



Trabajo final sobre la materia: *UFactory xArm6*

Máster en Ingeniería de Sistemas y Control

Asignatura: Robótica Industrial

Curso: 2025/2026

Alumno: Javier Arambarri Calvo

24 de diciembre de 2025

Autoría

El autor de este documento es Javier Arambarri Calvo, alumno del Máster en Ingeniería de Sistemas y Control matriculado por la Universidad Complutense de Madrid en el curso académico 2025-2026 e investigador contratado en el grupo de investigación ViSens de la Escuela de Ingeniería de Bilbao [ehu_visens].

Para que así conste, se firma digitalmente este documento.

Propiedad

El autor y el grupo ViSens son titulares de los derechos de propiedad intelectual asociados a este trabajo.

Nota

Las imágenes que no indican fuente son propias.

Índice

1. Introducción	8
1.1. Objetivo y motivación del proyecto	8
1.2. Metodología	8
2. UFactory xArm 6	9
2.1. Estructura mecánica: eslabones y articulaciones	9
2.2. Actuadores	10
2.3. Sensores integrados y opcionales	11
2.4. Efecto	12
2.5. Sistema de control	13
2.6. Software	14
3. Estudio cinemático	15
3.1. Cinemática directa	15
3.1.1. Parámetros de Denavit-Hartenberg	15
3.1.2. Parámetros modificados de Denavit-Hartenberg	17
3.2. Cinemática inversa	21
3.2.1. Pseudoinversa del Jacobiano y Newton-Raphson	23
3.2.2. Cálculo de cinemática inversa para el <i>UFactory xArm6</i>	25
3.3. Cinemática diferencial y Jacobiano	26
3.4. Generación de trayectorias	27
3.4.1. Trayectorias lineales en espacio cartesiano	27
3.4.2. Trayectorias circulares en espacio cartesiano	27
3.4.3. Trayectorias polinómicas en espacio articular	27
3.4.4. Trayectorias <i>spline (multi-waypoint)</i>	28
3.5. Cálculo de la trayectoria en <i>Matlab</i> con la <i>toolbox</i> de Peter Corke	29
3.6. Control cinemático	30
4. Estudio dinámico	34
4.1. Ecuaciones de movimiento	35
4.1.1. Enfoque energético (Lagrangiano)	35
4.1.2. Método recursivo de Newton-Euler	36
4.1.3. Ejemplo de una iteración para el eslabón 3 del <i>UFactory xArm6</i>	37
4.2. Control dinámico	39
5. Escalado y parametrización temporal de trayectorias	41
5.1. <i>IPTP: Iterative Parabolic Time Parameterization</i>	41
5.2. <i>ISP: Iterative Spline Parameterization</i>	42
5.3. <i>TOTG: Time-Optimal Trajectory Generation</i>	42
5.4. <i>TOPP-RA: Time-Optimal Path Parameterization via Reachability Analysis</i>	43
5.5. Implementación propia 1: escalado temporal uniforme y reinterpolación quíntica	44
5.6. Implementación propia 2: <i>retiming</i> dinámico por par articular	44
6. Controlador PID	46

7. Programación con MoveIt2-ROS2-Gazebo	49
7.1. Algoritmos de planificación y generación de trayectorias en MoveIt2	49
7.1.1. Planificadores geométricos (<i>OMPL</i>)	49
7.1.2. Interpolación cartesiana	50
7.1.3. Algoritmos de parametrización temporal	50
7.1.4. Ejecución de la trayectoria	51
7.2. Programación básica con la GUI de MoveIt2 en RVIZ	51
7.3. Programación básica con un nodo en C++	55
8. Evaluación y análisis de costes	57
8.1. Costes directos	57
8.2. Costes indirectos	57
8.3. Relación coste–beneficio	58
Anexo: código	59
Estudio cinemático	59
Estudio dinámico	72
Controlador	75
Programación básica de MoveIt2 con un nodo en C++	83
Referencias adicionales al material del Campus Virtual	87

Índice de figuras

1.	<i>UFactory xArm 6</i> . Fuente: [10].	9
2.	<i>Reductor armónico</i> . Fuente: [11].	10
3.	Ejemplo de elemento terminal. Fuente: [4].	12
4.	<i>Control box analógico</i> . Fuente: [4].	13
5.	<i>Control box digital</i> . Fuente: [4].	13
6.	<i>xArm Studio</i> . Fuente: [4].	14
7.	Parámetros clásicos Denavit-Hartenberg. Fuente: [9].	15
8.	Parámetros modificados Denavit-Hartenberg. Fuente: [9].	18
9.	Modelado del <i>UFactory xArm6</i> en <i>Matlab</i> con parámetros DH modificados.	20
10.	Trayectoria articular realizable.	29
11.	Trayectoria realizable del efector.	30
12.	Trayectoria articular realizable.	31
13.	Trayectoria realizable del efector.	31
14.	Velocidad cartesiana del efector.	32
15.	Velocidades articulares.	32
16.	Aceleración cartesiana del efector.	33
17.	Aceleraciones articulares.	33
18.	Sistemas de referencia para el estudio dinámico. Fuente: [9].	34
19.	Parámetros de la dinámica del robot. Fuente: [9].	34
20.	Par articular en la articulación 1.	47
21.	Par articular en la articulación 2.	47
22.	Par articular en la articulación 3.	47
23.	Par articular en la articulación 4.	47
24.	Par articular en la articulación 5.	47
25.	Par articular en la articulación 6.	47
26.	Trayectoria articular.	47
27.	Trayectoria cartesiana.	47
28.	Velocidades articulares.	48
29.	Aceleraciones articulares.	48
30.	Par articular total para las seis articulaciones durante el seguimiento.	48
31.	Visualizaciones real, inicial y final del robot en RVIZ.	51
32.	Planificación y ejecución de la trayectoria.	52
33.	Colisiones activadas.	53
34.	Panel de <i>joints</i>	53
35.	Movimiento de la trayectoria planificada.	54
36.	Visualización de puntos de paso.	54
37.	Trayectoria cartesiana.	55
38.	Movimiento del robot en Gazebo.	56

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>	11
3.	Resumen de tipo de señal, comunicación y alimentación de los elementos terminales del efecto del <i>xArm 6</i>	12
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.	34

Índice de códigos

1.	Código Matlab para la cinemática directa con parámetros DH clásicos.	59
2.	Salida numérica de la cinemática directa DH clásica.	59
3.	Código Matlab para la cinemática directa con parámetros DH modificados.	60
4.	Salida numérica de la cinemática directa DH modificada.	60
5.	Código Matlab para obtener las ecuaciones de posición y orientación con parámetros DH modificados.	60
6.	Cálculo de la cinemática inversa del robot <i>UFactory xArm6</i>	61
7.	Salida del cálculo de la cinemática inversa del robot <i>UFactory xArm6</i>	63
8.	Cálculo de la cinemática diferencial y Jacobiano mediante parámetros DH modificados.	63
9.	Generación de trayectoria articular realizable.	65
10.	Salida de la generación de trayectoria articular realizable.	67
11.	Control cinemático completo.	67
12.	Salida del control cinemático completo.	71
13.	Momentos de inercia del los eslabones del robot en estudio.	72
14.	Cálculo de pares de los actuadores.	73
15.	Salida del cálculo de pares de los actuadores.	74
16.	Programa principal.	75
17.	Modelado.	77
18.	Planificador cinemático.	78
19.	Replanificador dinámico.	80
20.	Controlador.	81
21.	Nodo C++ para controlar el robot con MoveIt2.	83

1. Introducción

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

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

1.1. Objetivo y motivación del proyecto

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

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

1.2. Metodología

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

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

2. UFactory xArm 6

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

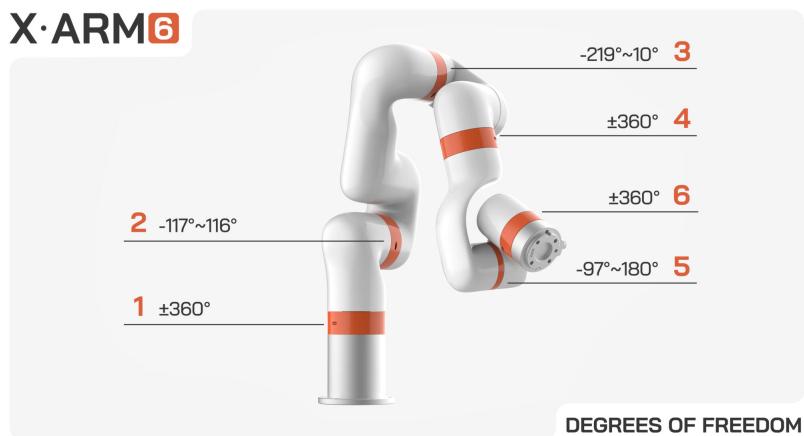


Figura 1: *UFactory xArm 6*. Fuente: [10].

2.1. Estructura mecánica: eslabones y articulaciones

La estructura mecánica del robot se compone de un cuerpo ligero fabricado en aluminio y fibra de carbono, con un peso total cercano a los 12.5 kg, lo que facilita su instalación en entornos de laboratorio o docencia. Los distintos eslabones del brazo están formados por piezas huecas unidos mediante 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 $\pm 360^\circ$, mientras que las articulaciones intermedias permiten rotaciones de hasta $\pm 180^\circ$, y las articulaciones de muñeca alcanzan rangos de $\pm 360^\circ$, 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 [11], figura 2, se basan en la deformación elástica controlada de un componente flexible para transmitir el movimiento. Este principio permite alcanzar relaciones de reducción muy elevadas en un volumen compacto, con una rigidez torsional superior a la de otros sistemas de engranajes. En el *xArm 6*, su empleo asegura un movimiento suave y preciso, además de minimizar el retroceso mecánico (*backlash*), lo que resulta fundamental para aplicaciones de ensamblaje, manipulación de piezas y tareas de investigación que requieren gran exactitud. Otra ventaja de los reductores armónicos es su capacidad para soportar cargas elevadas en relación con su tamaño, lo que contribuye a que el robot pueda mantener una carga útil de hasta 5 kg sin comprometer la precisión. Asimismo, su diseño compacto permite integrar el actuador y el reductor dentro de cada articulación, reduciendo el volumen total del brazo y facilitando su instalación en espacios reducidos.

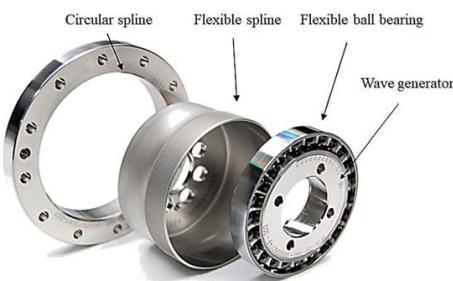


Figura 2: Reductor armónico. Fuente: [11].

Parámetro	Joint 1	Joint 2	Joint 3	Joint 4	Joint 5	Joint 6
Rango de trabajo	$\pm 360^\circ$	$-117^\circ - 116^\circ$	$-219^\circ - 10^\circ$	$\pm 360^\circ$	$-97^\circ - 180^\circ$	$\pm 360^\circ$
Par máximo	No proporcionado por el fabricante.					
Vel. y acel. máxima	$180^\circ/\text{s}, 1500^\circ/\text{s}^2$					
Carga útil máxima	5 kg					
Potencia nominal	150 W					
Repetibilidad	± 0.1 mm					

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

2.3. Sensores integrados y opcionales

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

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

El fabricante también ofrece una gama de sensores opcionales que amplían las capacidades del robot. Entre ellos destaca el sensor de fuerza o torque de seis ejes, instalado en la brida del manipulador, capaz de medir fuerzas y momentos en las tres direcciones espaciales (F_x , F_y , F_z , M_x , M_y , M_z). 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* [13], recomendada por el fabricante.



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

La brida del robot cumple con el patrón de taladros indicado en la ISO 9409-1:2004 [14], lo que facilita la integración de herramientas de terceros y asegura la compatibilidad con dispositivos de medida como el sensor de fuerza o torque de seis ejes. Este sensor se instala directamente en el 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	Control box / gripper
Gripper de vacío	Analógica / Digital	Interna	Control box / gripper
Bio gripper	Digital	Interna	Control box / gripper

Cuadro 3: Resumen de tipo de señal, comunicación y alimentación de los elementos terminales del 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: [4].



