .	21
4.	Salida numérica de la cinemática directa DH modificada. . . . .	21
5.	Código Matlab para obtener las ecuaciones de posición y orientación con parámetros DH modificados. . . . .	24
6.	Cálculo de la cinemática inversa del robot <i>UFactory xArm6</i> . . . . .	28
7.	Salida del cálculo de la cinemática inversa del robot <i>UFactory xArm6</i> . . . . .	30
8.	Cálculo de la cinemática diferencial y Jacobiano mediante parámetros DH modificados. . . . .	31
9.	Generación de trayectoria articular realizable. . . . .	37
10.	Salida de la generación de trayectoria articular realizable. . . . .	39
11.	Control cinemático completo. . . . .	44
12.	Salida del control cinemático completo. . . . .	48
13.	Momentos de inercia de los eslabones del robot en estudio. . . . .	49

## 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* [[ufactory\\_xarm6](#)] en simulación mediante el *framework ROS2* [[doi:10.1126/scirobotics.abm6074](#)] 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* [[gazebo\\_harmonic](#)] y la distribución *Jazzy Jalisco* [[ros2\\_jazzy](#)] de ROS2. Se ha utilizado la documentación oficial del fabricante, tanto el repositorio de ROS2 [[xarm\\_ros2](#)] como el manual de usuario del robot [[ufactory\\_xarm\\_manual](#)].

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.

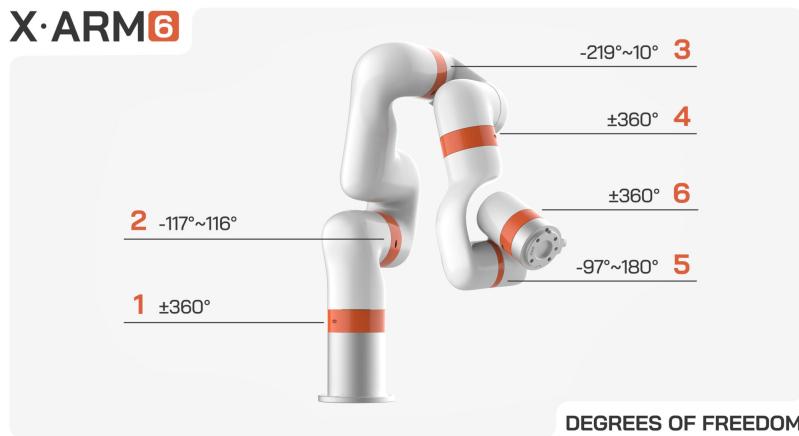


Figura 1: *UFactory xArm 6*. Fuente: [ufactory\_xarm].

### 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 carcasa mecanizada 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 carcasa 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 ±360°, mientras que las articulaciones intermedias permiten rotaciones de hasta ±180°, y las articulaciones de muñeca alcanzan rangos de ±360°, lo que proporciona gran flexibilidad para tareas de manipulación y orientación del efecto 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 [articleReductor], 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.

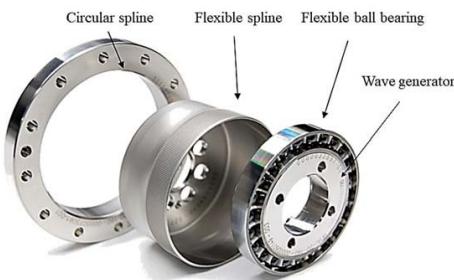


Figura 2: Reductor armónico. Fuente: [articleReductor].

## 2.3. Sensores integrados yopcionales

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 [barrientos2007fundamentos]. 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

Parámetro	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Rango de trabajo	$\pm 360^\circ$	-117° – 116°	-219° – 10°	$\pm 360^\circ$	-97° – 180°	$\pm 360^\circ$
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 °/s <sup>2</sup>			
Carga útil máxima				5 kg		
Potencia nominal				150 W		
Repetibilidad				$\pm 0.1$ mm		

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

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 *grippers* mecánicos y bio *grippers*, 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 efecto 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* [[IntelRealSenseD435](#)], recomendada por el fabricante.



Figura 3: Ejemplo de elemento terminal. Fuente: [[ufactory\\_xarm6](#)].

La brida del robot cumple con el patrón de taladros indicado en la ISO 9409-1:2004 [[ISO9409-1](#)], 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 efecto y permite medir fuerzas y momentos en las tres direcciones espaciales, ampliando las capacidades del manipulador en tareas de ensamblaje y manipulación delicada.

El efecto 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	<i>Control box / gripper</i>
<i>Gripper</i> de vacío	Analógica / Digital	Interna	<i>Control box / gripper</i>
<i>Bio gripper</i>	Digital	Interna	<i>Control box / gripper</i>

Cuadro 3: Resumen de tipo de señal, comunicación y alimentación de los elementos terminales del efecto 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 24V 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 efecto 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: [\[ufactory\\_xarm6\]](#).



Figura 5: *Control box* digital. Fuente: [\[ufactory\\_xarm6\]](#).

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 [[xarm\\_ros2](#)]. 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* [[ufactory\\_xarmstudio](#)], 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: [[ufactory\\_xarm6](#)].

### 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 [barrientos2007fundamentos] proporcionados por el fabricante, con

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

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

Size Parameters: (unit: mm)			Joint Coordinate Definition		
Kinematics	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: [ufactory\_xarm\_manual].

Para alcanzar los sistemas 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  $\theta$  de la articulación y se suma directamente a  $\theta_i$ .

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 los parámetros longitudinales  $d_1$ ,  $a_3$ ,  $a_5$ ,  $d_6$ .

La secuencia de transformaciones [corke1996] y la matriz de transformación homogénea [barrientos2007fundamentos] asociada a dichas transformaciones se presentan en las ecuaciones 1 y 2:

$${}^{i-1}A_i = \mathbf{R}_Z(\theta_i^* = \theta_i + offset_i) \mathbf{T}_Z(d_i) \mathbf{T}_X(a_i) \mathbf{R}_X(\alpha_i) \quad (1)$$

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

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

$$\begin{aligned} A_1 &= \begin{bmatrix} C(q_1) & 0 & -S(q_1) & 0 \\ S(q_1) & 0 & C(q_1) & 0 \\ 0 & -1 & 0 & 267 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ A_2 &= \begin{bmatrix} C(q_2 + offset_2) & -S(q_2 + offset_2) & 0 & a_2 C(q_2 + offset_2) \\ S(q_2 + offset_2) & C(q_2 + offset_2) & 0 & a_2 S(q_2 + offset_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_2 = 289,48866 \\ A_3 &= \begin{bmatrix} C(q_3 + offset_3) & 0 & -S(q_3 + offset_3) & a_3 C(q_3 + offset_3) \\ S(q_3 + offset_3) & 0 & C(q_3 + offset_3) & a_3 S(q_3 + offset_3) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_3 = 77,5 \\ A_4 &= \begin{bmatrix} C(q_4) & 0 & S(q_4) & 0 \\ S(q_4) & 0 & -C(q_4) & 0 \\ 0 & 1 & 0 & 342,5 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_5 = \begin{bmatrix} C(q_5) & 0 & -S(q_5) & a_5 C(q_5) \\ S(q_5) & 0 & C(q_5) & a_5 S(q_5) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_5 = 76 \\ A_6 &= \begin{bmatrix} C(q_6) & -S(q_6) & 0 & 0 \\ S(q_6) & C(q_6) & 0 & 0 \\ 0 & 0 & 1 & 97 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Siendo la transformación total:

$$T_0^6(q) = A_1(q_1) A_2(q_2 + offset_2) A_3(q_3 + offset_3) A_4(q_4) A_5(q_5) A_6(q_6)$$

Podemos calcular y comprobar el resultado en *Matlab* [[matlabonline2025](#)] y la *toolbox* de Peter Corke [[corke\\_robotics\\_toolbox](#)]:

```

1 clear; clc;
2
3 % Definir símbolos
4 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
5 pi2 = sym(pi)/2;
6
7 % Definición de eslabones según tu tabla DH
8 L1_link = Revolute('d', 267, 'a', 0, 'alpha', -pi2); % Eslabón 1
9 L2_link = Revolute('a', a2, 'alpha', 0, 'offset', T2_offset); % Eslabón 2
10 L3_link = Revolute('a', 77.5, 'alpha', -pi2, 'offset', T3_offset); % Eslabón 3
11 L4_link = Revolute('d', 342.5, 'alpha', pi2); % Eslabón 4

```

```

12 L5_link = Revolute('a', 76, 'alpha', -pi2); % Eslabón 5
13 L6_link = Revolute('d', 97, 'alpha', 0); % Eslabón 6
14
15 % Crear el robot
16 robot = SerialLink([L1_link L2_link L3_link L4_link L5_link L6_link], 'name', 'MiRobot');
17
18 % Vector de articulaciones
19 q = [q1 q2 q3 q4 q5 q6];
20
21 % Construir la transformación total manualmente con las matrices .T
22 T01 = simplify(L1_link.A(q1).T);
23 T12 = simplify(L2_link.A(q2).T);
24 T23 = simplify(L3_link.A(q3).T);
25 T34 = simplify(L4_link.A(q4).T);
26 T45 = simplify(L5_link.A(q5).T);
27 T56 = simplify(L6_link.A(q6).T);
28
29 % Transformación total simbólica
30 T06_sym = simplify(T01 * T12 * T23 * T34 * T45 * T56);
31
32 disp('Transformación total T_0^6 simbólica:');
33 disp(T06_sym);
34
35 % Sustitución numérica (ejemplo con tus valores y q_i = 0)
36 S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
37 S.a2 = 289.48866;
38 S.T2_offset = deg2rad(-79.34995);
39 S.T3_offset = deg2rad(79.34995);
40
41 T06_num = subs(T06_sym, S);
42 T06_num = vpa(T06_num, 10);
43
44 disp('Transformación total T_0^6 numérica:');
45 disp(T06_num);

```

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

```

1 Transformación total T_0^6 simbólica:
2 ... [muy larga para exponer]
3
4 Transformación total T_0^6 numérica:
5 [1.0, 0, 0, 207.0003735]
6 [ 0, -1.0, 0, 0]
7 [ 0, 0, -1.0, 112.0020111]
8 [ 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 [craig1986, paul1986], 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 [corke1996] y la matriz de transformación homogénea asociada a dichas transformaciones se presentan en las ecuaciones 3 y 4:

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

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

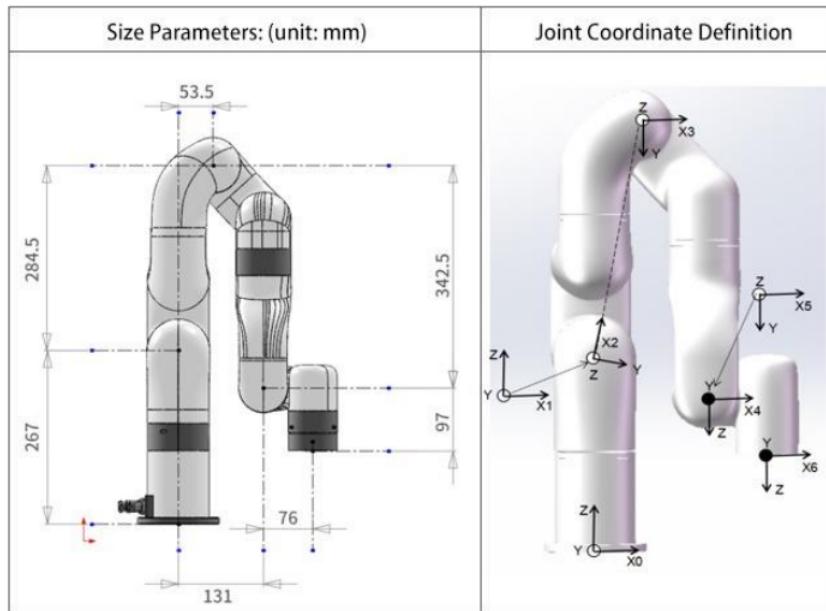
Esta convención ofrece ventajas como la asignación única y consistente de marcos a cada eslabón porque simplifica la implementación computacional y la integración con modelos de diseño asistido por computadora (CAD), donde se prefieren coordenadas locales de eslabón [corke1996]. Además, facilita los 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

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

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

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



Kinematics	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: [\[ufactory\\_xarm\\_manual\]](#).

Al igual que con los parámetros clásicos, sección 3.1.1, para alcanzar los sistemas 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:

$$\begin{aligned}
A_1 &= \begin{bmatrix} C(q_1) & -S(q_1) & 0 & 0 \\ S(q_1) & C(q_1) & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad d_1 = 267 \\
A_2 &= \begin{bmatrix} C(q_2 + T2_{\text{offset}}) & -S(q_2 + T2_{\text{offset}}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -S(q_2 + T2_{\text{offset}}) & -C(q_2 + T2_{\text{offset}}) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
A_3 &= \begin{bmatrix} C(q_3 + T3_{\text{offset}}) & -S(q_3 + T3_{\text{offset}}) & 0 & a_2 \\ S(q_3 + T3_{\text{offset}}) & C(q_3 + T3_{\text{offset}}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_2 = 289,48866 \\
A_4 &= \begin{bmatrix} C(q_4) & -S(q_4) & 0 & a_3 \\ 0 & 0 & 1 & d_4 \\ -S(q_4) & -C(q_4) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_3 = 77,5, \quad d_4 = 342,5 \\
A_5 &= \begin{bmatrix} C(q_5) & 0 & S(q_5) & 0 \\ S(q_5) & 0 & -C(q_5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
A_6 &= \begin{bmatrix} C(q_6) & -S(q_6) & 0 & a_5 \\ 0 & 0 & 1 & d_6 \\ -S(q_6) & -C(q_6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_5 = 76, \quad d_6 = 97
\end{aligned}$$

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* [[matlabonline2025](#)] y la *toolbox* de Peter Corke [[corke\\_robotics\\_toolbox](#)]:

```

1 clear; clc;
2 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
3 pi2 = sym(pi)/2;
4
5 % Link([theta d a alpha sigma offset], 'modified')
6 L1_link = Link([0      267      0      0  0], 'modified');
7 L2_link = Link([0      0      -pi2      0  T2_offset], 'modified');
8 L3_link = Link([0      0      a2      0  0  T3_offset], 'modified');
9 L4_link = Link([0      342.5    77.5  -pi2      0  0], 'modified');
10 L5_link = Link([0      0      0      pi2      0  0], 'modified');
11 L6_link = Link([0      97      76  -pi2      0  0], 'modified');
12
13 robot = SerialLink([L1_link L2_link L3_link L4_link L5_link L6_link], ...
14                      'name', 'MiRobot', 'modified');
15
16 q = [q1 q2 q3 q4 q5 q6];
17
18 T01 = simplify(L1_link.A(q1).T);
19 T12 = simplify(L2_link.A(q2).T);
20 T23 = simplify(L3_link.A(q3).T);
21 T34 = simplify(L4_link.A(q4).T);
22 T45 = simplify(L5_link.A(q5).T);
23 T56 = simplify(L6_link.A(q6).T);
24
25 T06_sym = simplify(T01 * T12 * T23 * T34 * T45 * T56);
26
27 % Sustitución numérica (q_i=0 y offsets reales)
28 S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
29 S.a2 = 289.48866;
30 S.T2_offset = deg2rad(-79.34995);
31 S.T3_offset = deg2rad(79.34995);
32
33 T06_num = vpa(subs(T06_sym, S), 10);
34 disp(T06_num);

```

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

```

1 [1.0,      0,      0,  207.0003735]
2 [ 0,  -1.0,      0,          0]
3 [ 0,      0, -1.0, 112.0020111]
4 [ 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 cinemática 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 [barrientos2007fundamentos]. 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 [xarm\_ros2] utiliza *MoveIt 2* [moveit2], 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:

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

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

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

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

$$r_{11} = \sin q_1 (\cos q_4 \sin q_6 + \cos q_5 \cos q_6 \sin q_4) - \cos q_1 \cos \phi (\sin q_4 \sin q_6 - \cos q_4 \cos q_5 \cos q_6)$$

$$- \cos q_1 \sin \phi \cos q_6 \sin q_5$$

$$r_{12} = \sin q_1 (\cos q_4 \cos q_6 - \cos q_5 \sin q_4 \sin q_6) - \cos q_1 \cos \phi (\cos q_6 \sin q_4 + \cos q_4 \cos q_5 \sin q_6)$$

$$+ \cos q_1 \sin \phi \sin q_5 \sin q_6$$

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

$$r_{21} = - \cos q_1 (\cos q_4 \sin q_6 + \cos q_5 \cos q_6 \sin q_4) - \sin q_1 \cos \phi (\sin q_4 \sin q_6 - \cos q_4 \cos q_5 \cos q_6)$$

$$- \sin q_1 \sin \phi \cos q_6 \sin q_5$$

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

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

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

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

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

Estas ecuaciones se han obtenido con la ayuda de *Matlab* [matlabonline2025] y la *toolbox* de Peter Corke [corke\_robots\_toolbox]:

```

1 clear; clc;
2 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
3 pi2 = sym(pi)/2;
4
5 % --- Definición de eslabones DH MODIFICADO ---
6 L1 = Link([0      267      0      0  0], 'modified');
7 L2 = Link([0      0      0 -pi2      0  T2_offset], 'modified');
8 L3 = Link([0      0      a2      0  0  T3_offset], 'modified');
9 L4 = Link([0  342.5  77.5 -pi2      0  0], 'modified');
10 L5 = Link([0      0      0  pi2      0  0], 'modified');
11 L6 = Link([0      97      76 -pi2      0  0], 'modified');
12
13 % --- Matrices parciales ---
14 T01 = L1.A(q1).T;
15 T12 = L2.A(q2).T;
16 T23 = L3.A(q3).T;
17 T34 = L4.A(q4).T;
18 T45 = L5.A(q5).T;
19 T56 = L6.A(q6).T;
20
21 % --- Transformación completa ---
22 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
23 R06 = T06(1:3,1:3);
24 p06 = T06(1:3,4);
25
26 % --- IMPRESIÓN DE ECUACIONES ---
27 disp('== ECUACIONES DE POSICIÓN p06 ==');
28 disp('p_x ='); pretty(p06(1))
29 disp('p_y ='); pretty(p06(2))
30 disp('p_z ='); pretty(p06(3))
31
32 disp(' ');
33 disp('== ECUACIONES DE ORIENTACIÓN R06 ==');
34 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 [barrientos2007fundamentos], la relación entre las velocidades articulares y la velocidad del efecto 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,

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

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 efecto 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 [siciliano2009robotics]:

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

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 efecto 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

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

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

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

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

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

### 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

$$\begin{aligned} f(q_0) &= f(1) = 1^2 = 1, \\ x_d - f(q_0) &= 4 - 1 = 3, \\ J(q_0) &= 2q_0 = 2 \cdot 1 = 2, \quad J^\dagger(q_0) = \frac{1}{2}. \end{aligned}$$

Por tanto,

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

#### Iteración 2

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

Así,

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

### Iteración 3

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

La siguiente actualización es

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

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

$$q^* \approx 2,$$

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* [siciliano2009robotics, nakamura1991advanced], 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 [buss2004selectively].

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 [wampler1986manipulator, nakamura1991advanced]. 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  $\lambda$  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* [matlabonline2025] y la *Robotics Toolbox* de Peter Corke [corke\_robotics\_toolbox] con la que se ha programado la cinemática inversa para el *UFactory xArm6*:

```

1 clear; clc;
2 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
3 pi2 = sym(pi)/2;
4
5 %=====
6 % DEFINICIÓN DE ESLABONES (DH MODIFICADO)
7 %=====
8 L1 = Link([0 267 0 0 0 0], 'modified');
9 L2 = Link([0 0 -pi2 0 T2_offset], 'modified');
10 L3 = Link([0 a2 0 0 T3_offset], 'modified');
11 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified');
12 L5 = Link([0 0 pi2 0 0 0], 'modified');
13 L6 = Link([0 97 76 -pi2 0 0], 'modified');
14
15 %=====
16 % MATRICES DE TRANSFORMACIÓN PARCIALES
17 %=====
18 T01 = L1.A(q1).T;
19 T12 = L2.A(q2).T;
20 T23 = L3.A(q3).T;
21 T34 = L4.A(q4).T;
22 T45 = L5.A(q5).T;
23 T56 = L6.A(q6).T;
24
25 %=====
26 % TRANSFORMACIÓN COMPLETA T_0^6
27 %=====
28 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
29 p06 = T06(1:3,4);
30
31 %% =====
32 % JACOBIANO LINEAL (para cinemática inversa)
33 % =====
34 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
35
36 %% =====
37 % CONVERTIR A FUNCIONES NUMÉRICAS
38 % =====
39 p_fun = matlabFunction(p06, 'Vars', ...
40 {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});
41
42 J_fun = matlabFunction(Jv, 'Vars', ...
43 {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});
44
45 T_fun = matlabFunction(T06, 'Vars', ...
46 {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});

```

```

47 %%% =====
48 % PARÁMETROS NUMÉRICOS DEL ROBOT
49 % =====
50 % =====
51 a2_val = 289.48866;
52 T2_offset_val = deg2rad(-79.34995);
53 T3_offset_val = deg2rad(79.34995);
54
55
56 %%% =====
57 % CINEMÁTICA INVERSA NUMÉRICA (-NEWTONRAPHSON, SOLO POSICIÓN)
58 % =====
59 % Posición deseada (se sabe que corresponde a q_i = 0 con esos offsets)
60 p_d = [207; 0; 112];
61
62 % Configuración inicial
63 qk = zeros(6,1); % punto de partida
64
65 tol = 1e-6;
66 maxIter = 50;
67
68 % Límites matemáticos de las articulaciones (incluyendo offsets en 2 y 3)
69 % Rangos físicos (tabla):
70 % J1: ±360°
71 % J2: -117° -- 116° (físicos)
72 % J3: -219° -- 10° (físicos)
73 % J4: ±360°
74 % J5: -97° -- 180°
75 % J6: ±360°
76 %
77 % Convertimos a límites "matemáticos" para q2 y q3 (sin offset):
78 % q2_mat [q2_fis_min - offset2, q2_fis_max - offset2]
79 % q3_mat [q3_fis_min - offset3, q3_fis_max - offset3]
80 q1_min = -360; q1_max = 360;
81 q2_min = -117 - rad2deg(T2_offset_val); % grados matemáticos
82 q2_max = 116 - rad2deg(T2_offset_val);
83 q3_min = -219 - rad2deg(T3_offset_val);
84 q3_max = 10 - rad2deg(T3_offset_val);
85 q4_min = -360; q4_max = 360;
86 q5_min = -97; q5_max = 180;
87 q6_min = -360; q6_max = 360;
88
89 q_min = deg2rad([q1_min, q2_min, q3_min, q4_min, q5_min, q6_min])';
90 q_max = deg2rad([q1_max, q2_max, q3_max, q4_max, q5_max, q6_max])';
91
92 % Centro del rango (para penalización suave)
93 q_mid = (q_min + q_max) / 2;
94
95 % Ganancia del término de penalización (joint limit avoidance)
96 K = 0.005;
97
98 for k = 1:maxIter
99
100 % Evaluar posición y Jacobiano
101 % (incluyen offsets vía a2_val, T2_offset_val, T3_offset_val)
102 p_now = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), ...
103 a2_val, T2_offset_val, T3_offset_val); % 3x1
104 J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), ...
105 a2_val, T2_offset_val, T3_offset_val); % 3x6
106
107 % Error
108 e = p_d - p_now;
109
110 if norm(e) < tol
111 break
112 end
113
114 % == GRADIENTE DE LA FUNCIÓN DE EVITACIÓN DE LÍMITES ==
115 gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);
116
117 % Paso -NewtonRaphson (pseudoinversa) + penalización suave
118 dq = pinv(J_now) * e - K * gradH;
119
120 % Actualizar
121 qk = qk + dq;

```