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

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

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

2.6. Software

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

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



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

3. Estudio cinemático

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

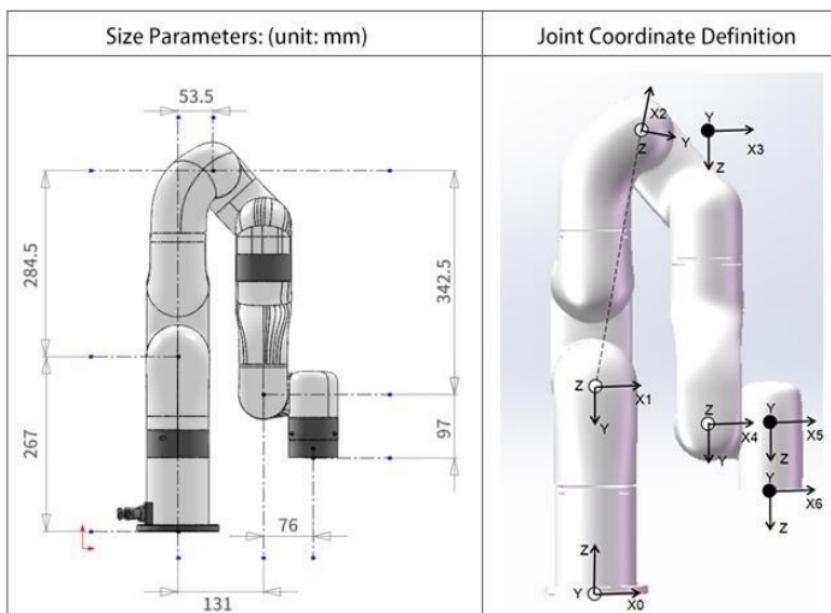
3.1. Cinemática directa

3.1.1. Parámetros de Denavit-Hartenberg

En la figura 9 se presentan los parámetros clásicos de Denavit-Hartenberg [12] proporcionados por el fabricante, con

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

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



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: [9].

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 θ de la articulación y se suma directamente a θ_i .

Además, a partir del análisis de la figura 9 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 [16] y la matriz de transformación homogénea [12] asociada a dichas transformaciones se presentan en las ecuaciones 1 y 2:

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

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

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

$$\begin{aligned} A_1 &= \begin{bmatrix} C(q_1) & 0 & -S(q_1) & 0 \\ S(q_1) & 0 & C(q_1) & 0 \\ 0 & -1 & 0 & 267 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ A_2 &= \begin{bmatrix} C(q_2 + \text{offset}_2) & -S(q_2 + \text{offset}_2) & 0 & a_2 C(q_2 + \text{offset}_2) \\ S(q_2 + \text{offset}_2) & C(q_2 + \text{offset}_2) & 0 & a_2 S(q_2 + \text{offset}_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_2 = 289,48866 \\ A_3 &= \begin{bmatrix} C(q_3 + \text{offset}_3) & 0 & -S(q_3 + \text{offset}_3) & a_3 C(q_3 + \text{offset}_3) \\ S(q_3 + \text{offset}_3) & 0 & C(q_3 + \text{offset}_3) & a_3 S(q_3 + \text{offset}_3) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_3 = 77,5 \\ A_4 &= \begin{bmatrix} C(q_4) & 0 & S(q_4) & 0 \\ S(q_4) & 0 & -C(q_4) & 0 \\ 0 & 1 & 0 & 342,5 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad A_5 = \begin{bmatrix} C(q_5) & 0 & -S(q_5) & a_5 C(q_5) \\ S(q_5) & 0 & C(q_5) & a_5 S(q_5) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad a_5 = 76 \\ A_6 &= \begin{bmatrix} C(q_6) & -S(q_6) & 0 & 0 \\ S(q_6) & C(q_6) & 0 & 0 \\ 0 & 0 & 1 & 97 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Siendo la transformación total:

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

Podemos calcular y comprobar el resultado en *Matlab* [2] y la *toolbox* de Peter Corke [17], códigos 1 y 2.

3.1.2. Parámetros modificados de Denavit-Hartenberg

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

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

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

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

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

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

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 [16]. Además, facilita los cálculos recursivos de cinemática, Jacobianos y dinámica, evitando las ambigüedades de colocación presentes en el método estándar y es la notación más clara y transparente para el análisis mecánico.

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

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

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

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

Size Parameters: (unit: mm)			Joint Coordinate Definition		
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: [9].

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:

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

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

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

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

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

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

Siendo la transformación total:

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

Podemos calcular y comprobar el resultado en *Matlab* [2] y la *toolbox* de Peter Corke [17], códigos 3 y 4. En la figura 9 se presenta el modelado del robot en *Matlab*.

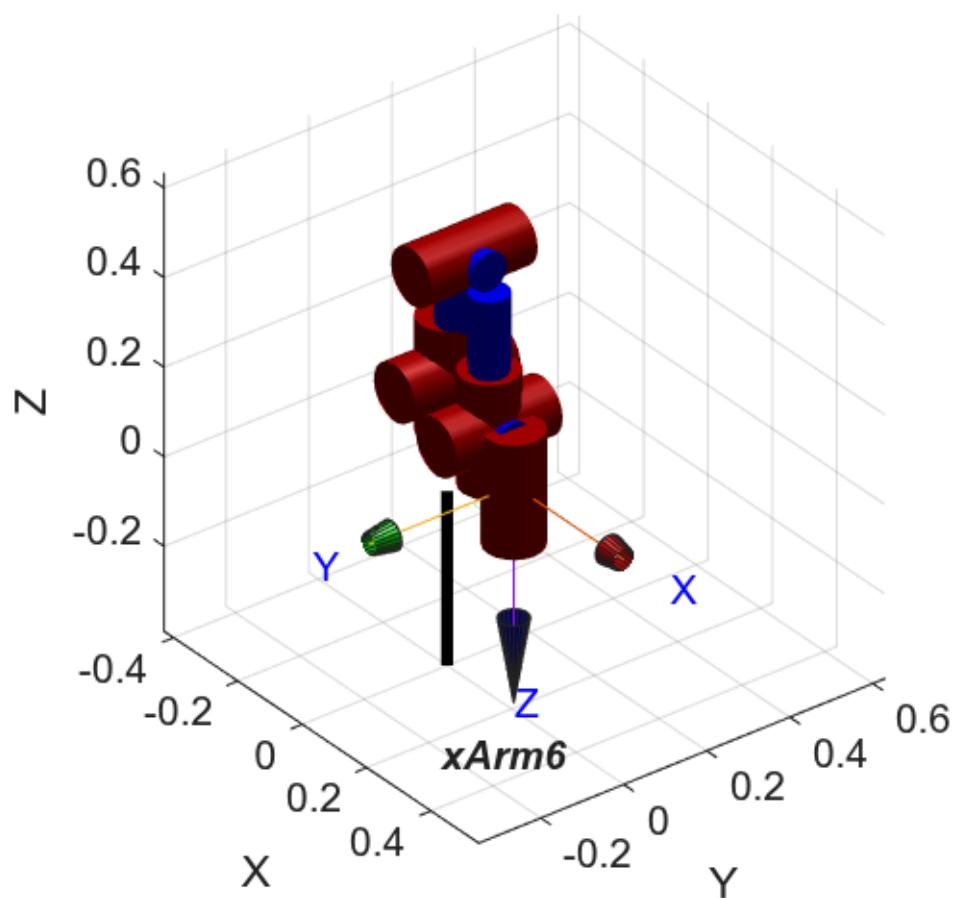


Figura 9: Modelado del *UFactory xArm6* en *Matlab* con parámetros DH modificados.

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 [12]. Sin embargo, esta propiedad no se cumple en el robot *UFactory xArm6*, ya que la posición y orientación depende de todas las articulaciones. Por tanto, no existe separación natural entre el brazo y la muñeca y la cinemática inversa se vuelve mucho más compleja porque no existe en forma cerrada.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Estas ecuaciones se han obtenido con la ayuda de *Matlab* [2] y la *toolbox* de Peter Corke [17] mediante el código 5.

3.2.1. Pseudoinversa del Jacobiano y Newton–Raphson

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

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

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

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

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 efector final. Definiendo el error cartesiano como

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

y linealizando la cinemática directa alrededor de la configuración actual,

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

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

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

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

Ejemplo sencillo de Newton–Raphson a mano

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

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

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

$$x_d = 4.$$

El problema de cinemática inversa se escribe entonces como

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

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

$$f(q_0) = f(1) = 1^2 = 1,$$

$$x_d - f(q_0) = 4 - 1 = 3,$$

$$J(q_0) = 2q_0 = 2 \cdot 1 = 2, \quad J^\dagger(q_0) = \frac{1}{2}.$$

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

$$f(q_1) = f(2,5) = (2,5)^2 = 6,25,$$

$$x_d - f(q_1) = 4 - 6,25 = -2,25,$$

$$J(q_1) = 2q_1 = 2 \cdot 2,5 = 5, \quad J^\dagger(q_1) = \frac{1}{5}.$$

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

$$f(q_2) = f(2,05) = (2,05)^2 \approx 4,2025,$$

$$x_d - f(q_2) \approx 4 - 4,2025 = -0,2025,$$

$$J(q_2) = 2q_2 = 2 \cdot 2,05 = 4,10, \quad J^\dagger(q_2) = \frac{1}{4,10}.$$

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 el *UFactory xArm6*

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

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

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

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

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

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

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

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

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

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

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

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

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

En los códigos 6 y 7 se expone la porción de código de *Matlab* [2] y la *Robotics Toolbox* de Peter Corke [17] con la que se ha programado la cinemática inversa para el *UFactory xArm6*.

3.3. Cinemática diferencial y Jacobiano

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

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

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

A partir de la transformación homogénea total

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

se pueden calcular las matrices de rotación y vectores de posición necesarios para construir el Jacobiano y para calcular las velocidades lineales y angulares de las articulaciones y del efector. Dado que la matriz de transformación resultante es una expresión grande, su manejo manual simbólico resulta muy tedioso, por lo que se ha calculado con ayuda de *Matlab* [2] y la *Robotics Toolbox* de Peter Corke [17] en el código 8.

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 [8]. Este planificador permite definir trayectorias en espacio articular y cartesiano, que son posteriormente ejecutadas en tiempo real a través de *ros2_control* [25].

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

3.4.1. Trayectorias lineales en espacio cartesiano

el 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. Trayectorias polinómicas en espacio articular

Cuando se especifican directamente configuraciones articulares, *Movelit* 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 de velocidad y aceleración 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:

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

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

Y el polinomio quíntico:

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

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

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

3.4.4. 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 de velocidad y aceleración 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 realizables. Es importante tener en cuenta que previo al cálculo de esta trayectoria es necesario obtener las posiciones articulares finales deseadas mediante cinemática inversa.

Las imágenes 10 y 11 muestran la trayectoria calculada mediante el código 9 para las posiciones indicadas en el código 10.

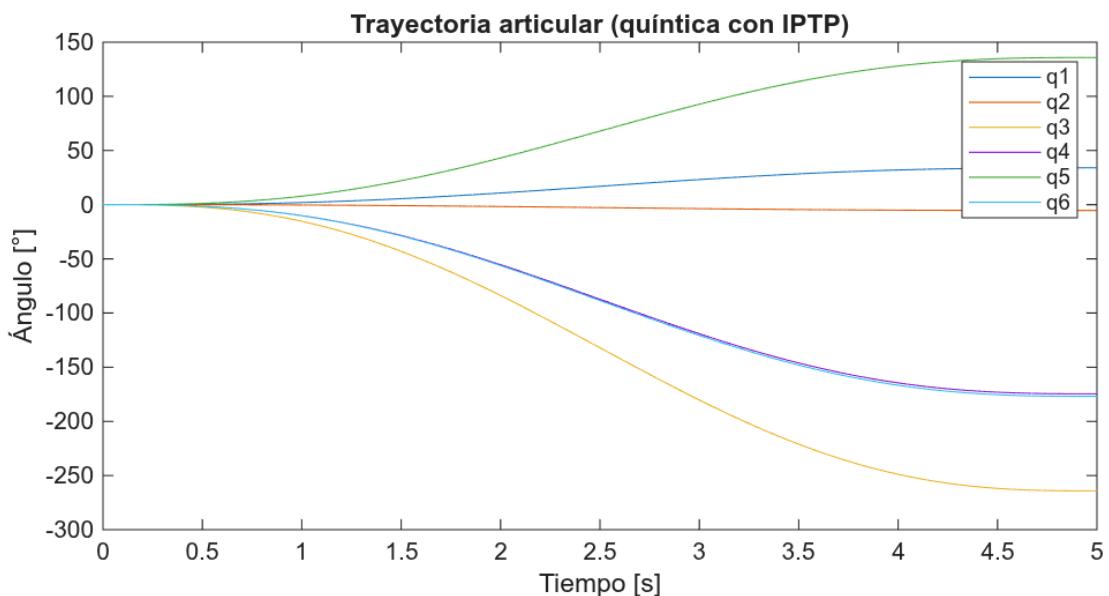


Figura 10: Trayectoria articular realizable.