```

122 end
123 % Redondear solución para limpiar ruido numérico
124 q_solution = round(qk, 12);
125
126 %% =====
127 % VERIFICACIÓN: EVALUAR T06(q*)
128 %
129 % =====
130 T_check = T_fun(q_solution(1), q_solution(2), q_solution(3), ...
131                 q_solution(4), q_solution(5), q_solution(6), ...
132                 a2_val, T2_offset_val, T3_offset_val);
133 p_check = T_check(1:3,4);
134
135 %% =====
136 % MOSTRAR RESULTADOS
137 % =====
138 disp('==> SOLUCIÓN DE CINEMÁTICA INVERSA (en radianes) ==')
139 disp(q_solution.)
140
141 disp('==> SOLUCIÓN DE CINEMÁTICA INVERSA (en grados) ==')
142 disp(rad2deg(q_solution.))
143
144 % Si se quieren ver los ángulos FÍSICOS de las articulaciones 2 y 3:
145 q_phys = q_solution;
146 q_phys(2) = q_phys(2) + T2_offset_val;
147 q_phys(3) = q_phys(3) + T3_offset_val;
148
149 disp('==> ÁNGULOS FÍSICOS (q + offset) EN GRADOS ==')
150 disp(rad2deg(q_phys.))
151
152 disp('==> POSICIÓN OBTENIDA ==')
153 disp(p_check)
154
155 disp('==> POSICIÓN DESEADA ==')
156 disp(p_d)
157
158 disp('==> ERROR VECTORIAL ==')
159 disp(e)
160
161 final_error = norm(e);
162 disp('==> ERROR FINAL ==')
163 disp(final_error)

```

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

```

1 ==> SOLUCIÓN DE CINEMÁTICA INVERSA (en radianes) ==
2      0   -0.0032   -0.0039      0    0.0154      0
3
4 ==> SOLUCIÓN DE CINEMÁTICA INVERSA (en grados) ==
5      0   -0.1856   -0.2235      0    0.8796      0
6
7 ==> ÁNGULOS FÍSICOS (q + offset) EN GRADOS ==
8      0   -79.5355   79.1264      0    0.8796      0
9
10 ==> POSICIÓN OBTENIDA ==
11     207.7231
12     0
13     112.1151
14
15 ==> POSICIÓN DESEADA ==
16     207
17     0
18     112
19
20 ==> ERROR VECTORIAL ==
21     -0.7231
22     0.0000
23     -0.1151
24
25 ==> ERROR FINAL ==
26     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* [matlabonline2025] y la *Robotics Toolbox* de Peter Corke [corke\_robotics\_toolbox]:

```

1 clear; clc;
2 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
3 pi2 = sym(pi)/2;
4
5 %% =====
6 % DEFINICIÓN DE ESLABONES (DH MODIFICADO)
7 % =====
8 L1 = Link([0 267 0 0 0 0], 'modified');
9 L2 = Link([0 0 -pi2 0 T2_offset], 'modified');
10 L3 = Link([0 a2 0 0 T3_offset], 'modified');
11 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified');
12 L5 = Link([0 0 pi2 0 0], 'modified');
13 L6 = Link([0 97 76 -pi2 0 0], 'modified');
14
15 %% =====
16 % MATRICES DE TRANSFORMACIÓN PARCIALES
17 % =====
18 T01 = L1.A(q1).T;
19 T12 = L2.A(q2).T;
20 T23 = L3.A(q3).T;
21 T34 = L4.A(q4).T;
22 T45 = L5.A(q5).T;
23 T56 = L6.A(q6).T;
24
25 %% =====
26 % TRANSFORMACIÓN COMPLETA T_0^6
27 % =====
28 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
29 R06 = T06(1:3,1:3);
30 p06 = T06(1:3,4);
31
32 %% =====
33 % IMPRESIÓN DE ECUACIONES SIMBÓlicas
34 % =====
35 disp('== ECUACIONES DE POSICIÓN p06 ==');
36 disp('p_x ='); pretty(p06(1))
37 disp('p_y ='); pretty(p06(2))
38 disp('p_z ='); pretty(p06(3))

```

```

39 disp(' ');
40 disp('== ECUACIONES DE ORIENTACIÓN R06 ==');
41 pretty(R06)
42
43 %% =====
44 % JACOBIANO GEOMÉTRICO (Jv + Jw)
45 % =====
46 % Jacobiano lineal usando jacobian
47 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
48
49 % Ejes z_i en el marco base
50 z0 = [0;0;1];
51 T01_ = T01;
52 T02_ = T01*T12;
53 T03_ = T01*T12*T23;
54 T04_ = T01*T12*T23*T34;
55 T05_ = T01*T12*T23*T34*T45;
56
57 z1 = T01_(1:3,3);
58 z2 = T02_(1:3,3);
59 z3 = T03_(1:3,3);
60 z4 = T04_(1:3,3);
61 z5 = T05_(1:3,3);
62
63 % Jacobiano angular
64 Jw = [z0 z1 z2 z3 z4 z5];
65
66 % Jacobiano geométrico completo
67 J = simplify([Jv; Jw]);
68
69 disp('== JACOBIANO GEOMÉTRICO ==');
70 pretty(J)
71
72 %% =====
73 % VELOCIDADES SIMBÓLICAS
74 % =====
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 efecto
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 EFECTO ==');
102 pretty(v_linear)
103
104 disp(' ');
105 disp('== VELOCIDAD ANGULAR DEL EFECTO ==');
106 pretty(v_angular)
107
108 %% =====
109 % EVALUACIÓN NUMÉRICA DE T06, J Y VELOCIDADES
110 % =====
111 disp(' ');
112 disp('== EVALUACIÓN NUMÉRICA DE T06, J Y VELOCIDADES ==');
113 disp('== EVALUACIÓN NUMÉRICA DE T06, J Y VELOCIDADES ==');

```

```

114 disp('====='); 
115 % Valores numéricos
116 S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
117 S.a2 = 289.48866;
118 S.T2_offset = deg2rad(-79.34995);
119 S.T3_offset = deg2rad(79.34995);
120
121 % Imprimir valores usados
122 disp(' ');
123 disp('== VALORES USADOS PARA LA EVALUACIÓN NUMÉRICA ==');
124 fprintf('q1 = %.4f rad\n', S.q1);
125 fprintf('q2 = %.4f rad\n', S.q2);
126 fprintf('q3 = %.4f rad\n', S.q3);
127 fprintf('q4 = %.4f rad\n', S.q4);
128 fprintf('q5 = %.4f rad\n', S.q5);
129 fprintf('q6 = %.4f rad\n', S.q6);
130 fprintf('a2 = %.5f mm\n', S.a2);
131 fprintf('T2_offset = %.4f rad (%.2f°)\n', S.T2_offset, rad2deg(S.T2_offset));
132 fprintf('T3_offset = %.4f rad (%.2f°)\n', S.T3_offset, rad2deg(S.T3_offset));
133
134 % Transformación numérica
135 T06_num = vpa(subs(T06, S), 10);
136 disp(' ');
137 disp('== TRANSFORMACIÓN HOMOGÉNEA T_0^6 NUMÉRICA ==');
138 disp(T06_num);
139
140 % Jacobiano numérico
141 J_num = vpa(subs(J, S), 10);
142 disp(' ');
143 disp('== JACOBIANO GEOMÉTRICO NUMÉRICO ==');
144 disp(J_num);
145
146 % Velocidades articulares numéricas
147 dq_val = [0.2; 0.1; -0.15; 0.05; 0.1; -0.05];
148 vel_num = double(J_num * dq_val);
149
150 v_linear_num = vel_num(1:3);
151 v_angular_num = vel_num(4:6);
152
153 disp(' ');
154 disp('== VELOCIDAD LINEAL DEL EFECTOR (numérica) ==');
155 disp(v_linear_num.');
156
157 disp(' ');
158 disp('== VELOCIDAD ANGULAR DEL EFECTOR (numérica) ==');
159 disp(v_angular_num.');
160

```

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 *Movelit 2* y el paquete *xarm\_planner*, integrados en el repositorio oficial [[xarm\\_ros2](#)]. Este planificador permite definir trayectorias en espacio articular y cartesiano, que son posteriormente ejecutadas en tiempo real a través de *ros2\_control* [[ros2\\_control](#)].

Los tipos de trayectorias que se pueden generar son [[barrientos2007fundamentos](#)]: lineales y circulares en espacio cartesiano, polinómicas en espacio articular y multipunto (*multi-waypoint*).

### 3.4.1. Trayectorias lineales en espacio cartesiano

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

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

con condiciones de contorno

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

Aunque la interpolación cartesiana es lineal, *Movelit* 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  $\theta(t)$ :

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

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:

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

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  $q(t)$ :

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

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:

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

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

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

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

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

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:

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

con condiciones de contorno en posición y velocidad:

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

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

Y el polinomio quíntico:

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

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

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

### 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:

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

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 realizable. 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.

```

1 clear; clc; close all;
2 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
3 pi2 = sym(pi)/2;
4
5 %% =====
6 % Definición de eslabones (DH modificado)
7 %% =====
8 L1 = Link([0 267 0 0 0 0], 'modified');
9 L2 = Link([0 0 -pi2 0 T2_offset], 'modified');
10 L3 = Link([0 a2 0 0 T3_offset], 'modified');
11 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified');
12 L5 = Link([0 0 pi2 0 0], 'modified');
13 L6 = Link([0 97 76 -pi2 0 0], 'modified');
14
15 %% =====
16 % Transformación completa T0^6
17 %% =====
18 T01 = L1.A(q1).T; T12 = L2.A(q2).T; T23 = L3.A(q3).T;
19 T34 = L4.A(q4).T; T45 = L5.A(q5).T; T56 = L6.A(q6).T;
20 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);