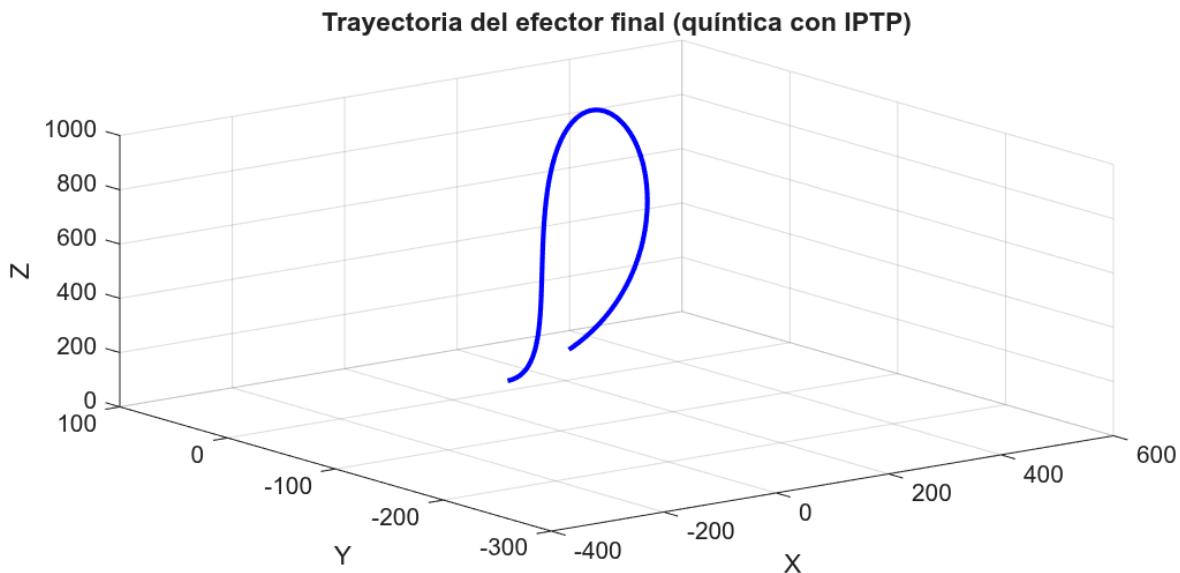


Figura 11: Trayectoria realizable del efecto.

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 12, 13, 14, 15, 16 y 17 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.

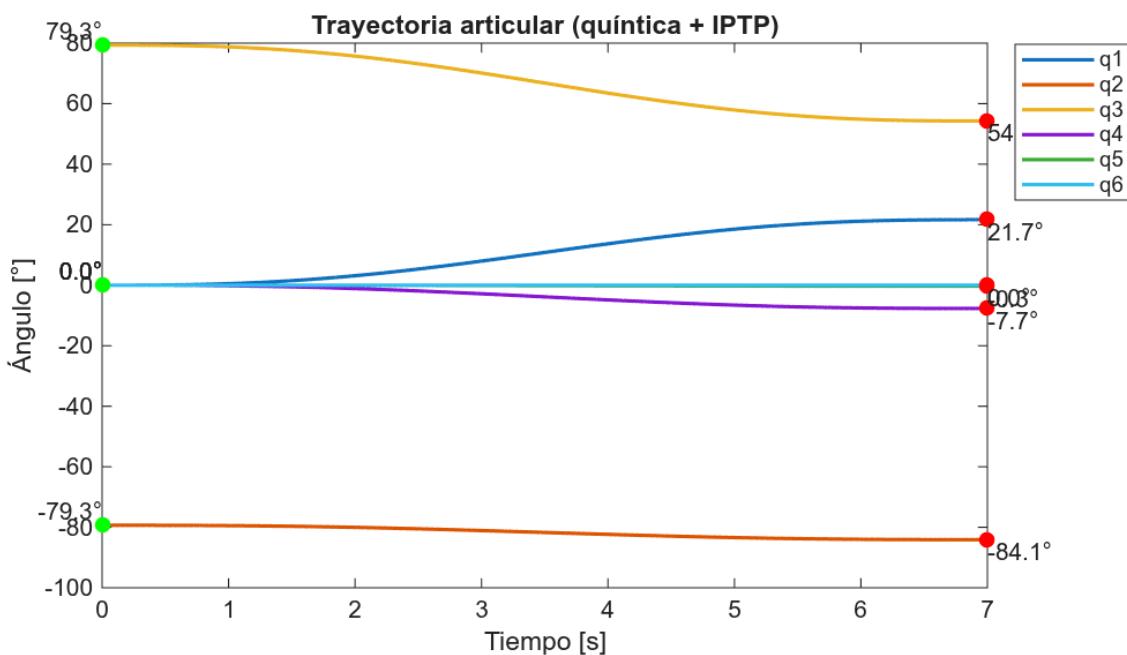


Figura 12: Trayectoria articular realizable.

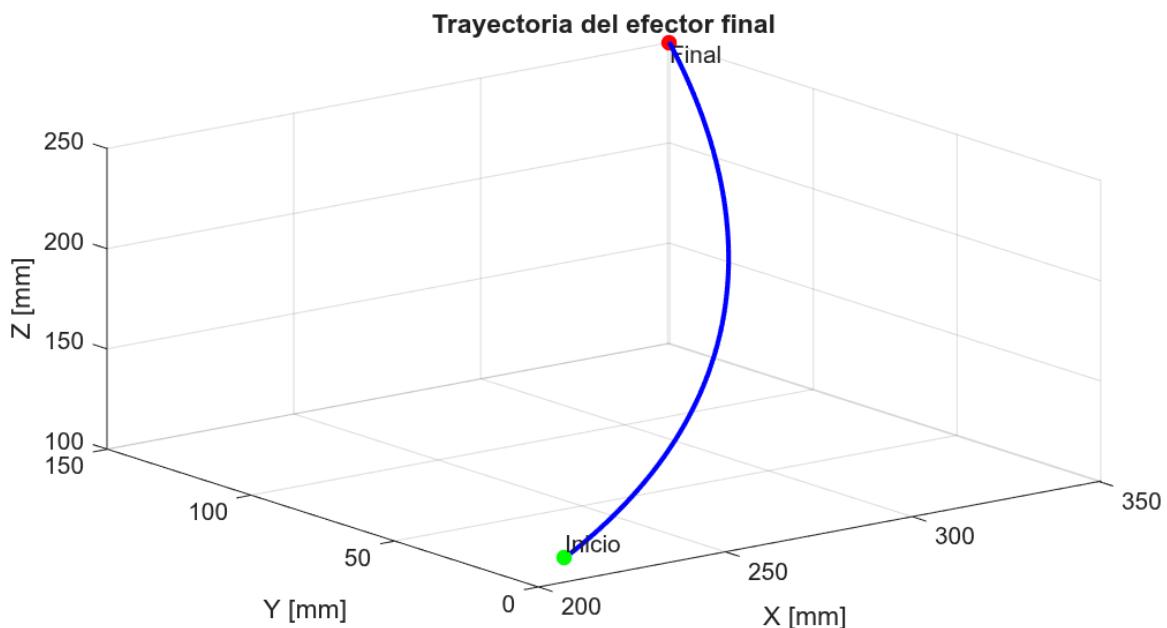


Figura 13: Trayectoria realizable del efecto.

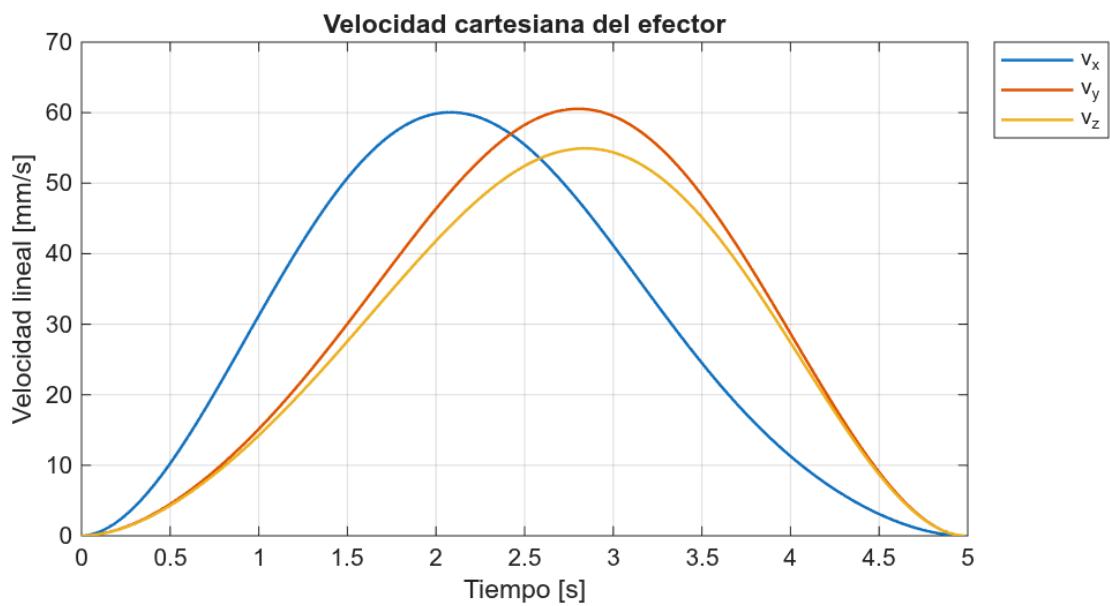


Figura 14: Velocidad cartesiana del efecto.

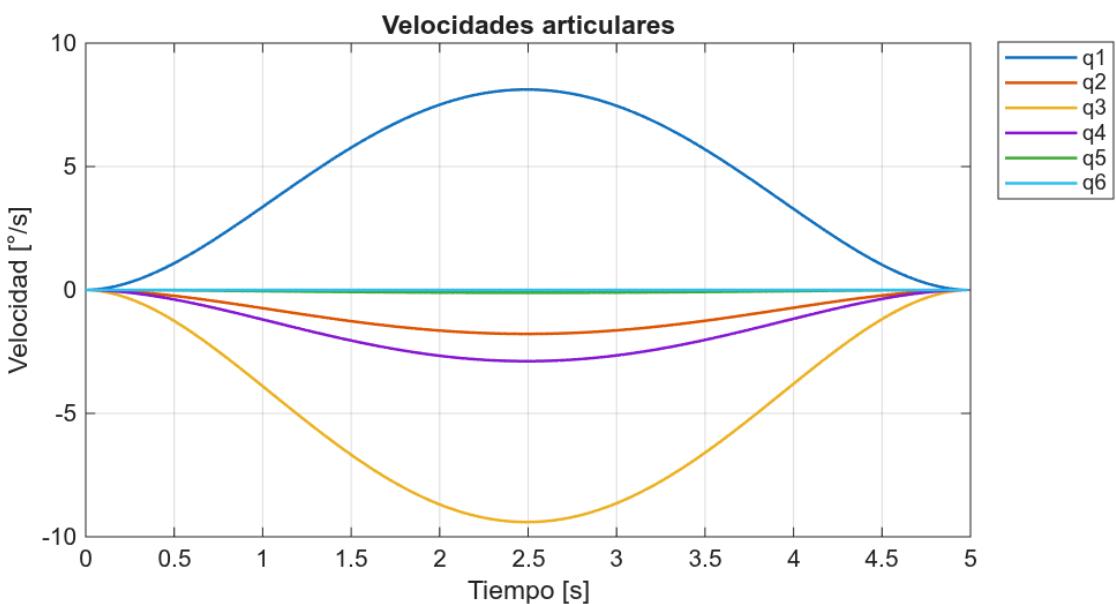


Figura 15: Velocidades articulares.

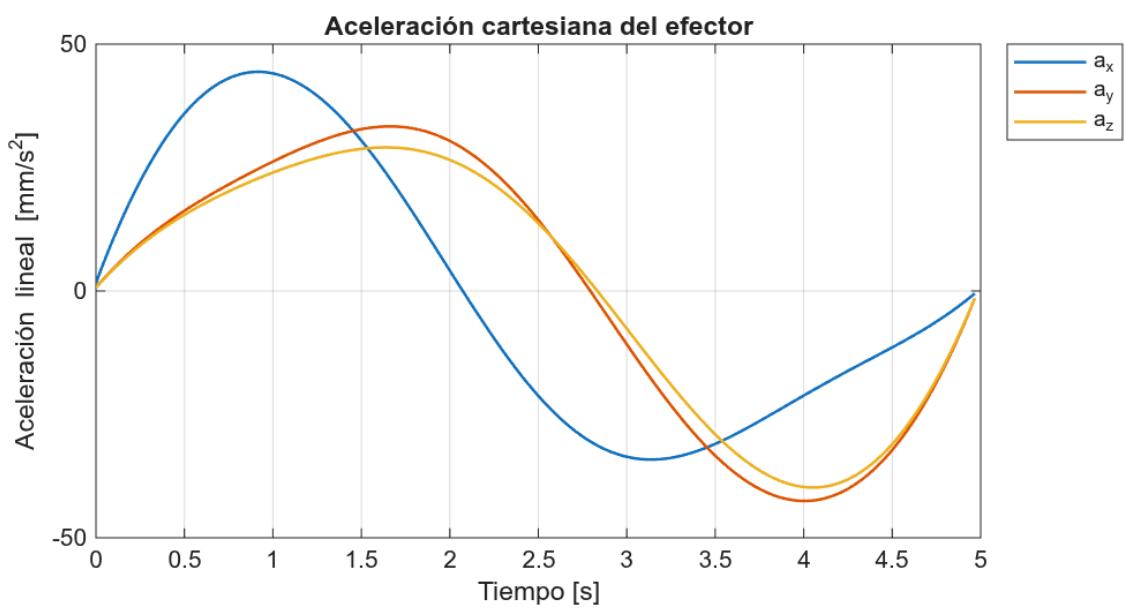


Figura 16: Aceleración cartesiana del efecto.

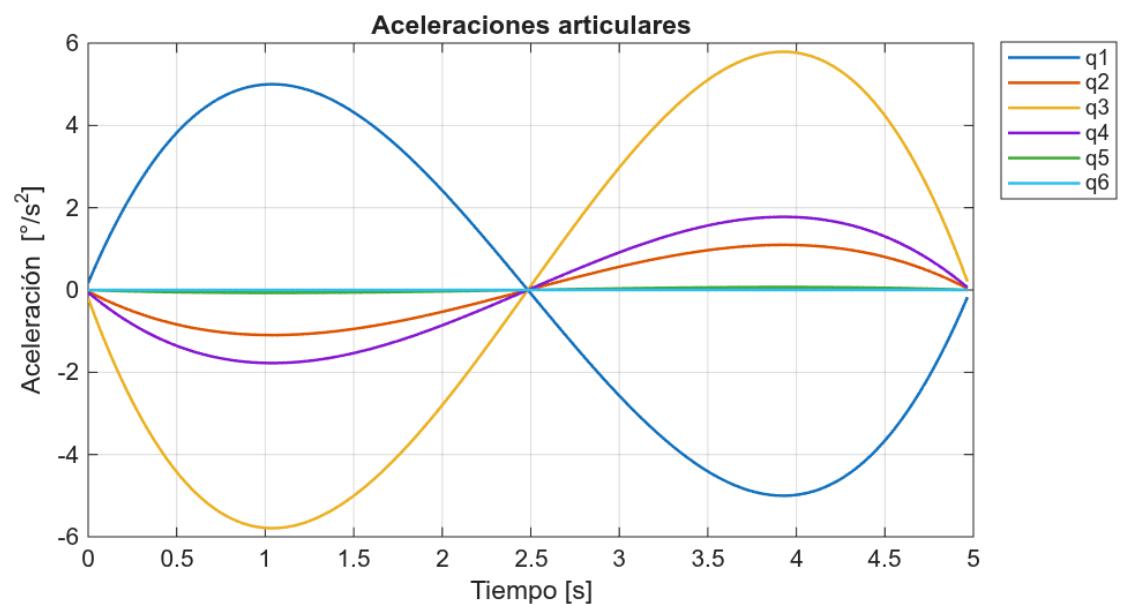
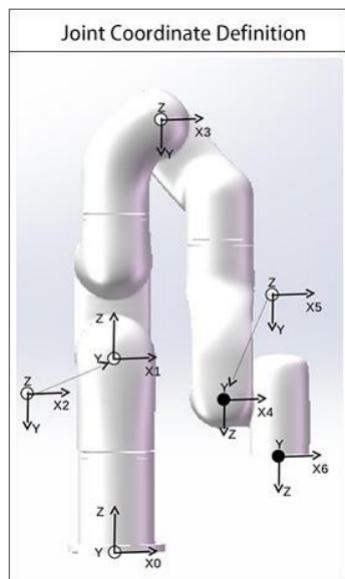


Figura 17: Aceleraciones articulares.

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 19, respecto de los sistemas de referencia de la figura 18 con la tabla de parámetros 5. También proporcionan en los modelos de simulación los momentos de inercia [26] 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 18: Sistemas de referencia para el estudio dinámico. Fuente: [9].

Figura 19: Parámetros de la dinámica del robot. Fuente: [9].

i	θ_i (°)	d_i (mm)	a_{i-1} (mm)	α_{i-1} (°)
1	θ_1	267	0	0
2	θ_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.

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.
- τ son los pares actuados en las articulaciones.

En [21] 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_{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 [21] o por el método recursivo de Newton–Euler [12], 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 , ω_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 τ 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 τ_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 [21]:

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

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:

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

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

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

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

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

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

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

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

$$\begin{aligned} \omega_3 &= R_3^\top \omega_2 + \dot{q}_3 z_3 = \begin{bmatrix} \cos \theta_3 & \sin \theta_3 & 0 \\ -\sin \theta_3 & \cos \theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \omega_{2x} \\ \omega_{2y} \\ \omega_{2z} \end{bmatrix} + \dot{q}_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \\ \Rightarrow & \begin{cases} \omega_{3x} = \cos \theta_3 \omega_{2x} + \sin \theta_3 \omega_{2y}, \\ \omega_{3y} = -\sin \theta_3 \omega_{2x} + \cos \theta_3 \omega_{2y}, \\ \omega_{3z} = \omega_{2z} + \dot{q}_3. \end{cases} \end{aligned}$$

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 $r_{c_3} = [r_x, r_y, r_z]^\top$ (coordenadas del centro de masa del eslabón 3, figura 19):

$$a_{c_3} = a_3 + \alpha_3 \times r_{c_3} + \omega_3 \times (\omega_3 \times r_{c_3}).$$

Fuerzas y momentos en el centro de masa Con la masa m_3 y el tensor de inercia en el CoM I_{c_3} :

$$F_3 = m_3 a_{c_3}, \quad N_3 = I_{c_3} \alpha_3 + \omega_3 \times (I_{c_3} \omega_3).$$

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

$$f_3 = F_3 + \sum_{k=4}^6 R_k f_k,$$

$$n_3 = N_3 + r_{c_3} \times F_3 + \sum_{k=4}^6 (R_k n_k + r_{3k} \times (R_k f_k)),$$

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. Calcular los pares necesarios en cada articulación.
4. Recalcular la trayectoria o escalar temporalmente si los pares superan los valores máximos.

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 τ dados q, \dot{q}, \ddot{q}) como la dinámica directa (aceleraciones articulares \ddot{q} dadas q, \dot{q}, τ).

También es necesario introducir los siguientes parámetros dinámicos de cada link para que *Matlab* pueda calcular el par necesario:

- J_m : inercia del rotor del motor reflejada en el eje de la articulación. Este término representa la resistencia del motor a cambios de velocidad angular. En muchos modelos simplificados se toma $J_m = 0$ al ser despreciable frente a la inercia de los eslabones.
- G : relación de transmisión entre el motor y el eje de salida. Un valor elevado de G corresponde a reductores armónicos o planetarios que multiplican el par y reducen la velocidad. En simulaciones básicas se suele fijar $G = 1$.
- B : coeficiente de fricción viscosa en la articulación, modelado como un par proporcional a la velocidad angular ($\tau_B = B \cdot \dot{q}$). Representa pérdidas por rozamiento fluido o lubricación interna.
- T_c : par de fricción de tipo Coulomb, independiente de la velocidad y dependiente únicamente del sentido de giro. Se define como un vector $T_c = [T_c^+, T_c^-]$ que especifica el par de fricción en movimiento positivo y negativo respectivamente.

Estos parámetros permiten extender el modelo dinámico clásico añadiendo los términos de fricción y transmisión [21], de modo que la ecuación completa se expresa como:

$$\tau = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) + B\dot{q} + T_c.$$

El fabricante no aporta estos valores, y aunque se pueden obtener experimentalmente, se va a utilizar la ecuación dinámica simple al igual que se emplea en el simulador *Gazebo*, que se implementar en el código 14. que calcula los pares de los motores necesarios a aplicar. En la salida del programa, código 15, se observa que el par del tercer motor es superior a su par máximo: $|\tau_3(t)| > \tau_{3,\text{máx.}}$.

Esto se debe principalmente a que únicamente se han introducido valores arbitrarios de velocidades, aceleraciones y posiciones finales sin calcular correctamente la trayectoria como en el control cinemático. Por tanto, en este caso la trayectoria planificada no es físicamente realizable en el tiempo nominal, pues el motor no puede generar el par necesario. La solución consiste en aplicar un escalado temporal de la trayectoria, aumentando el tiempo total de ejecución y reduciendo así las aceleraciones articulares. De este modo, el par requerido en cada articulación también disminuye y se mantiene dentro de los límites físicos del motor. La articulación que alcanza antes su límite de par es la que determina el nuevo tiempo mínimo necesario para completar el movimiento, y el resto de articulaciones deben ajustarse para avanzar de manera coordinada, garantizando que todas lleguen al objetivo simultáneamente.

5. Escalado y parametrización temporal de trayectorias

En esta sección se describen los principales métodos de parametrización temporal de trayectorias bajo restricciones cinemáticas y dinámicas. Estas técnicas permiten corregir discontinuidades y respetar los límites de aceleración, velocidad o par máximos del robot en estudio. Se incluyen los algoritmos clásicos de la literatura (*IPTP*, *ISP*, *TOTG*, *TOPP-RA*) y las implementaciones desarrolladas en este trabajo, basadas en escalado temporal uniforme y *retiming* dinámico por par articular. Es importante resaltar que aunque en las gráficas expuestas se indique *IPTP*, la implementación corresponde a la expuesta en este apartado.

5.1. *IPTP: Iterative Parabolic Time Parameterization*

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* [27, 28] 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:

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

5.2. ISP: Iterative Spline Parameterization

El método *ISP* [29, 30] ajusta iterativamente la parametrización temporal de una trayectoria *spline* para garantizar el cumplimiento de límites de velocidad y aceleración.

Cuando se detecta una violación en un instante de tiempo t_k , por ejemplo:

$$|\dot{q}(t_k)| > \dot{q}_{\text{máx}},$$

se incrementa localmente el tiempo asociado a dicho nodo mediante un factor $\alpha_k > 1$:

$$t_k \leftarrow t_k + \Delta t_k, \quad \Delta t_k = (\alpha_k - 1)(t_k - t_{k-1}).$$

Tras modificar las muestras temporales, se reconstruye la spline:

$$q(t) = \text{spline}(q_0, q_1, \dots, q_N; t_0, t_1, \dots, t_N).$$

El procedimiento es iterativo y se repite hasta converger o respetar los límites en estudio. Este enfoque es robusto y adecuado para trayectorias suaves de orden elevado, aunque no produce soluciones estrictamente óptimas en tiempo.

5.3. TOTG: Time-Optimal Trajectory Generation

El objetivo de *TOTG* [27, 31] es encontrar la parametrización temporal $s(t)$ de una trayectoria geométrica $q(s)$, con $s \in [0, 1]$, que minimiza el tiempo total cumpliendo límites de velocidad y aceleración articulares.

La velocidad articular viene dada por $\dot{q}(t) = q'(s) \dot{s}$ y la aceleración por $\ddot{q}(t) = q''(s) \ddot{s} + q'(s) \dot{s}^2$.

Las restricciones articulares se expresan como:

$$|\dot{q}_i(t)| = |q'_i(s) \dot{s}| \leq \dot{q}_{i,\text{máx}},$$

$$|\ddot{q}_i(t)| = |q'_i(s) \ddot{s} + q''_i(s) \dot{s}^2| \leq \ddot{q}_{i,\max}.$$

De estas desigualdades se obtiene un rango admisible para la aceleración escalar:

$$\ddot{s} \in [\ddot{s}_{\min}(s, \dot{s}), \ddot{s}_{\max}(s, \dot{s})].$$

El algoritmo clásico se basa en dos barridos:

1. Barrido hacia adelante (aceleración máxima). Se integra $\ddot{s} = \ddot{s}_{\max}(s, \dot{s})$ hasta que alguna restricción se activa.
2. Barrido hacia atrás (deceleración máxima). Desde el final, se integra $\ddot{s} = \ddot{s}_{\min}(s, \dot{s})$ para garantizar que la trayectoria puede detenerse cumpliendo límites.

Finalmente, el perfil óptimo de velocidad $\dot{s}(s)$ es la envolvente superior de ambos barridos, por lo que el tiempo total óptimo se obtiene integrando:

$$T_{\text{opt}} = \int_0^1 \frac{1}{\dot{s}(s)} ds.$$

TOTG produce una solución óptima en tiempo, pero requiere resolver ecuaciones diferenciales y manejar cambios de régimen dinámico, lo que dificulta su implementación.

5.4. *TOPP-RA: Time-Optimal Path Parameterization via Reachability Analysis*

TOPP-RA [32] reformula la parametrización temporal como un problema de alcanzabilidad en un sistema lineal con restricciones. Sea una trayectoria geométrica $q(s)$ discretizada en N puntos, $s_0 < s_1 < \dots < s_N$, la velocidad escalar se discretiza como $\dot{s}_k = v_k$ y la aceleración escalar se approxima mediante diferencias finitas:

$$\ddot{s}_k \approx \frac{v_{k+1}^2 - v_k^2}{2\Delta s_k}, \quad \Delta s_k = s_{k+1} - s_k.$$

Las restricciones articulares se expresan como un conjunto de desigualdades lineales en v_k^2 y v_{k+1}^2 :

$$A_k v_{k+1}^2 + B_k v_k^2 \leq C_k,$$

donde A_k, B_k, C_k dependen de $q'(s_k), q''(s_k), \dot{q}_{\max}$ y \ddot{q}_{\max} .