```

```

21 p06 = T06(1:3,4);
22 %
23 % Funciones numéricas
24 T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
25
26 %% =====
27 % Parámetros numéricos del robot
28 % =====
29 a2_val = 289.48866;
30 T2_offset_val = deg2rad(-79.34995);
31 T3_offset_val = deg2rad(79.34995);
32
33 % Límites articulares (matemáticos, en grados)
34 q1_min = -360; q1_max = 360;
35 q2_min = -117 - rad2deg(T2_offset_val);
36 q2_max = 116 - rad2deg(T2_offset_val);
37 q3_min = -219 - rad2deg(T3_offset_val);
38 q3_max = 10 - rad2deg(T3_offset_val);
39 q4_min = -360; q4_max = 360;
40 q5_min = -97; q5_max = 180;
41 q6_min = -360; q6_max = 360;
42
43 % Convertir a radianes
44 q_min = deg2rad([q1_min q2_min q3_min q4_min q5_min q6_min]);
45 q_max = deg2rad([q1_max q2_max q3_max q4_max q5_max q6_max]);
46
47 %% =====
48 % Trayectoria articular (quintic)
49 % =====
50 q0 = zeros(1,6); % inicio en 0
51 % finales aleatorios dentro de rango
52 qf = q_min + rand(1,6).*(q_max - q_min);
53
54 T = 5; % duración nominal inicial
55 n = 200; t = linspace(0,T,n);
56
57 % Polinomio quintico (vel y acc inicial/final = 0)
58 Q = zeros(n,6);
59 for i=1:6
60     a0 = q0(i); a1 = 0; a2 = 0;
61     a3 = 10*(qf(i)-q0(i))/T^3;
62     a4 = -15*(qf(i)-q0(i))/T^4;
63     a5 = 6*(qf(i)-q0(i))/T^5;
64     Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
65 end
66
67 %% =====
68 % IPTP: ajuste temporal
69 % =====
70 % Límites dinámicos (ejemplo, ajusta según datasheet)
71 vel_max = deg2rad([180 180 180 180 180 180]); % rad/s
72 acc_max = deg2rad([1500 1500 1500 1500 1500 1500]); % rad/s^2
73
74 dt = t(2)-t(1);
75 dQ = diff(Q)/dt; % velocidades
76 ddQ = diff(dQ)/dt; % aceleraciones
77
78 % Escalado temporal si excede límites
79 scale_v = max(max(abs(dQ)./vel_max));
80 scale_a = max(max(abs(ddQ)./acc_max));
81 scale = max([1, scale_v, sqrt(scale_a)]);
82
83 if scale > 1
84     T = T*scale; % estira el tiempo total
85     t = linspace(0,T,n);
86     for i=1:6
87         a0 = q0(i); a1 = 0; a2 = 0;
88         a3 = 10*(qf(i)-q0(i))/T^3;
89         a4 = -15*(qf(i)-q0(i))/T^4;
90         a5 = 6*(qf(i)-q0(i))/T^5;
91         Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
92     end
93 end
94 %% =====
95

```

```

96 % Visualización de articulaciones
97 % =====
98 figure;
99 plot(t, rad2deg(Q));
100 xlabel('Tiempo [s]'); ylabel('Ángulo [°]');
101 legend('q1','q2','q3','q4','q5','q6');
102 title('Trayectoria articular (quintica con IPTP)');
103
104 %% =====
105 % Trayectoria del efecto final
106 % =====
107 P = zeros(n,3);
108 for k=1:n
109     Tnow = T_fun(Q(k,1),Q(k,2),Q(k,3),Q(k,4),Q(k,5),Q(k,6), ...
110                 a2_val,T2_offset_val,T3_offset_val);
111     P(k,:) = Tnow(1:3,4)';
112 end
113
114 figure;
115 plot3(P(:,1),P(:,2),P(:,3),'b-','LineWidth',2);
116 grid on; xlabel('X'); ylabel('Y'); zlabel('Z');
117 title('Trayectoria del efecto final (quintica con IPTP)');
118
119 %% =====
120 % Imprimir posiciones inicial y final
121 % =====
122 % Valores físicos (sumando offset en q2 y q3)
123 q0_phys = q0;
124 q0_phys(2) = q0_phys(2) + T2_offset_val;
125 q0_phys(3) = q0_phys(3) + T3_offset_val;
126
127 qf_phys = qf;
128 qf_phys(2) = qf_phys(2) + T2_offset_val;
129 qf_phys(3) = qf_phys(3) + T3_offset_val;
130
131 disp('== Valores iniciales físicos (grados) ==');
132 disp(rad2deg(q0_phys));
133
134 disp('== Valores finales físicos (grados) ==');
135 disp(rad2deg(qf_phys));

```

Código 9: Generación de trayectoria articular realizable.

1	== Valores iniciales físicos (grados) ==
2	0 -79.3500 79.3500 0 0 0
3	
4	== Valores finales físicos (grados) ==
5	33.9952 -84.7005 -184.8117 -174.5941 135.8787 -176.9168

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

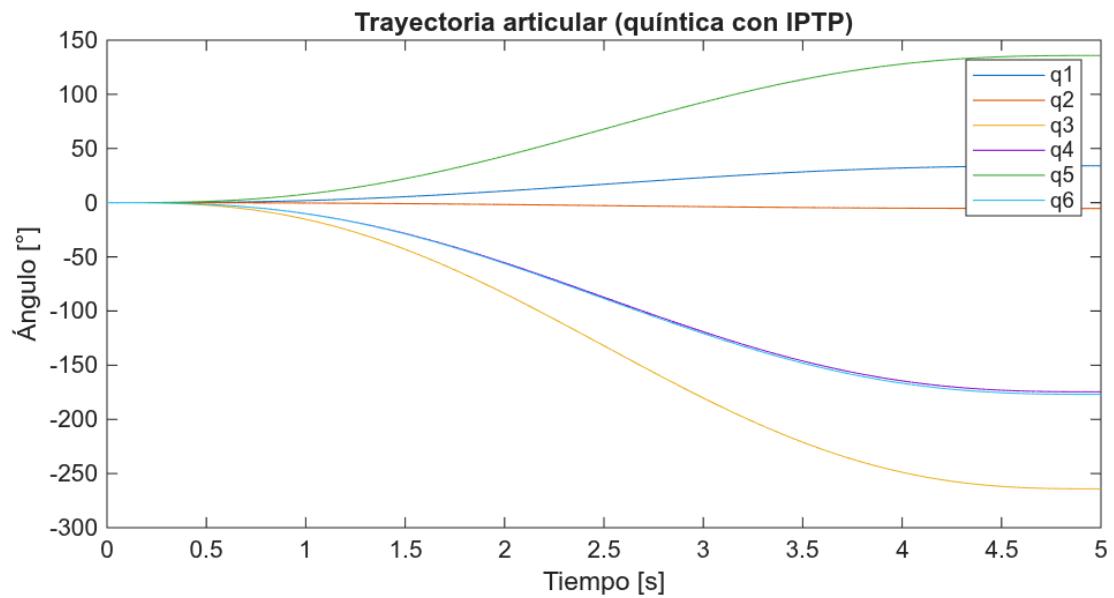


Figura 9: Trayectoria articular realizable.

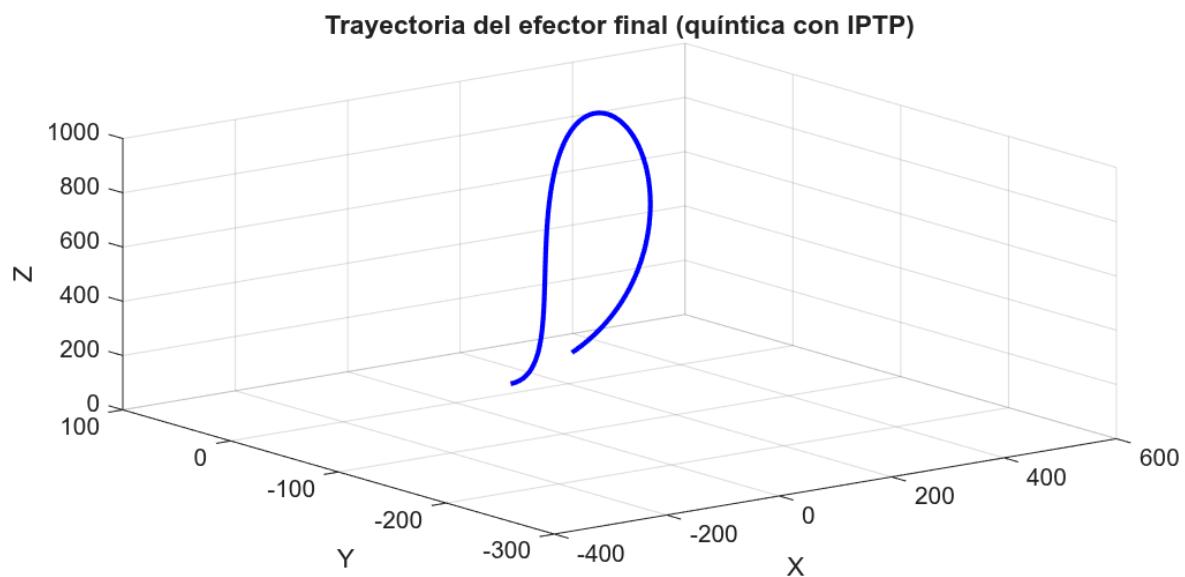


Figura 10: Trayectoria realizable del efecto.

### 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 efecto y las velocidades y aceleraciones articulares y cartesianas, y los códigos 11 y 12 la implementación y la salida por terminal.

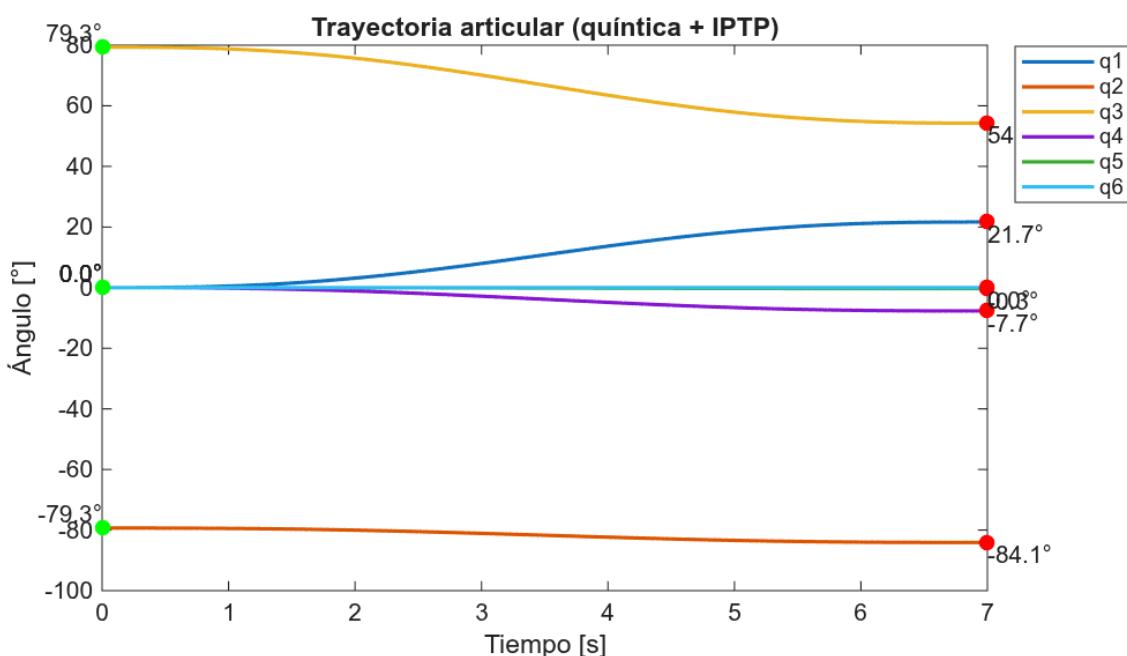


Figura 11: Trayectoria articular realizable.