El algoritmo se basa en dos barridos:

1. Barrido hacia adelante. Se calcula el conjunto alcanzable de velocidades:

$$v_{k+1}^2 \in \mathcal{R}_{k+1}(v_k^2),$$

resolviendo las restricciones lineales, donde $\mathcal{R}_k(\cdot)$ representa el conjunto alcanzable de velocidades, modelado como un intervalo linealmente restringido $\mathcal{V}_k = [v_k^{\min}, v_k^{\max}]$.

2. Barrido hacia atrás. Se impone que la trayectoria pueda desacelerar hasta el final:

$$v_k^2 \in \mathcal{R}_k(v_{k+1}^2).$$

Por último, el perfil óptimo se obtiene seleccionando en cada punto el valor máximo admisible:

$$v_k = \sqrt{v_k^{2,\max}}.$$

El tiempo total se calcula como:

$$T_{\text{RA}} = \sum_{k=0}^{N-1} \frac{\Delta s_k}{v_k}.$$

TOPP-RA es numéricamente estable, eficiente y robusto, y actualmente uno de los métodos más utilizados para *retiming* bajo restricciones cinemáticas.

5.5. Implementación propia 1: escalado temporal uniforme y reinterpolación quíntica

Se aplica un escalado temporal global para garantizar que la trayectoria quíntica cumple los límites de velocidad y aceleración. El procedimiento es:

1. Se genera una trayectoria quíntica base entre q_0 y q_f .
2. Se calculan numéricamente $\dot{q}(t)$ y $\ddot{q}(t)$.
3. Se obtiene el factor mínimo de escalado:

$$s = \max \left(1, \frac{\max |\dot{q}|}{\dot{q}_{\max}}, \sqrt{\frac{\max |\ddot{q}|}{\ddot{q}_{\max}}} \right).$$

4. Se genera una nueva quíntica con tiempo ampliado $T_{\text{scaled}} = s T_0$.

Este método es una forma simplificada de *TOTG*, pero no constituye un *retiming* diferencial ni una solución óptima. Su ventaja es la simplicidad y suavidad de la trayectoria resultante. Su implementación se encuentra en el planificador cinemático del código ??.

5.6. Implementación propia 2: *retiming* dinámico por par articular

Además del escalado cinemático, se implementa un escalado temporal dinámico basado en el modelo dinámico del robot. El objetivo es garantizar que el par articular requerido cumple:

$$\forall i \in \{1, \dots, 6\}, \forall t \in [0, T], \quad |\tau_i(t)| \leq \tau_{i,\max}.$$

Dado que el par escala con el tiempo según la ecuación básica de movimiento (ec. 5) y que bajo un escalado temporal $t \rightarrow kt$ se cumple:

$$\dot{q} \rightarrow \frac{\dot{q}}{k}, \quad \ddot{q} \rightarrow \frac{\ddot{q}}{k^2},$$

el par disminuye al aumentar k . Esto permite plantear una búsqueda binaria sobre k :

1. Evaluar el par requerido con $k = 1$.
2. Si cumple límites, no se modifica la trayectoria.

3. Si no cumple, se realiza una búsqueda binaria en $k \in [1, k_{\max}]$.
4. Para cada k , se recalculan \dot{q} , \ddot{q} y el par.
5. Se selecciona el menor k que satisface todas las restricciones.

Este procedimiento garantiza una trayectoria dinámicamente factible sin alterar la forma espacial de la trayectoria, únicamente su parametrización temporal. Es sencillo, robusto y adecuado para trayectorias suaves generadas mediante interpolación polinómica. Su implementación se presenta en el replanificador cinemático del código ??.

Tal como se indica en la tabla 1, el fabricante no proporciona los valores reales de par máximo de cada articulación, ya que el controlador comercial opera como una “caja negra”. Por este motivo, en la práctica no es posible aplicar un retiming basado en límites reales del actuador. Aun así, se ha implementado el algoritmo con fines didácticos, utilizando valores estimados de τ_{\max} para ilustrar el funcionamiento del método. Si se desea observar el efecto del escalado, basta con modificar el vector de pares máximos en el código.

6. Controlador PID

Una vez generada una trayectoria articular dinámicamente factible, es necesario disponer de un controlador capaz de seguirla con precisión. En este trabajo se ha implementado un controlador PID por articulación [12], combinado con el modelo dinámico del robot para generar el par de control adecuado en cada instante.

El objetivo del controlador es minimizar el error entre la trayectoria deseada $\{q_d(t), \dot{q}_d(t), \ddot{q}_d(t)\}$ y la trayectoria real del robot $\{q(t), \dot{q}(t)\}$. Para ello, se define el error articular:

$$e(t) = q_d(t) - q(t), \quad \dot{e}(t) = \dot{q}_d(t) - \dot{q}(t),$$

y la acción PID se aplica sobre la aceleración de referencia:

$$\ddot{q}_{\text{ref}}(t) = \ddot{q}_d(t) + K_p e(t) + K_d \dot{e}(t) + K_i \int e(t) dt.$$

A partir de esta aceleración de referencia, el par de control se obtiene mediante la ecuación dinámica inversa del robot:

$$\tau_{\text{PID}}(t) = M(q) \ddot{q}_{\text{ref}}(t) + C(q, \dot{q}) \dot{q}(t) + g(q),$$

donde $M(q)$ es la matriz de inercia, $C(q, \dot{q})$ el término centrífugo–coriolis y $g(q)$ el vector de gravedad. Este enfoque permite compensar la dinámica del manipulador y mejorar significativamente el seguimiento de la trayectoria.

Para simular condiciones no ideales, se añade una perturbación externa arbitraria $\tau_{\text{pert}}(t)$, de modo que el par total aplicado es:

$$\tau_{\text{total}}(t) = \tau_{\text{PID}}(t) + \tau_{\text{pert}}(t).$$

La evolución real del robot se obtiene integrando la dinámica directa:

$$\ddot{q}(t) = M^{-1}(q(t)) (\tau_{\text{total}}(t) - C(q(t), \dot{q}(t)) \dot{q}(t) - g(q(t))).$$

Aunque el diseño detallado de controladores PID y su sintonización queda fuera del alcance de esta asignatura, en este trabajo se ha implementado con fines didácticos. De hecho, los fabricantes, por seguridad, acostumbran a proporcionar el controlador del robot como una “caja negra” inalterable. Por tanto, esta sección es meramente académica, ya que el robot real trae su propio controlador sintonizado.

En las figuras 20–25 se presentan las trayectorias calculadas y reales seguidas por el robot, donde se observa que las articulaciones 4, 5 y 6 son más difíciles de controlar y requieren una correcta sintonización de constantes K_p , K_d y K_i . En las figuras 26 y 27 se muestran las trayectorias articulares y cartesiana calculadas, en la 28 y 29 las velocidades y articulaciones articulares calculadas y en la 30 el par articular calculado durante la trayectoria.

La implementación se encuentra en el código 20.

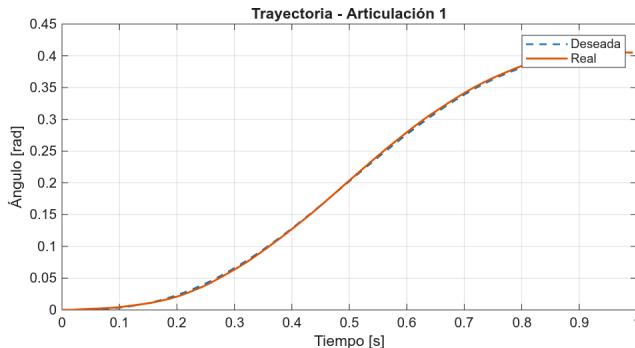


Figura 20: Par articular en la articulación 1.

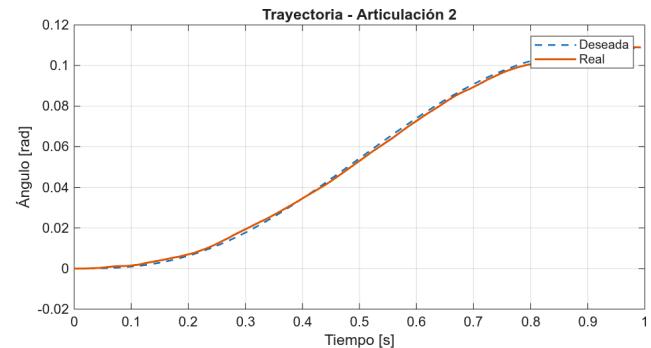


Figura 21: Par articular en la articulación 2.

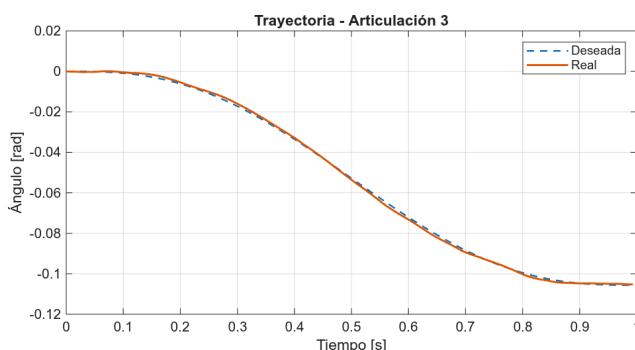


Figura 22: Par articular en la articulación 3.

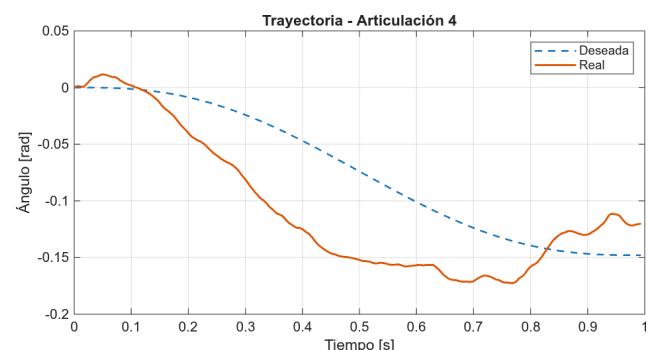


Figura 23: Par articular en la articulación 4.

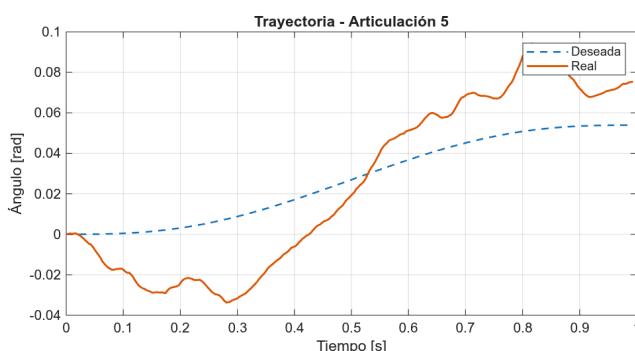


Figura 24: Par articular en la articulación 5.

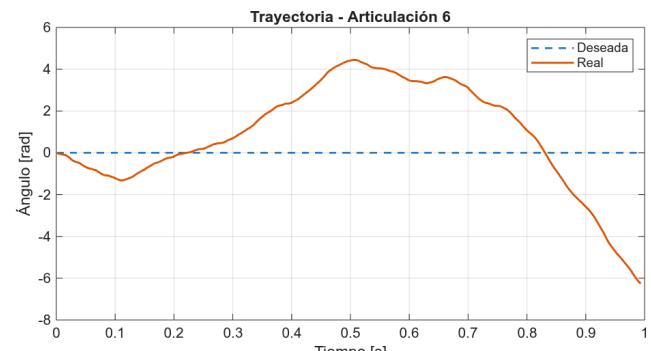


Figura 25: Par articular en la articulación 6.

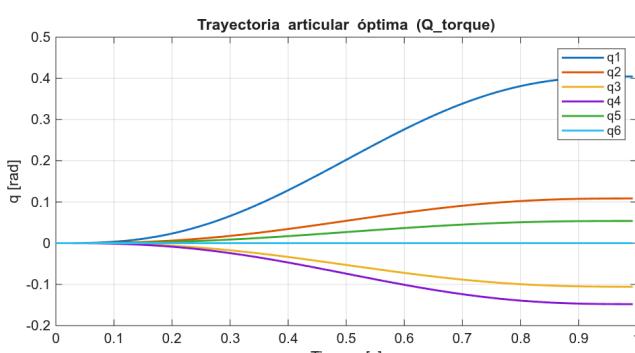


Figura 26: Trayectoria articular.

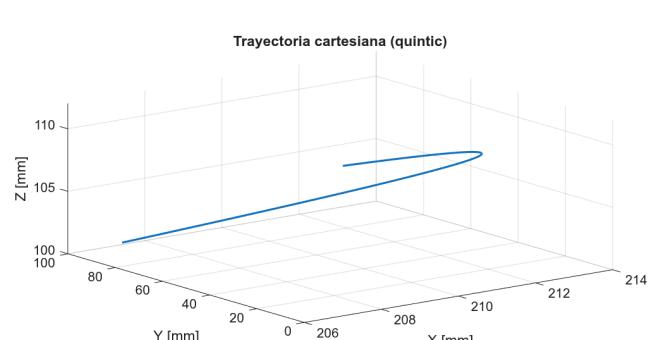


Figura 27: Trayectoria cartesiana.

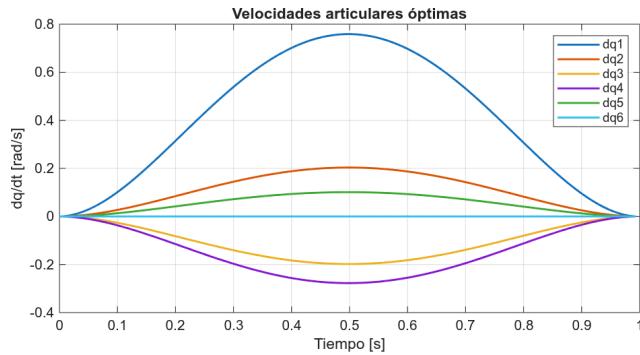


Figura 28: Velocidades articulares.

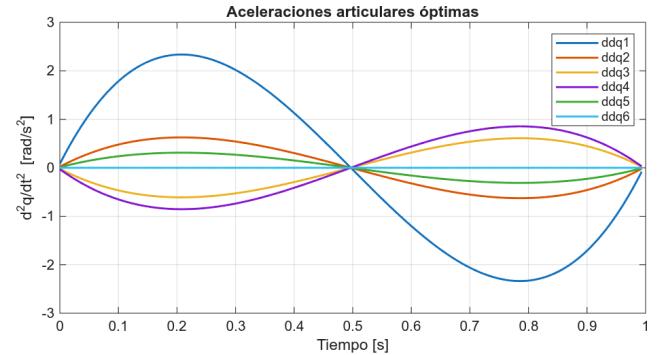


Figura 29: Aceleraciones articulares.

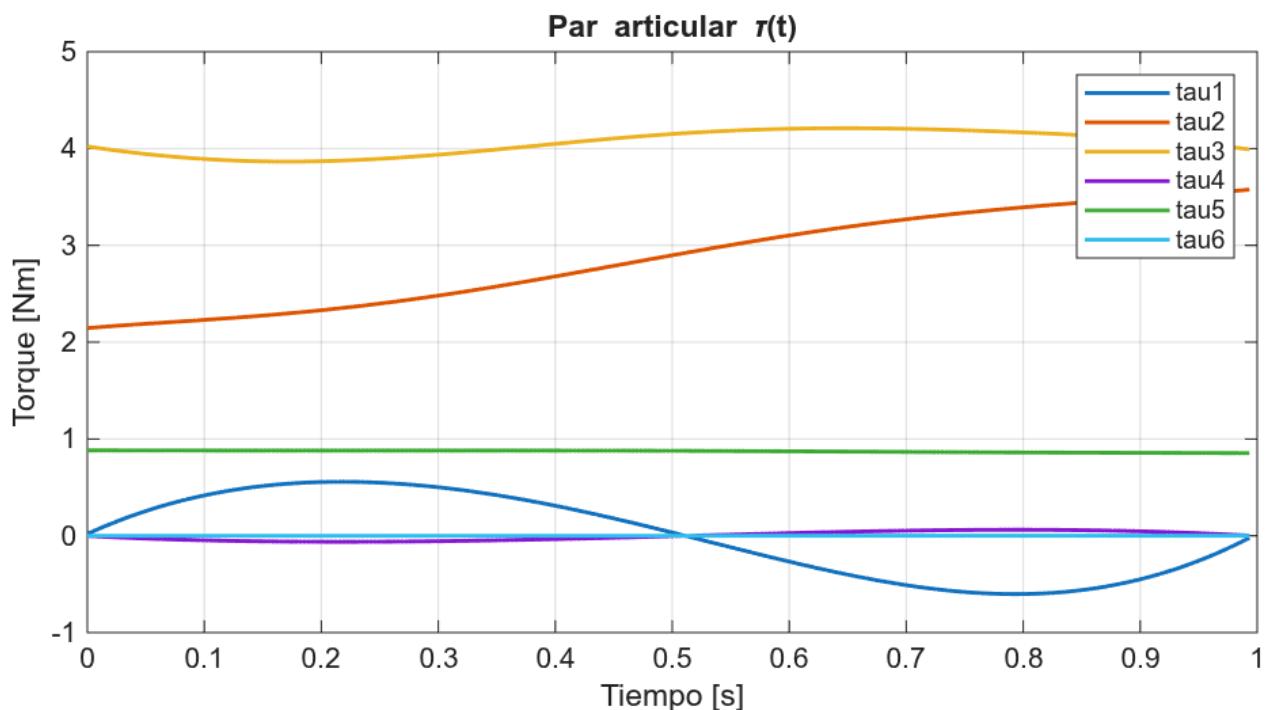


Figura 30: Par articular total para las seis articulaciones durante el seguimiento.

7. Programación con MoveIt2-ROS2-Gazebo

La integración entre MoveIt2, ROS2 y Gazebo constituye uno de los entornos más completos para la programación, simulación y control de manipuladores robóticos modernos.

MoveIt2 es el *framework* de planificación de movimiento más extendido en el ecosistema ROS2, proporcionando algoritmos avanzados de cinemática, planificación de trayectorias, control y percepción [20].

Por su parte, ROS2 actúa como la capa de comunicación distribuida que permite la interacción entre nodos, servicios y acciones [7], mientras que Gazebo ofrece un entorno de simulación física realista para validar el comportamiento del robot antes de su despliegue en *hardware real* [6].

En el caso de manipuladores comerciales como los brazos de la familia *xArm*, el fabricante proporciona paquetes ROS2 ya configurados, incluyendo descripciones URDF/XACRO, controladores, modelos de Gazebo y configuraciones básicas de MoveIt2 [8]. Esto permite que el usuario pueda comenzar a trabajar directamente desde la interfaz gráfica de MoveIt2 en RVIZ [rviz] o mediante nodos que invoquen los servicios y acciones expuestos por MoveIt2, sin necesidad de realizar configuraciones complejas adicionales.

En el robot real, la controladora del fabricante es la que implementa la dinámica y el control de bajo nivel, siendo un componente cerrado e inalterable. Esto implica que MoveIt2 únicamente envía trayectorias en forma de puntos de referencia o comandos de posición y velocidad, mientras que la controladora se encarga de ejecutar el movimiento de manera segura y estable. No obstante, en los apartados anteriores de la memoria se ha expuesto cómo se diseñan y crean estas controladoras.

En esta sección se va a introducir el funcionamiento de MoveIt2 y las trayectorias y algoritmos que emplea, y la programación básica mediante la interfaz gráfica y automatizada mediante un nodo de ROS2 en C++.

7.1. Algoritmos de planificación y generación de trayectorias en MoveIt2

MoveIt2 no solo calcula rutas geométricas en el espacio articular o cartesiano, sino que también aplica algoritmos de *time-parameterization* para generar trayectorias dinámicamente ejecutables. Estos algoritmos transforman una ruta geométrica en una trayectoria suave que respeta límites de velocidad, aceleración y, en algunos casos, *jerk*, que es la derivada de la aceleración con respecto del tiempo y mide los cambios bruscos en la aceleración. A continuación se mencionan los principales métodos utilizados por MoveIt2, quedando fuera del alcance de este proyecto su descripción detallada.

7.1.1. Planificadores geométricos (*OMPL*)

MoveIt2 utiliza la librería *OMPL* (*Open Motion Planning Library*) para generar rutas geométricas en el espacio de configuraciones, sin información temporal ni dinámica, por lo que para que el

robot pueda ejecutarla es necesario aplicar un algoritmo de parametrización temporal. Entre los algoritmos más empleados para la planificación geométrica se encuentran:

- *RRTConnect*: planificador rápido basado en muestreo, muy usado por defecto.
- *RRT*: versión óptima que minimiza la longitud de la trayectoria.
- *PRM / PRM*: planificadores basados en grafos probabilísticos.
- *KPIECE, BKPIECE*: planificadores basados en proyección.

7.1.2. Interpolación cartesiana

Cuando se requiere que el efecto final siga una trayectoria estricta en el espacio cartesiano (por ejemplo, líneas rectas o movimientos de inserción), MoveIt2 emplea interpolación lineal entre poses consecutivas. Esta técnica es útil en tareas de soldadura, pulido o aplicación de adhesivos, ensamblaje o inserción de piezas.

La interpolación cartesiana genera una ruta suave en el espacio del efecto final, pero nuevamente sin información temporal.

7.1.3. Algoritmos de parametrización temporal

Una vez generada la ruta geométrica, MoveIt2 aplica un algoritmo de *time-parameterization* para obtener una trayectoria ejecutable. Los principales métodos disponibles son:

- *Iterative Parabolic Time Parameterization (IPTP)*: algoritmo clásico que genera trayectorias con perfiles parabólicos de velocidad. Garantiza continuidad en posición y velocidad, pero no en *jerk*.
- *Time-Optimal Trajectory Generation (TOTG)*: algoritmo que calcula la trayectoria más rápida posible respetando límites de velocidad y aceleración. Produce movimientos suaves y eficientes.
- *Ruckig (jerk-limited)*: opción moderna para generar trayectorias con continuidad en posición, velocidad, aceleración y *jerk*. Es ideal para robots industriales y manipuladores sensibles a cambios bruscos.
- *Splines cúbicos y quinticos*: utilizados internamente para suavizar trayectorias y garantizar continuidad en derivadas superiores. Los polinomios de quinto orden son comunes en robótica por su suavidad y estabilidad.

En la mayoría de configuraciones por defecto, MoveIt2 emplea *IPTP* o *Ruckig*, dependiendo de la versión y del paquete del robot.

7.1.4. Ejecución de la trayectoria

Es importante remarcar que MoveIt2 no ejecuta la trayectoria. Su función termina cuando genera una secuencia de puntos temporizados. La ejecución real depende del motor físico de Gazebo en simulación o de la controladora del fabricante en el robot real, que en robots comerciales como xArm, la controladora interna es la responsable de aplicar la dinámica, compensación de gravedad, control de torque y estabilidad, mientras que MoveIt2 solo envía referencias de posición o velocidad.

7.2. Programación básica con la GUI de MoveIt2 en RVIZ

Al arrancar RVIZ, MoveIt2 y Gazebo aparecen dos ventanas: RVIZ y Gazebo. En RVIZ se muestra una visualización del robot en color naranja, representando el estado o posición objetivo (*goal*), es decir, al que se quiere llegar. Si se activa la casilla “*Query start state*” se añade una tercera visualización en color verde, posición de inicio, y si se desactiva “*Show robot visual*” aparece una tercera visualización en color gris: la posición actual del robot en Gazebo. En la figura 31 se observan estas tres visualizaciones. Las posiciones iniciales y finales se pueden mover arrastrando las esferas ubicadas en los correspondientes efectores.

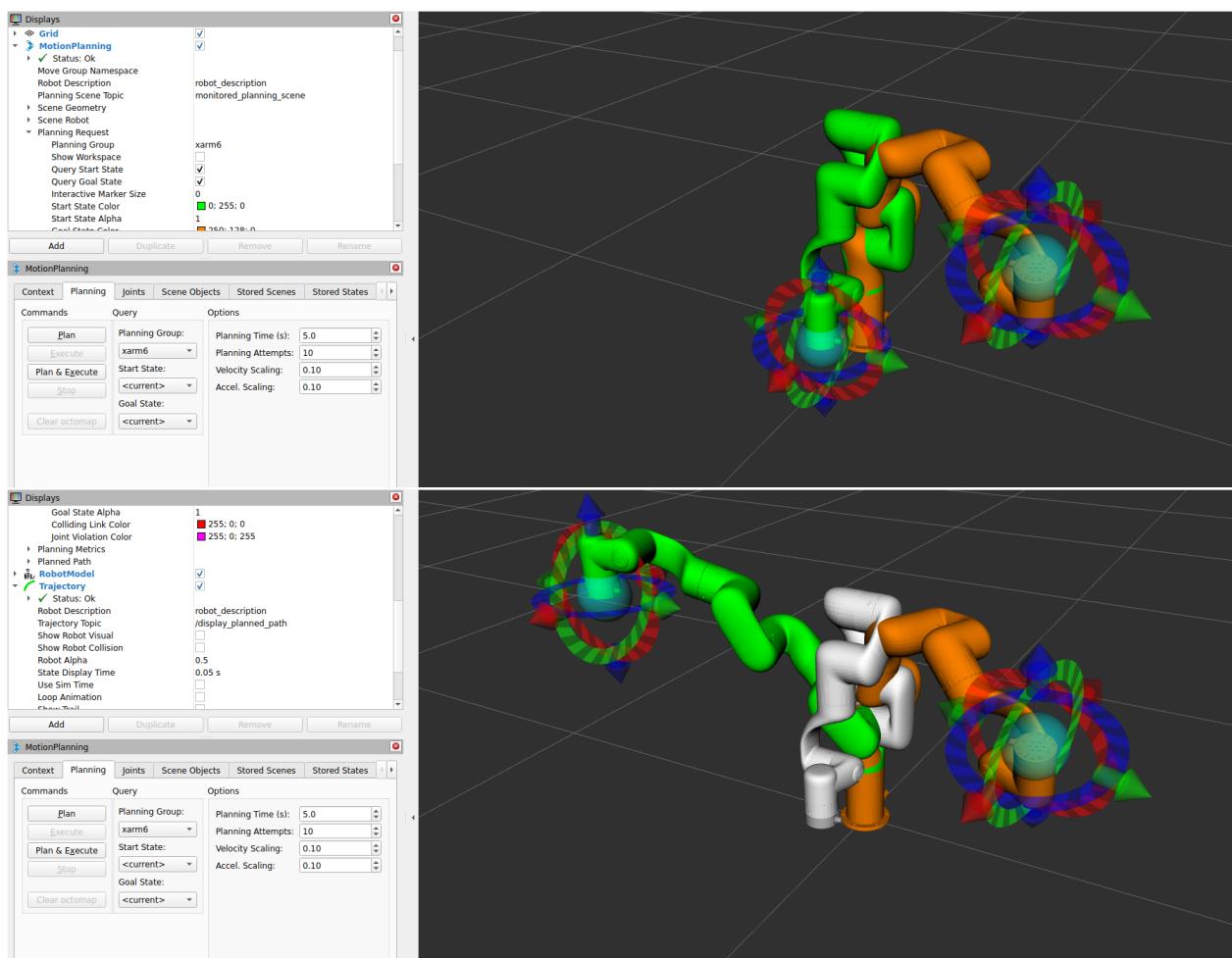


Figura 31: Visualizaciones real, inicial y final del robot en RVIZ.