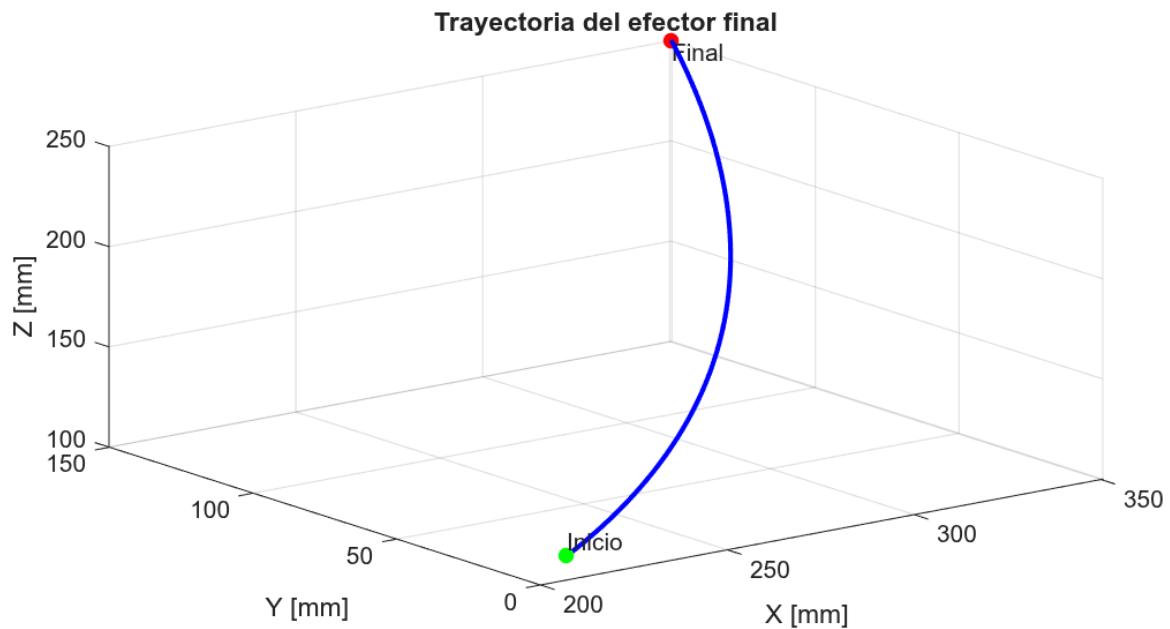


Figura 12: Trayectoria realizable del efecto.

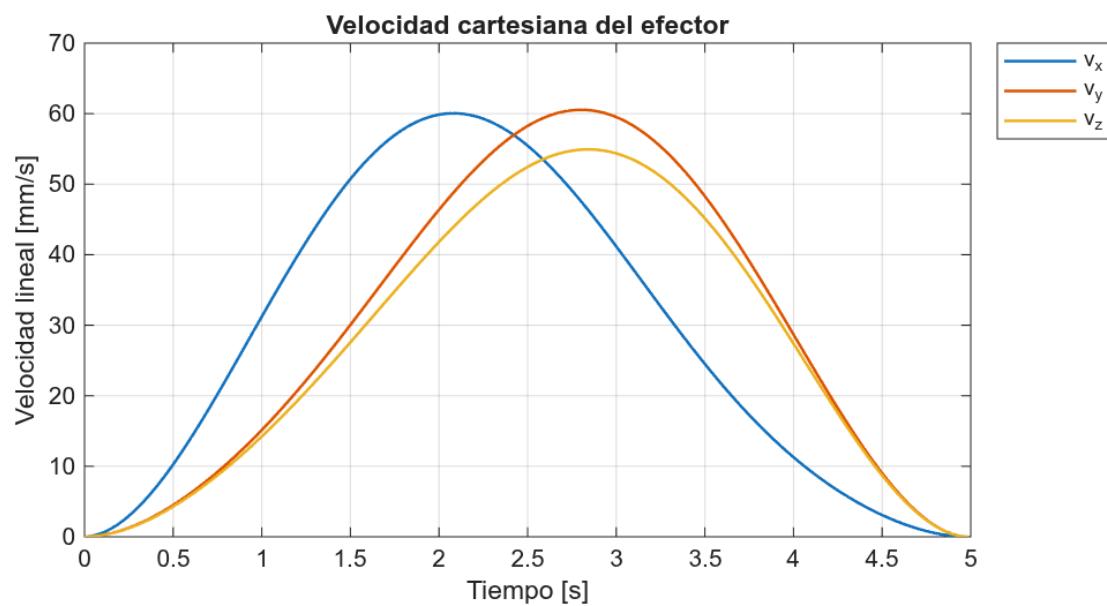


Figura 13: Velocidad cartesiana del efecto.

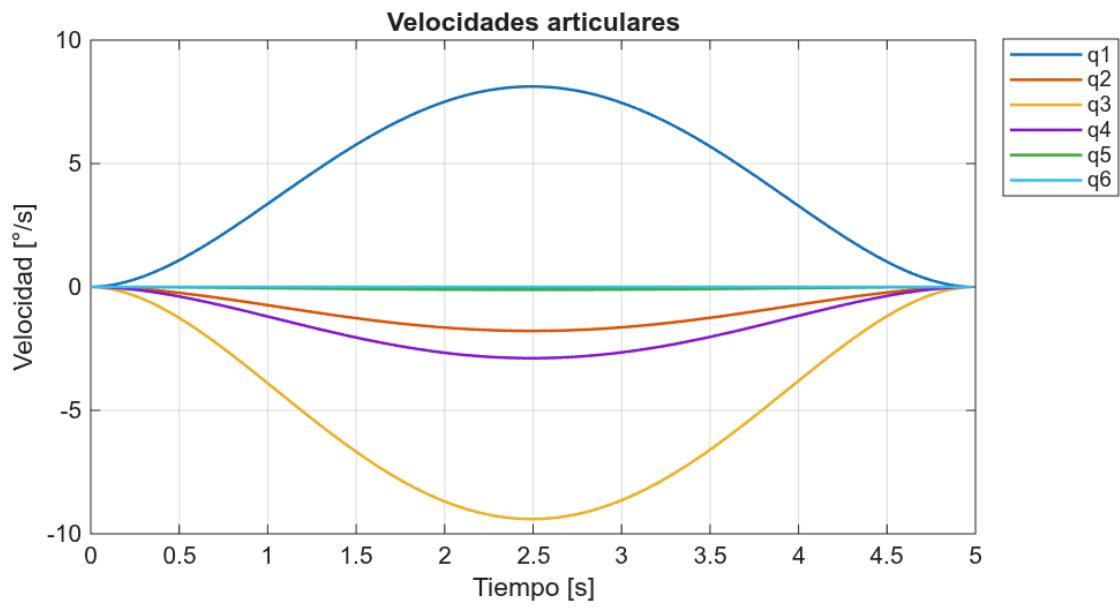


Figura 14: Velocidades articulares.

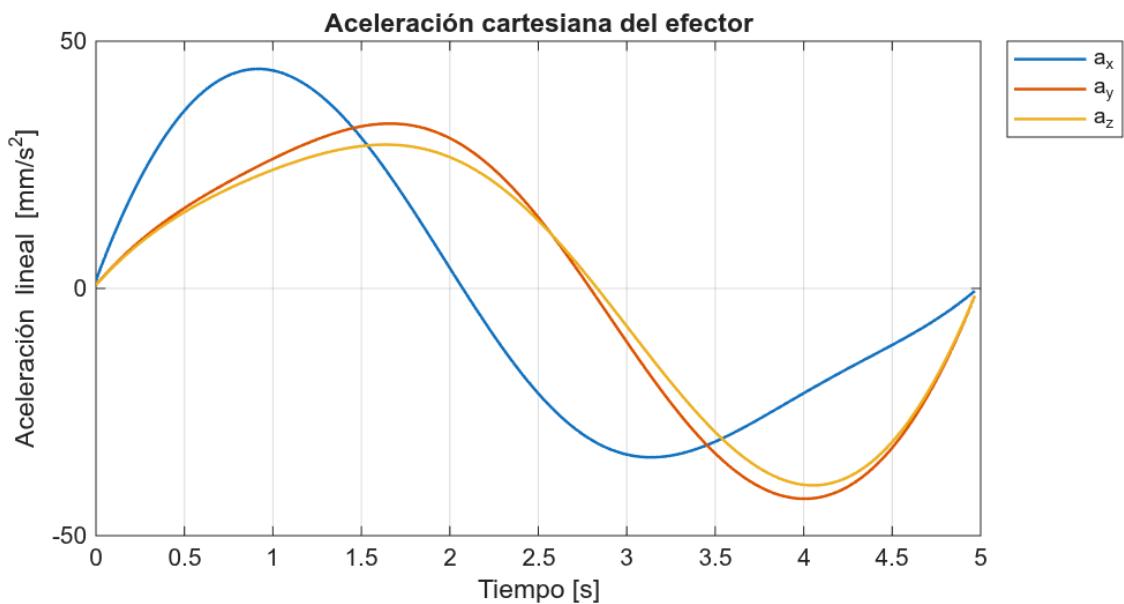


Figura 15: Aceleración cartesiana del efector.

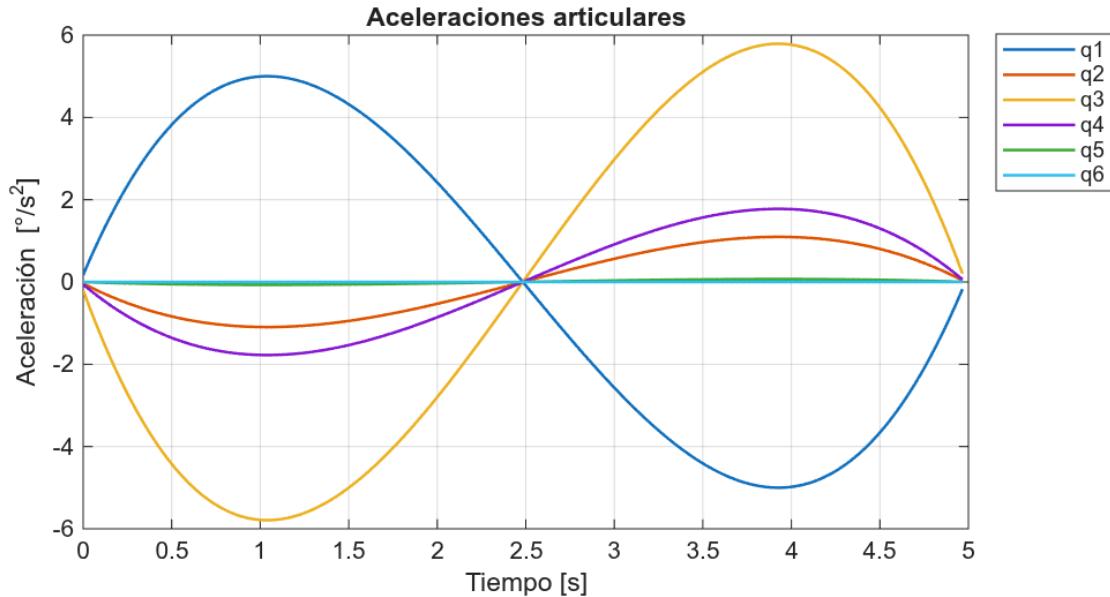


Figura 16: Aceleraciones articulares.

```

1 clear; clc; close all;
2 syms q1 q2 q3 q4 q5 q6 a2 T2_offset T3_offset real
3 pi2 = sym(pi)/2;
4
5 %%%%%%
6 % Modelo xArm6 (DH modificado)
7 %%%%%%
8 L1 = Link([0 267 0 0 0 0], 'modified');
9 L2 = Link([0 0 0 -pi2 0 T2_offset], 'modified');
10 L3 = Link([0 0 a2 0 0 T3_offset], 'modified');
11 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified');
12 L5 = Link([0 0 pi2 0 0 0], 'modified');
13 L6 = Link([0 97 76 -pi2 0 0], 'modified');
14
15 %%%%%%
16 % Transformaciones y Jacobiano
17 %%%%%%
18 T01 = L1.A(q1).T; T12 = L2.A(q2).T; T23 = L3.A(q3).T;
19 T34 = L4.A(q4).T; T45 = L5.A(q5).T; T56 = L6.A(q6).T;
20 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
21 p06 = T06(1:3,4);
22 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
23
24 % Funciones numéricas
25 p_fun = matlabFunction(p06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
26 J_fun = matlabFunction(Jv, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
27 T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
28
29 %%%%%%
30 % Parámetros del robot
31 %%%%%%
32 a2_val = 289.48866;
33 T2_offset_val = deg2rad(-79.34995);
34 T3_offset_val = deg2rad(79.34995);
35
36 % Límites físicos → matemáticos (deg)
37 q1_min = -360; q1_max = 360;
38 q2_min = -117 - rad2deg(T2_offset_val);
39 q2_max = 116 - rad2deg(T2_offset_val);
40 q3_min = -219 - rad2deg(T3_offset_val);
41 q3_max = 10 - rad2deg(T3_offset_val);
42 q4_min = -360; q4_max = 360;
43 q5_min = -97; q5_max = 180;
44 q6_min = -360; q6_max = 360;
45
46 q_min = deg2rad([q1_min, q2_min, q3_min, q4_min, q5_min, q6_min]);
47 q_max = deg2rad([q1_max, q2_max, q3_max, q4_max, q5_max, q6_max]);

```

```

48 %% =====
49 % Entrada: articulaciones actuales, pose final y duración
50 % =====
51
52 dlg = inputdlg( ...
53     {'Articulaciones actuales [q1 q2 q3 q4 q5 q6] en grados (matemático):', ...
54     'Posición final del efecto [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 efecto: ');
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

```

```

123 % =====
124 dt = t(2) - t(1); % intervalo de muestreo
125 dQ = diff(Q) / dt; % velocidades articulares (rad/s), tamaño (n-1) x 6
126 ddQ = diff(dQ) / dt; % aceleraciones articulares (rad/s^2), tamaño (n-2) x 6
127
128 % Velocidad cartesiana del efecto usando Jacobiano lineal
129 v_cart = zeros(n-1,3);
130 for k = 1:(n-1)
131     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);
132     v_cart(k,:) = (J_now * dQ(k,:)).'; % [vx vy vz] en mm/s si p_fun/T_fun están en mm
133 end
134
135 %% =====
136 % Impresión de resultados (físicos y matemáticos)
137 % =====
138 q0 = q_init(:).';
139 qf = q_goal(:).';
140
141 % Valores físicos (sumando offsets en q2 y q3)
142 q0_phys = q0; q0_phys(2) = q0_phys(2) + T2_offset_val; q0_phys(3) = q0_phys(3) + T3_offset_val;
143 qf_phys = qf; qf_phys(2) = qf_phys(2) + T2_offset_val; qf_phys(3) = qf_phys(3) + T3_offset_val;
144
145 fprintf('\n==== Estado ====\n');
146 fprintf('Duración planificada (IPTP): %.2f s\n', T_scaled);
147
148 disp('==== Articulaciones iniciales (matemático) [deg] ==='); disp(rad2deg(q0));
149 disp('==== Articulaciones finales (matemático) [deg] ==='); disp(rad2deg(qf));
150 disp('==== Articulaciones iniciales (físico) [deg] ==='); disp(rad2deg(q0_phys));
151 disp('==== Articulaciones finales (físico) [deg] ==='); disp(rad2deg(qf_phys));
152 disp('==== Velocidades articulares iniciales [deg/s] ==='); disp(rad2deg(dQ(1,:)));
153 disp('==== Velocidades articulares finales [deg/s] ==='); disp(rad2deg(dQ(end,:)));
154 disp('==== Aceleraciones articulares iniciales [deg/s^2] ==='); disp(rad2deg(ddQ(1,:)));
155 disp('==== Aceleraciones articulares finales [deg/s^2] ==='); disp(rad2deg(ddQ(end,:)));
156
157 %% =====
158 % Trayectoria del efecto final
159 % =====
160 P = zeros(n,3);
161 for k=1:n
162     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);
163     P(k,:) = Tnow(1:3,4).';
164 end
165
166 %% =====
167 % Gráficas con anotaciones (posiciones físicas)
168 % =====
169 % Añadir offsets a J2 y J3 para graficar valores físicos
170 Q_phys = Q;
171 Q_phys(:,2) = Q_phys(:,2) + T2_offset_val; % offset J2
172 Q_phys(:,3) = Q_phys(:,3) + T3_offset_val; % offset J3
173
174 figure;
175 plot(t, rad2deg(Q_phys), 'LineWidth',1.5); hold on;
176 xlabel('Tiempo [s]'); ylabel('Ángulo [°]');
177 legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
178 title('Trayectoria articular (quintica + IPTP, valores físicos');
179
180 % Puntos inicio/fin visibles pero EXCLUIDOS de la leyenda
181 for i=1:6
182     plot(t(1), rad2deg(Q_phys(1,i)), 'go', 'MarkerFaceColor','g', 'HandleVisibility','off'); % inicio
183     text(t(1), rad2deg(Q_phys(1,i)), sprintf('%1.1f', rad2deg(q0_phys(i))), ...
184         'VerticalAlignment','bottom','HorizontalAlignment','right');
185     plot(t(end), rad2deg(Q_phys(end,i)), 'ro', 'MarkerFaceColor','r', 'HandleVisibility','off'); % final
186     text(t(end), rad2deg(Q_phys(end,i)), sprintf('%1.1f', rad2deg(qf_phys(i))), ...
187         'VerticalAlignment','top','HorizontalAlignment','left');
188 end
189
190 %% =====
191 % Gráficas de velocidades y aceleraciones articulares
192 % =====
193 % Nota: dQ es de longitud n-1 y ddQ de n-2
194 figure;
195 plot(t(1:end-1), rad2deg(dQ), 'LineWidth',1.2); grid on;
196 xlabel('Tiempo [s]'); ylabel('Velocidad [°/s]');
197 legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');

```

```

198 title('Velocidades articulares');
199
200 figure;
201 plot(t(1:end-2), rad2deg(ddQ), 'LineWidth',1.2); grid on;
202 xlabel('Tiempo [s]'); ylabel('Aceleración [°/s^2]');
203 legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
204 title('Aceleraciones articulares');
205
206 %% =====
207 % Gráfica de velocidad y aceleración cartesiana del efecto
208 % =====
209 figure;
210 plot(t(1:end-1), v_cart, 'LineWidth',1.2); grid on;
211 xlabel('Tiempo [s]'); ylabel('Velocidad lineal [mm/s]');
212 legend('v_x','v_y','v_z','Location','bestoutside');
213 title('Velocidad cartesiana del efecto');
214
215 a_cart = diff(v_cart) / dt; % aceleración cartesiana (mm/s^2), ...
216 % tamaño (n-2) x 3
217 figure;
218 plot(t(1:end-2), a_cart, 'LineWidth',1.2); grid on;
219 xlabel('Tiempo [s]'); ylabel('Aceleración lineal [mm/s^2]');
220 legend('a_x','a_y','a_z','Location','bestoutside');
221 title('Aceleración cartesiana del efecto');
222
223 %% =====
224 % Trayectoria del efecto final (posición)
225 % =====
226 figure;
227 plot3(P(:,1),P(:,2),P(:,3),'b-','LineWidth',2); hold on;
228 plot3(P(1,1),P(1,2),P(1,3),'go','MarkerFaceColor','g'); text(P(1,1),P(1,2),P(1,3),'Inicio',...
229 'VerticalAlignment','bottom');
230 plot3(P(end,1),P(end,2),P(end,3),'ro','MarkerFaceColor','r'); text(P(end,1),P(end,2),P(end,3),...
231 'Final','VerticalAlignment','top');
232 grid on; xlabel('X [mm]'); ylabel('Y [mm]'); zlabel('Z [mm]');
233 title('Trayectoria del efecto final');
234
235 %% =====
236 % Funciones auxiliares
237 % =====
238 function q_sol = newtonIK(p_d, q_seed, p_fun, J_fun, a2_val, T2_offset_val, T3_offset_val, q_min, ...
239 q_max, q_mid, K, tol, maxIter)
240 % -NewtonRaphson SIN clamping (solo penalización suave de límites)
241 qk = q_seed(:);
242 for it = 1:maxIter
243     p_now = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), a2_val, T2_offset_val, T3_offset_val);
244     J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), a2_val, T2_offset_val, T3_offset_val);
245     e = p_d - p_now;
246     if norm(e) < tol, break; end
247     gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);
248     dq = pinv(J_now) * e - K * gradH;
249     qk = qk + dq;
250 end
251 q_sol = qk;
252 end
253
254 function Q = quinticTraj(q0, qf, T, t)
255 % Polinomio quíntico con vel y acc cero en extremos
256 n = numel(t); Q = zeros(n, numel(q0));
257 for i=1:numel(q0)
258     a0 = q0(i); a1 = 0; a2 = 0;
259     a3 = 10*(qf(i)-q0(i))/T^3;
260     a4 = -15*(qf(i)-q0(i))/T^4;
261     a5 = 6*(qf(i)-q0(i))/T^5;
262     Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
263 end
264 end
265
266 function [Q_out, t_out, T_scaled] = iptp_scale_time(Q, t, vel_max, acc_max)
267 % Escalado temporal para respetar límites de velocidad y aceleración
268 dt = t(2) - t(1);
269 dQ = diff(Q)/dt;
270 ddQ = diff(dQ)/dt;
271 scale_v = max(max(abs(dQ) ./ vel_max));
272 scale_a = max(max(abs(ddQ) ./ acc_max));

```

```

273 scale = max([1, scale_v, sqrt(scale_a)]);
274 if scale > 1
275     T_scaled = t(end) * scale;
276     t_out = linspace(0, T_scaled, numel(t));
277     % Recalcular la quintic con el nuevo T para mantener vel/acc cero en extremos
278     Q_out = zeros(size(Q));
279     q0 = Q(1,:); qf = Q(end,:);
280     for i=1:size(Q,2)
281         a0 = q0(i); a1 = 0; a2 = 0;
282         a3 = 10*(qf(i)-q0(i))/T_scaled^3;
283         a4 = -15*(qf(i)-q0(i))/T_scaled^4;
284         a5 = 6*(qf(i)-q0(i))/T_scaled^5;
285         Q_out(:,i) = a0 + a1*t_out + a2*t_out.^2 + a3*t_out.^3 + a4*t_out.^4 + a5*t_out.^5;
286     end
287 else
288     Q_out = Q; t_out = t; T_scaled = t(end);
289 end
290 end

```

Código 11: Control cinemático completo.

```

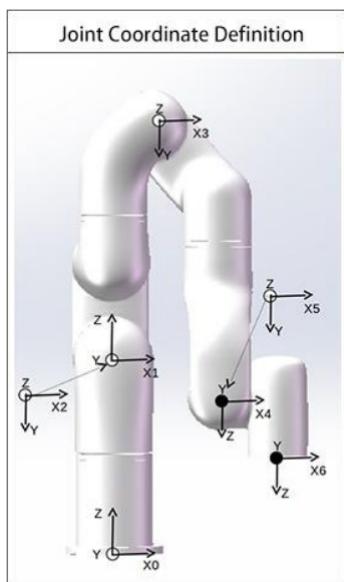
1 Objetivo a 455.5 mm, dentro del alcance máximo 700.0 mm -> Realizable
2
3 === Estado ===
4 Duración planificada (IPTP): 5.00 s
5 \n==== Articulaciones iniciales (matemático) [deg] ====
6      0      0      0      0      0      0
7
8 === Articulaciones finales (matemático) [deg] ===
9      21.6552   -4.7464   -25.0741   -7.6934   -0.3000      0
10
11 === Articulaciones iniciales (físico) [deg] ===
12      0   -79.3500    79.3500      0      0      0
13
14 === Articulaciones finales (físico) [deg] ===
15      21.6552   -84.0963    54.2758   -7.6934   -0.3000      0
16
17 === Velocidades articulares iniciales [deg/s] ===
18      1.0e-03 *
19
20      0.4820   -0.1057   -0.5581   -0.1712   -0.0067      0
21
22 === Velocidades articulares finales [deg/s] ===
23      1.0e-03 *
24
25      0.4820   -0.1057   -0.5581   -0.1712   -0.0067      0
26
27 === Aceleraciones articulares iniciales [deg/s^2] ===
28      0.1718   -0.0377   -0.1989   -0.0610   -0.0024      0
29
30 === Aceleraciones articulares finales [deg/s^2] ===
31      -0.1718    0.0377    0.1989    0.0610    0.0024

```

Código 12: Salida del control cinemático completo.