Al pulsar el botón de “*Plan*” en el panel de *MotionPlanning* se genera la planificación desde la pose inicial a la final, pero al pulsar “*Plan & Execute*” la trayectoria se recalcula para ejecutarse desde la posición actual, en color gris, figura 32. Cuando finaliza la ejecución, las posiciones inicial, actual y final coinciden.

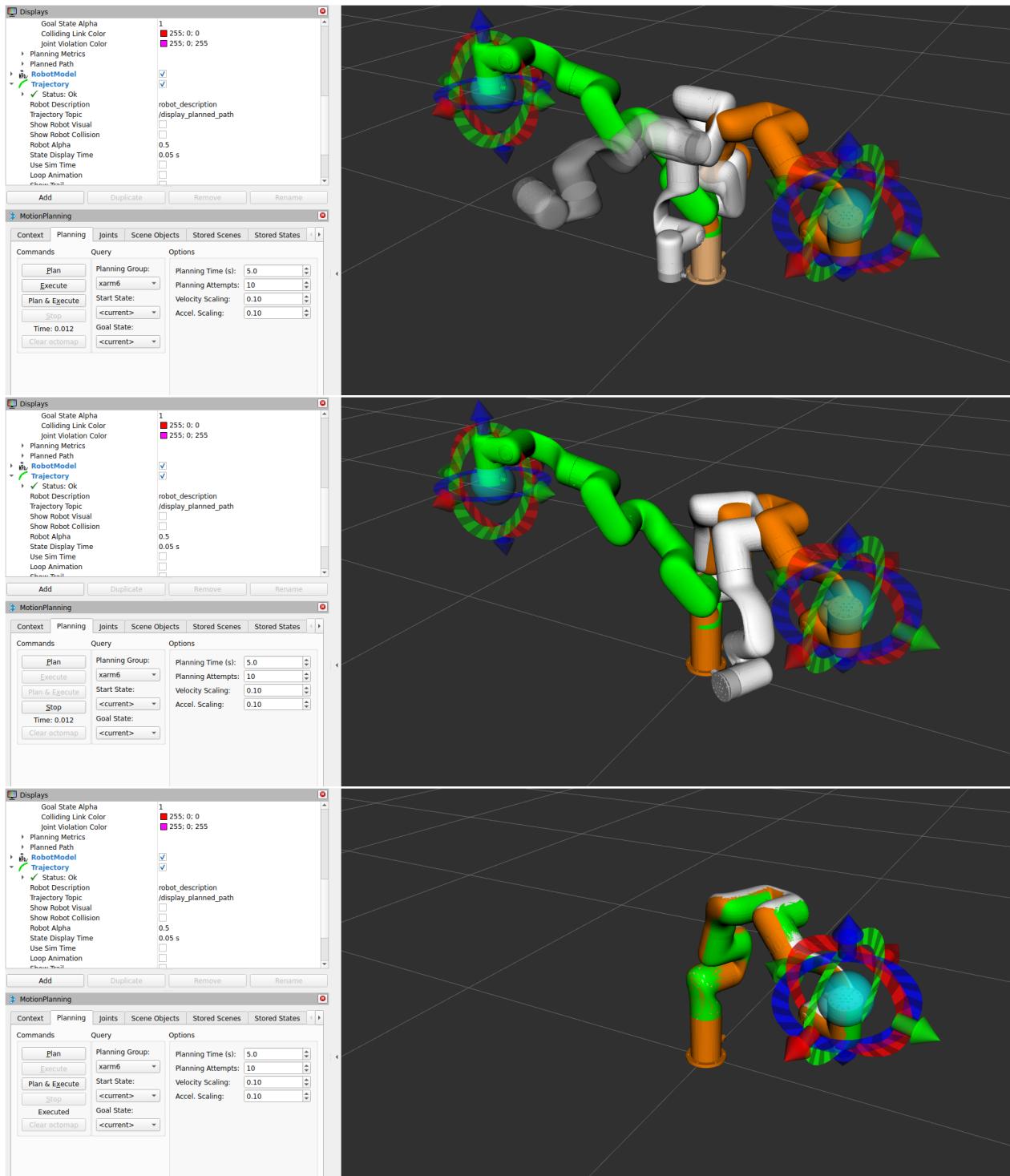


Figura 32: Planificación y ejecución de la trayectoria.

En la trayectoria del ejemplo no se han producido colisiones en las posiciones inicial ni final, pero si ocurriesen, para garantizar la seguridad y que son estados factibles es recomendable activa la casilla “*Collision Enabled*”, figura 33. También se puede activar “*Collision-aware IK*” para que la cinemática inversa sea consciente de las posiciones que generan colisiones.

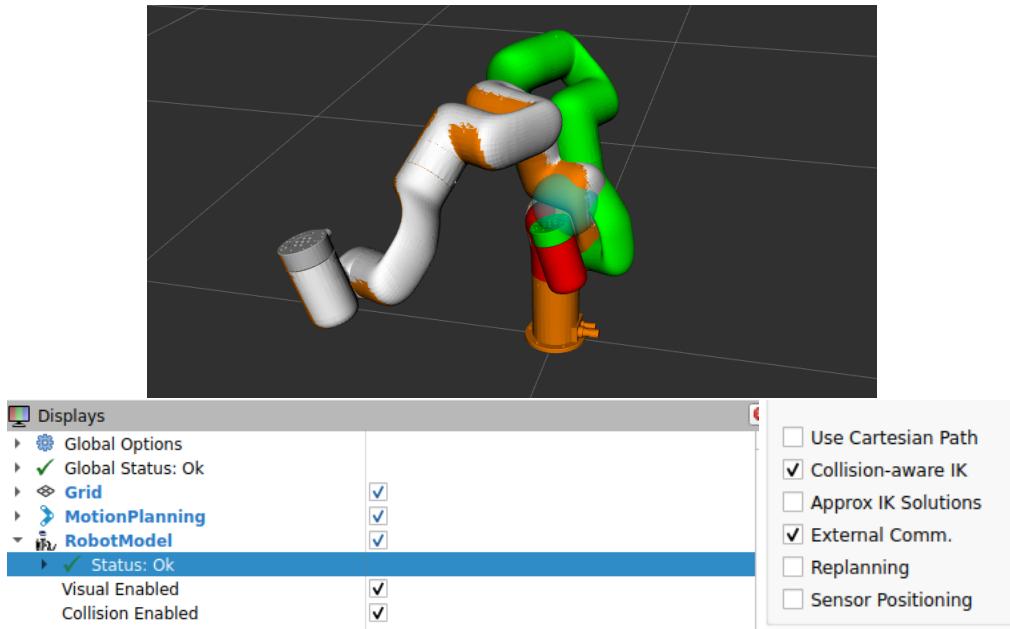


Figura 33: Colisiones activadas.

Otra forma de mover las poses inicial y final es mediante el panel “*Joints*” del panel de *MotionPlanning*, figura 34, donde “*Nullspace exploration*” permite descubrir nuevas configuraciones que mantienen la posición, diseñado para robots redundantes.

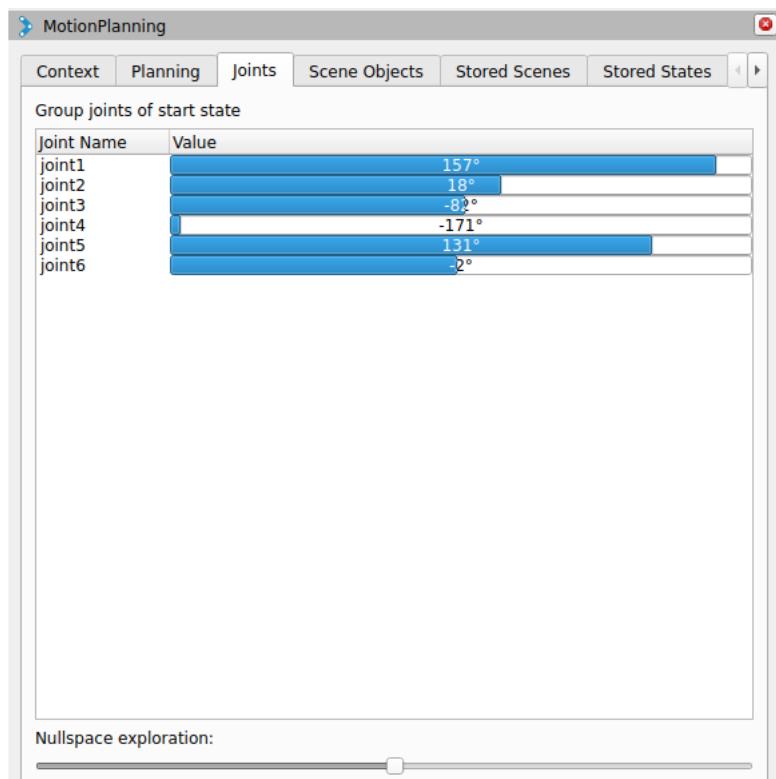


Figura 34: Panel de *joints*.

Si se activa “Show trail” se muestra el movimiento de la trayectoria planificada, figura 35. Para ver cada punto de paso generado, hay que activar el “Trajectory Slider” y deshabilitar el movimiento anterior, figura 36.

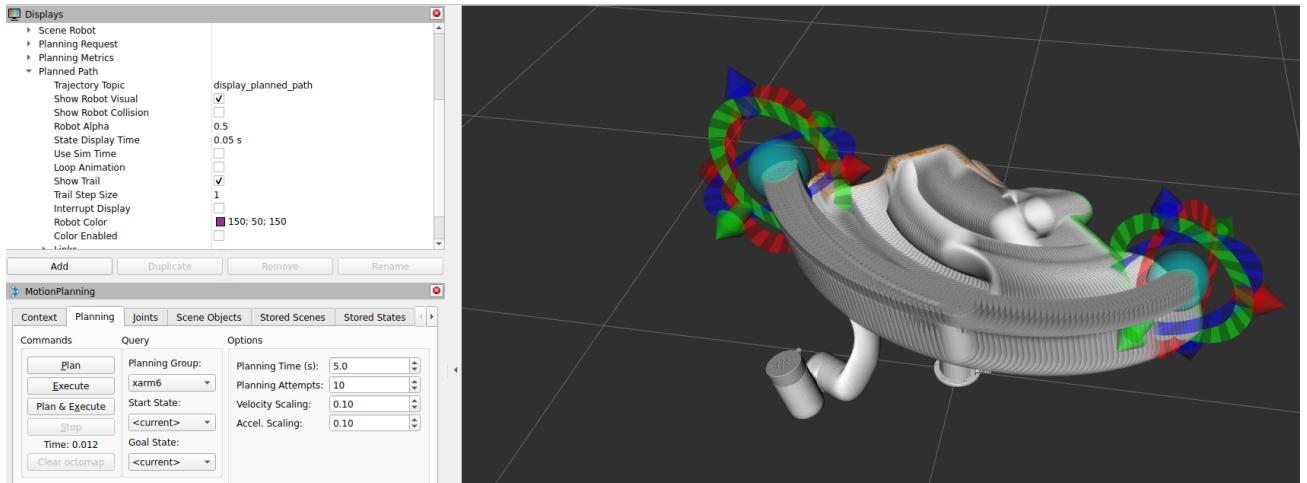


Figura 35: Movimiento de la trayectoria planificada.