## 4. Estudio dinámico

El fabricante proporciona las características físicas de cada eslabón, en particular la masa y la posición de su centro de masa (*CoM*), figura 18, respecto de los sistemas de referencia de la figura 17 con la tabla de parámetros 5. También proporcionan en los modelos de simulación los momentos de inercia [Spong2006] de cada eslabón, expuestos en el código 13. El modelo del robot con el que se trabaja corresponde al modelo 1 del robot, que se alimenta a través de la AC Control Box.



UFACTORY xArm 6-Model 1

Dynamics	Mass (Kg)	Center of Mass (mm)
Link1	2.177	[0.15,27.24,-13.57]
Link2	2.011	[36.7,-220.9,33.56]
Link3	1.725	[69.77,113.5,11.6]
Link4	1.211	[-0.2,20.0,-26.0]
Link5	1.206	[63.87,29.3,3.5]
Link6	0.17	[0,-6.77,-10.98]

Figura 17: Sistemas de referencia para el estudio dinámico. Fuente: [ufactory\_xarm\_manual].

Figura 18: Parámetros de la dinámica del robot. Fuente: [ufactory\_xarm\_manual].

$i$	$\theta_i$ (°)	$d_i$ (mm)	$a_{i-1}$ (mm)	$\alpha_{i-1}$ (°)
1	$\theta_1$	267	0	0
2	$\theta_2$	0	0	-90
3	$\theta_3 + (T2\_offset = -79,34995)$	$d_3$	$a_2 = 289,48866$	0
4	$\theta_4 + (T2\_offset = 79,34995)$	342,5	77,5	-90
5	0	0	0	90
6	0	97	76	-90

Cuadro 5: Parámetros DH modificados del UFactory xArm6 para el estudio dinámico.

```

1 xarm6_type6_HT_BR.yaml: https://github.com/xArm-Developer/xarm_ros2/blob/jazzy/xarm_description/config/
2   link_inertial/xarm6_type6_HT_BR.yaml
3
4 link1:
5   inertia:
6     ixx: 0.005433
7     ixy: 9.864e-06
8     ixz: -2.68e-05

```

```

9     iyy: 0.004684
10    iyz: -0.000826936
11    izz: 0.0031118
12    mass: 2.177
13    origin:
14      x: 0.00015
15      y: 0.02724
16      z: -0.01357
17 link2:
18   inertia:
19     ixx: 0.0271342
20     ixy: 0.004736
21     ixz: 0.00068673
22     iyy: 0.0053854
23     iyz: -0.0047834
24     izz: 0.0262093
25   mass: 2.011
26   origin:
27     x: 0.0367
28     y: -0.22088
29     z: 0.03356
30 link3:
31   inertia:
32     ixx: 0.006085
33     ixy: -0.0015
34     ixz: 0.0009558
35     iyy: 0.0036652
36     iyz: 0.0018091
37     izz: 0.0057045
38   mass: 1.725
39   origin:
40     x: 0.06977
41     y: 0.1135
42     z: 0.01163
43 link4:
44   inertia:
45     ixx: 0.0046981
46     ixy: -6.486e-06
47     ixz: -1.404e-05
48     iyy: 0.0042541
49     iyz: -0.0002877
50     izz: 0.00123664
51   mass: 1.211
52   origin:
53     x: -0.0002
54     y: 0.02
55     z: -0.026
56 link5:
57   inertia:
58     ixx: 0.0013483
59     ixy: -0.00042677
60     ixz: 0.00028758
61     iyy: 0.00175694
62     iyz: 0.0001244
63     izz: 0.002207
64   mass: 1.206
65   origin:
66     x: 0.06387
67     y: 0.02928
68     z: 0.0035
69 link6:
70   inertia:
71     ixx: 9.3e-05
72     ixy: -0.0
73     ixz: -0.0
74     iyy: 5.87e-05
75     iyz: -3.6e-06
76     izz: 0.000132
77   mass: 0.17
78   origin:
79     x: 0
80     y: -0.00677
81     z: -0.01098

```

Código 13: Momentos de inercia del los eslabones del robot en estudio.

## 4.1. Ecuaciones de movimiento

Las ecuaciones de movimiento de un robot manipulador son las expresiones matemáticas que describen cómo las fuerzas y pares aplicados en las articulaciones producen el movimiento del sistema, teniendo en cuenta su masa, inercias, geometría y la acción de la gravedad. Se tendrá una ecuación de movimiento por cada eslabón y finalmente se obtiene el vector de pares articulares  $\tau = [\tau_1, \tau_2, \dots, \tau_6] N \cdot m$ . La expresión general de la ecuación de movimiento es la siguiente:

$$M(q) \ddot{q} + C(q, \dot{q}) \dot{q} + g(q) = \tau \quad (5)$$

donde:

- $M(q)$  es la matriz de inercia (simétrica definida positiva).
- $C(q, \dot{q}) \dot{q}$  agrupa términos centrífugos y de Coriolis.
- $g(q)$  es el vector de gravedad.
- $\tau$  son los pares actuados en las articulaciones.

En [siciliano2009robotics] se extiende la ecuación de movimiento para añadir las fricciones viscosa  $F_v \dot{q}$  y de Coulomb  $F_c \operatorname{sgn}(\dot{q})$  y las fuerzas externas, donde  $J(q)$  es el jacobiano geométrico del efecto final y  $f_{\text{ext}}$  las fuerzas externas aplicadas. Esta versión extendida se presenta a continuación:

$$M(q) \ddot{q} + C(q, \dot{q}) \dot{q} + g(q) + F_v \dot{q} + F_c \operatorname{sgn}(\dot{q}) = \tau + J^\top(q) f_{\text{ext}}$$

Como el fabricante no proporciona los datos de fricción y se va a trabajar en simulación, únicamente se estudia la expresión general.

Las ecuaciones de movimiento pueden calcularse mediante el balance de energías Lagrangiano [siciliano2009robotics] o por el método recursivo de Newton–Euler [barrientos2007fundamentos], siendo este último el más habitual por ser computacionalmente más eficiente.

### 4.1.1. Enfoque energético (Lagrangiano)

En términos energéticos, los componentes pueden obtenerse a partir del lagrangiano  $\mathcal{L} = T - V$ , donde  $T$  es la energía cinética total del manipulador y  $V$  la energía potencial gravitatoria.

La energía cinética se calcula como:

$$T = \frac{1}{2} \sum_{i=1}^n (m_i \dot{p}_{c_i}^\top \dot{p}_{c_i} + \omega_i^\top I_{c_i} \omega_i),$$

donde  $m_i$  es la masa del eslabón  $i$ ,  $\dot{p}_{c_i}$  la velocidad del centro de masa del eslabón  $i$ ,  $\omega_i$  la velocidad angular del eslabón  $i$  e  $I_{c_i}$  su tensor de inercia respecto al centro de masa.

La energía potencial gravitatoria se expresa como:

$$V = \sum_{i=1}^n m_i g^\top p_{c_i},$$

donde  $g$  es el vector de aceleración de la gravedad y  $p_{c_i}$  la posición del centro de masa del eslabón  $i$ .

A partir de estas expresiones, se obtienen los componentes de la ecuación de movimiento (ec. 5):

$$M_{ij}(q) = \frac{\partial^2 T}{\partial \dot{q}_i \partial \dot{q}_j}, \quad g_i(q) = \frac{\partial V}{\partial q_i},$$

y los elementos de  $C(q, \dot{q})$  mediante los símbolos de Christoffel:

$$c_{ijk} = \frac{1}{2} \left( \frac{\partial M_{ij}}{\partial q_k} + \frac{\partial M_{ik}}{\partial q_j} - \frac{\partial M_{jk}}{\partial q_i} \right), \quad [C(q, \dot{q})]_{ij} = \sum_{k=1}^n c_{ijk} \dot{q}_k.$$

#### 4.1.2. Método recursivo de Newton–Euler

Mientras que el enfoque Lagrangiano produce explícitamente  $M(q)$ ,  $C(q, \dot{q})$ ,  $g(q)$ , Newton–Euler es más eficiente computacionalmente para evaluar  $\tau$  dado  $\{q, \dot{q}, \ddot{q}\}$  y es el más usado en simulación y control en tiempo real.

El método recursivo de Newton–Euler calcula directamente los pares articulares  $\tau_i$  a partir de  $\{q, \dot{q}, \ddot{q}\}$  (variables articulares) y los parámetros dinámicos  $\{m_i, r_{c_i}, I_{c_i}\}$  (masas, centros de masa e inercias), mediante un barrido hacia adelante (cinemática y aceleraciones) y otro hacia atrás (acumulación de fuerzas y momentos).

**Barrido hacia adelante** Condiciones en la base:

$$\omega_0 = 0, \quad \alpha_0 = 0, \quad a_0 = -g.$$

Para  $i = 1, \dots, n$ , con  $R_i$  y  $p_i$  de las transformaciones homogéneas y  $z_i$  el eje articular:

$$\omega_i = R_i^\top \omega_{i-1} + \dot{q}_i z_i,$$

$$\alpha_i = R_i^\top \alpha_{i-1} + \ddot{q}_i z_i + \dot{q}_i z_i \times (R_i^\top \omega_{i-1}),$$

$$a_i = R_i^\top (a_{i-1} + \alpha_{i-1} \times p_i + \omega_{i-1} \times (\omega_{i-1} \times p_i)),$$

$$a_{c_i} = a_i + \alpha_i \times r_{c_i} + \omega_i \times (\omega_i \times r_{c_i}).$$

#### Fuerzas y momentos en el centro de masa

$$F_i = m_i a_{c_i}, \quad N_i = I_{c_i} \alpha_i + \omega_i \times (I_{c_i} \omega_i).$$

**Barrido hacia atrás** Acumulación desde el efecto hasta la base. Para  $i = n, \dots, 1$ :

$$f_i = F_i + \sum_{k=i+1}^n R_k f_k,$$

$$n_i = N_i + r_{c_i} \times F_i + \sum_{k=i+1}^n (R_k n_k + r_{ik} \times (R_k f_k)),$$

donde  $r_{ik}$  es el vector del origen  $i$  al origen  $k$ .

**Proyección sobre el eje articular** Para articulación rotacional:

$$\tau_i = n_i^\top z_i,$$

y opcionalmente se añaden fricciones [siciliano2009robotics]:

$$\tau_i \leftarrow \tau_i + f_{v,i} \dot{q}_i + f_{c,i} \operatorname{sgn}(\dot{q}_i).$$

#### 4.1.3. Ejemplo de una iteración para el eslabón 3 del *UFactory xArm6*

A modo ilustrativo, se muestra una iteración completa del método recursivo de Newton–Euler para el eslabón  $i = 3$  del robot en estudio, utilizando la tabla 5 y las matrices  $A_i$  calculadas en el apartado 3.1.2.

#### Transformaciones homogéneas y descomposición en rotación y traslación

$$d_1 = 267, \quad a_2 = 289,48866, \quad a_3 = 77,5, \quad d_4 = 342,5, \quad a_5 = 76, \quad d_6 = 97.$$

Para el cálculo en el barrido hacia adelante del eslabón 3, necesitamos  ${}^0A_1$ ,  ${}^1A_2$  y  ${}^2A_3$ . Denotando  $C_i = \cos(\cdot)$  y  $S_i = \sin(\cdot)$ :

$$A_1(q_1) = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 & 0 \\ \sin q_1 & \cos q_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow {}^0R_1 = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 \\ \sin q_1 & \cos q_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad {}^0p_1 = \begin{bmatrix} 0 \\ 0 \\ d_1 \end{bmatrix}.$$

$$A_2(\theta_2 = q_2 + T2_{\text{offset}}) = \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin \theta_2 & -\cos \theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \Rightarrow {}^1R_2 = \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 \\ 0 & 0 & 1 \\ -\sin \theta_2 & -\cos \theta_2 & 0 \end{bmatrix}, \quad {}^1p_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

$$A_3(\theta_3 = q_3 + T3_{\text{offset}}) = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 & a_2 \\ \sin \theta_3 & \cos \theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \Rightarrow {}^2R_3 = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad {}^2p_3 = \begin{bmatrix} a_2 \\ 0 \\ 0 \end{bmatrix}.$$

La pose del marco 3 respecto a la base se obtiene con:

$${}^0T_3 = {}^0A_1 {}^1A_2 {}^2A_3, \quad {}^0R_3 = {}^0R_1 {}^1R_2 {}^2R_3, \quad {}^0p_3 = {}^0p_1 + {}^0R_1 {}^1p_2 + {}^0R_1 {}^1R_2 {}^2p_3.$$

$${}^0R_3 = {}^0R_1 {}^1R_2 {}^2R_3 = \begin{bmatrix} \cos q_1 & -\sin q_1 & 0 \\ \sin q_1 & \cos q_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta_2 & -\sin \theta_2 & 0 \\ 0 & 0 & 1 \\ -\sin \theta_2 & -\cos \theta_2 & 0 \end{bmatrix} \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$${}^0R_3 = {}^0R_2 {}^2R_3 = \begin{bmatrix} \cos q_1 \cos \theta_2 & -\cos q_1 \sin \theta_2 & -\sin q_1 \\ \sin q_1 \cos \theta_2 & -\sin q_1 \sin \theta_2 & \cos q_1 \\ -\sin \theta_2 & -\cos \theta_2 & 0 \end{bmatrix} \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 \\ \sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$${}^0R_3 = \begin{bmatrix} \cos q_1(\cos \theta_2 \cos \theta_3 - \sin \theta_2 \sin \theta_3) & \cos q_1(-\cos \theta_2 \sin \theta_3 - \sin \theta_2 \cos \theta_3) & -\sin q_1 \\ \sin q_1(\cos \theta_2 \cos \theta_3 - \sin \theta_2 \sin \theta_3) & \sin q_1(-\cos \theta_2 \sin \theta_3 - \sin \theta_2 \cos \theta_3) & \cos q_1 \\ -(\sin \theta_2 \cos \theta_3 + \cos \theta_2 \sin \theta_3) & \sin \theta_2 \sin \theta_3 - \cos \theta_2 \cos \theta_3 & 0 \end{bmatrix}.$$

$${}^0p_3 = {}^0p_1 + {}^0R_1 {}^1p_2 + {}^0R_1 {}^1R_2 {}^2p_3 = \begin{bmatrix} 0 \\ 0 \\ 267 \end{bmatrix} + \underbrace{{}^0R_1 \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{=0} + ({}^0R_1 {}^1R_2) \begin{bmatrix} 289,48866 \\ 0 \\ 0 \end{bmatrix}.$$

Como  ${}^0R_1 {}^1R_2$  tiene primera columna  $[\cos q_1 \cos \theta_2, \sin q_1 \cos \theta_2, -\sin \theta_2]^\top$ , se obtiene:

$$\begin{aligned} {}^0p_3 &= \begin{bmatrix} 289,48866 \cos q_1 \cos \theta_2 \\ 289,48866 \sin q_1 \cos \theta_2 \\ 267 - 289,48866 \sin \theta_2 \end{bmatrix}. \\ {}^0T_3 &= \begin{bmatrix} {}^0R_3 & {}^0p_3 \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}. \end{aligned}$$

**Barrido hacia adelante** Tomamos el eje articular local  $z_3 = [0, 0, 1]^T$  y convertimos las longitudes a unidades del Sistema Internacional:  $d_1 = 0,267$  m,  $a_2 = 0,28948866$  m.

### Velocidad angular

$$\omega_3 = R_3^\top \omega_2 + \dot{q}_3 z_3 = \begin{bmatrix} \cos \theta_3 & \sin \theta_3 & 0 \\ -\sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{2x} \\ \omega_{2y} \\ \omega_{2z} \end{bmatrix} + \dot{q}_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

$$\Rightarrow \begin{cases} \omega_{3x} = \cos \theta_3 \omega_{2x} + \sin \theta_3 \omega_{2y}, \\ \omega_{3y} = -\sin \theta_3 \omega_{2x} + \cos \theta_3 \omega_{2y}, \\ \omega_{3z} = \omega_{2z} + \dot{q}_3. \end{cases}$$

### Aceleración angular

$$\alpha_3 = R_3^\top \alpha_2 + \ddot{q}_3 z_3 + \dot{q}_3 (z_3 \times (R_3^\top \omega_2)),$$

con

$$\begin{aligned} R_3^\top \omega_2 &= \begin{bmatrix} \cos \theta_3 \omega_{2x} + \sin \theta_3 \omega_{2y} \\ -\sin \theta_3 \omega_{2x} + \cos \theta_3 \omega_{2y} \\ \omega_{2z} \end{bmatrix}. \\ \Rightarrow \begin{cases} \alpha_{3x} = \cos \theta_3 \alpha_{2x} + \sin \theta_3 \alpha_{2y} - \dot{q}_3 (-\sin \theta_3 \omega_{2x} + \cos \theta_3 \omega_{2y}), \\ \alpha_{3y} = -\sin \theta_3 \alpha_{2x} + \cos \theta_3 \alpha_{2y} + \dot{q}_3 (\cos \theta_3 \omega_{2x} + \sin \theta_3 \omega_{2y}), \\ \alpha_{3z} = \alpha_{2z} + \ddot{q}_3. \end{cases} \end{aligned}$$

**Aceleración lineal del origen del marco 3** Con  $p_3 = [0,28948866, 0, 0]$  m se obtiene

$$a_3 = R_3^\top (a_2 + \alpha_2 \times p_3 + \omega_2 \times (\omega_2 \times p_3)).$$

Expandiendo:

$$\begin{aligned} \alpha_2 \times p_3 &= \begin{bmatrix} 0 \\ \alpha_{2z} 0,28948866 \\ -\alpha_{2y} 0,28948866 \end{bmatrix}, \quad \omega_2 \times (\omega_2 \times p_3) = \begin{bmatrix} -0,28948866(\omega_{2y}^2 + \omega_{2z}^2) \\ 0,28948866 \omega_{2x} \omega_{2y} \\ 0,28948866 \omega_{2x} \omega_{2z} \end{bmatrix}. \\ \Rightarrow a_3 &= R_3^\top \begin{bmatrix} a_{2x} - 0,28948866(\omega_{2y}^2 + \omega_{2z}^2) \\ a_{2y} + 0,28948866 \alpha_{2z} + 0,28948866 \omega_{2x} \omega_{2y} \\ a_{2z} - 0,28948866 \alpha_{2y} + 0,28948866 \omega_{2x} \omega_{2z} \end{bmatrix}. \end{aligned}$$

**Aceleración del centro de masa** Finalmente, con  $r_{c3} = [r_x, r_y, r_z]^\top$  (coordenadas del centro de masa del eslabón 3, figura 18):

$$a_{c3} = a_3 + \alpha_3 \times r_{c3} + \omega_3 \times (\omega_3 \times r_{c3}).$$

**Fuerzas y momentos en el centro de masa** Con la masa  $m_3$  y el tensor de inercia en el CoM  $I_{c3}$ :

$$F_3 = m_3 a_{c3}, \quad N_3 = I_{c3} \alpha_3 + \omega_3 \times (I_{c3} \omega_3).$$

**Barrido hacia atrás** Contribuciones de los eslabones posteriores ( $k = 4, 5, 6$ ) ya acumuladas y expresadas en el marco 3 mediante:

$$\begin{aligned} f_3 &= F_3 + \sum_{k=4}^6 R_k f_k, \\ n_3 &= N_3 + r_{c3} \times F_3 + \sum_{k=4}^6 (R_k n_k + r_{3k} \times (R_k f_k)), \end{aligned}$$

donde:

- $R_k$ : matriz de rotación que transforma vectores desde el marco del eslabón  $k$  al marco del eslabón 3. Matemáticamente,  $R_k = {}^3R_k = ({}^0R_3)^\top {}^0R_k$ .
- $r_{3k}$ : vector de posición desde el origen del marco 3 hasta el origen del marco  $k$ , expresado en el marco 3. Se obtiene como  $r_{3k} = ({}^0R_3)^\top ({}^0p_k - {}^0p_3)$ .

Cada término  $R_k f_k$  representa la fuerza del eslabón  $k$  expresada en el marco 3, y  $r_{3k} \times (R_k f_k)$  es el momento adicional debido a la aplicación de esa fuerza a una distancia  $r_{3k}$  del origen del marco 3.

**Proyección sobre el eje articular** El par en la articulación 3 se obtiene proyectando el momento sobre su eje:

$$\tau_3 = n_3^\top z_3,$$

y, si se modela fricción:

$$\tau_3 \leftarrow \tau_3 + f_{v,3} \dot{q}_3 + f_{c,3} \operatorname{sgn}(\dot{q}_3).$$

## 4.2. Control dinámico

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

1. Calcular la cinemática directa y diferencial.
2. Calcular la cinemática inversa y generar trayectorias articulares.
3. Propagar velocidades y aceleraciones mediante el barrido hacia adelante.
4. Calcular fuerzas y momentos en cada eslabón.
5. Acumular las contribuciones mediante el barrido hacia atrás.
6. Proyectar los momentos sobre los ejes articulares para obtener los pares  $\tau_i$ .

En este sentido, se ha programado en *Matlab*, junto con la *Robotics Toolbox* de Peter Corke, el control dinámico completo del robot *UFactory xArm6*. El programa utiliza los parámetros dinámicos del modelo (masas, centros de masa e inercias) extraídos del URDF oficial, y permite calcular tanto la dinámica inversa (torques articulares  $\tau$  dados  $q, \dot{q}, \ddot{q}$ ) como la dinámica directa (aceleraciones articulares  $\ddot{q}$  dadas  $q, \dot{q}, \tau$ ).

Al inicio, el programa solicita la posición articular de origen y la posición cartesiana a alcanzar, junto con 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. A partir de estas trayectorias articulares, se aplica el algoritmo de Newton–Euler para obtener los torques requeridos en cada articulación.

Las imágenes ??, ?? y ?? muestran los torques articulares calculados, las aceleraciones articulares obtenidas por dinámica directa y la comparación entre las aceleraciones de referencia y las reconstruidas. Los códigos ?? y ?? recogen la implementación en *Matlab* y la salida por terminal.