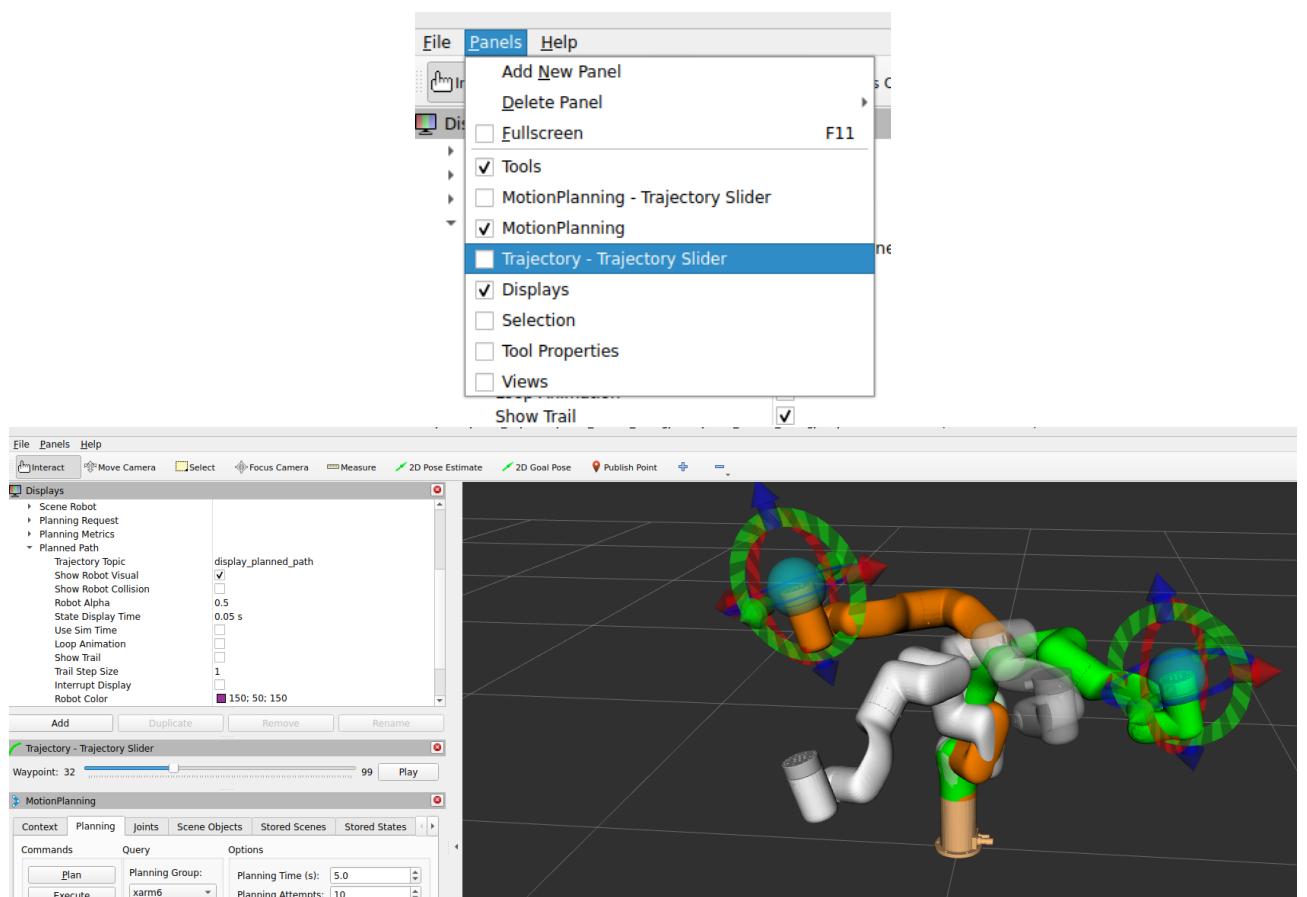


Figura 36: Visualización de puntos de paso.

Si se activa la casilla de “*Cartesian Path*” en el cuadro “*Options*” del panel de *MotionPlanning* se genera una trayectoria cartesiana, figura 37

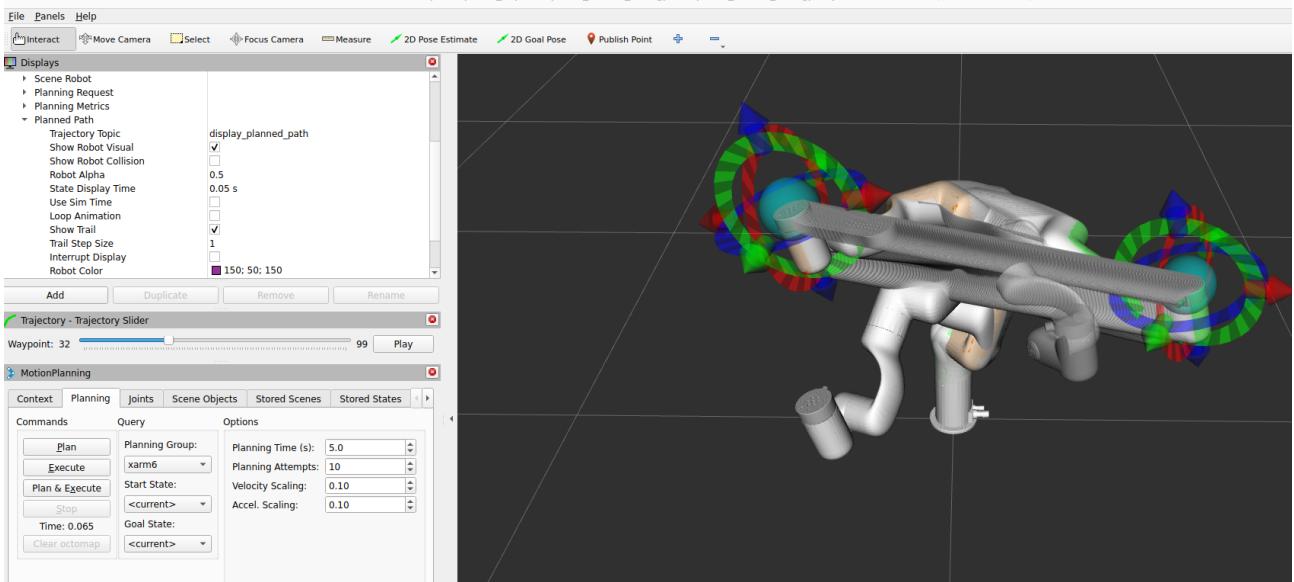


Figura 37: Trayectoria cartesiana.

Por último, se puede controlar la velocidad y aceleración, entre otros parámetros, mediante el cuadro “*Options*” del panel de *MotionPlanning*.

7.3. Programación básica con un nodo en C++

Movelt2 expone una serie de servicios y acciones en ROS2 que permiten interactuar con el planificador de movimiento desde nodos externos. Estos *endpoints* constituyen la interfaz principal para solicitar planes, ejecutar trayectorias, obtener información del estado cinemático o consultar la escena de planificación. En particular, el componente `move_group` actúa como servidor central y ofrece una acción denominada `/move_action`, basada en el tipo `moveit_msgs/action/MoveGroup`. Esta acción permite enviar objetivos de movimiento tanto articulares como cartesianos, junto con restricciones, factores de escalado de velocidad y aceleración, y otros parámetros de planificación.

Entre los servicios y acciones más relevantes que expone Movelt2 se encuentran:

- Acción `/move_action`: interfaz principal para solicitar planes y ejecutar movimientos mediante el tipo `MoveGroup`. Permite definir objetivos articulares, pose targets, rutas cartesianas y restricciones complejas.
- Servicio `/compute_cartesian_path`: genera rutas cartesianas interpoladas directamente en el espacio del efecto final.
- Servicio `/get_planning_scene`: devuelve la escena de planificación actual, incluyendo obstáculos, colisiones y estado del robot.
- Servicio `/query_planner_interfaces`: lista los planificadores disponibles (OMPL, CHOMP, STOMP, etc.).
- Acción `/execute_trajectory`: ejecuta una trayectoria ya planificada sin volver a calcularla.

En este trabajo se utiliza la acción `/move_action`, que es la forma más directa de solicitar a MoveIt2 que planifique y ejecute un movimiento. Para ello se implementa un nodo en C++ que actúa como *action client*. El nodo se conecta al servidor de acciones de MoveIt2, construye un objetivo articular y espera a que la acción finalice antes de continuar con la secuencia.

El nodo desarrollado se muestra en el código 21 y en la figura 38 se observa la ejecución de la trayectoria en Gazebo.

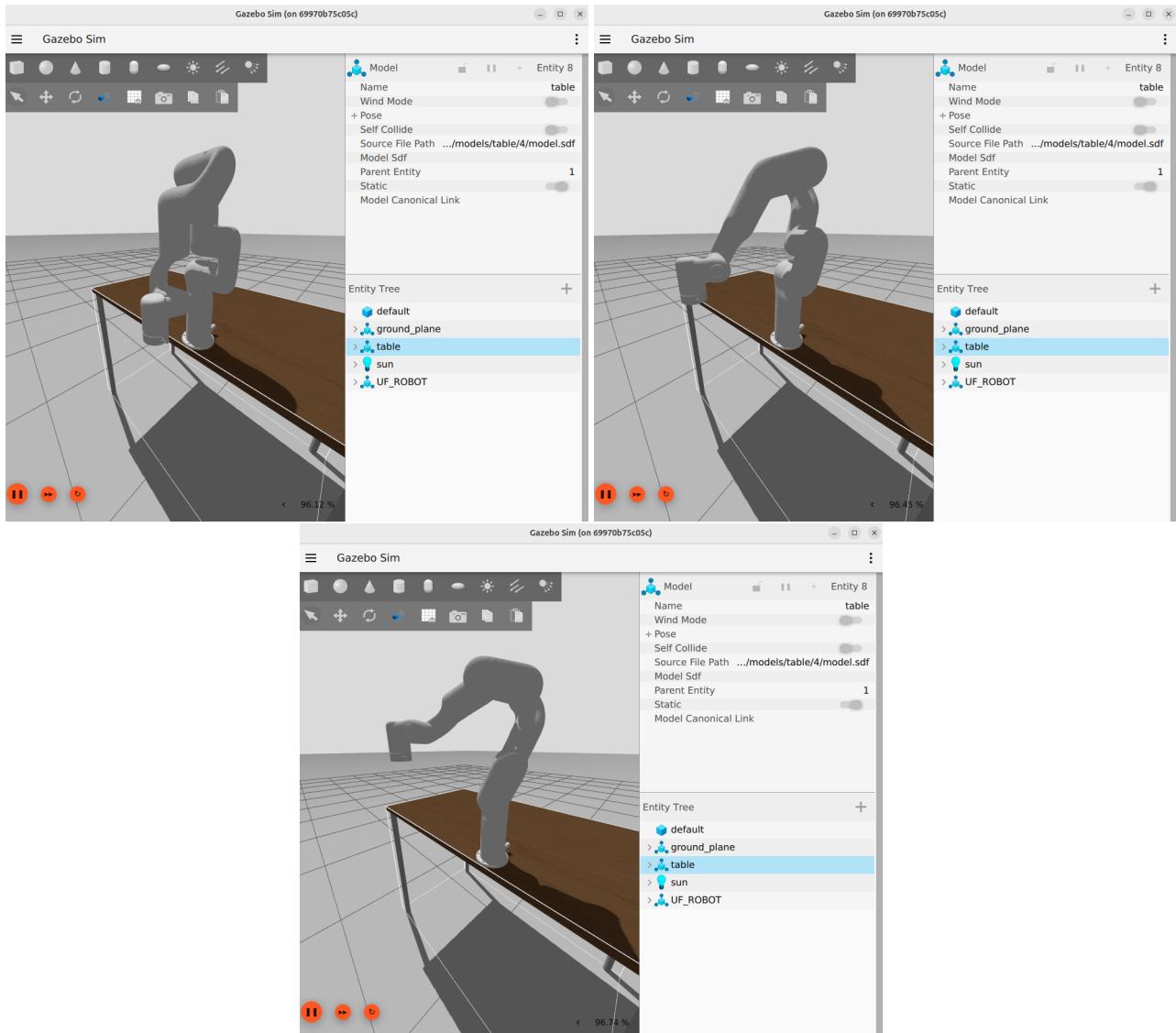


Figura 38: Movimiento del robot en Gazebo.

8. Evaluación y análisis de costes

La adquisición e integración de un manipulador industrial como el *xArm6* de *UFactory* implica una serie de costes directos e indirectos que deben ser evaluados para determinar la viabilidad económica del sistema. En esta sección se analizan los componentes principales del coste y la relación coste–beneficio en el contexto de aplicaciones de manipulación robótica.

8.1. Costes directos

Los costes directos corresponden a los elementos necesarios para disponer del robot en funcionamiento básico. Para el caso del manipulador *xArm6*, los principales componentes son:

- Robot *UFactory xArm6*: incluye el brazo robótico de 6 grados de libertad, la controladora integrada y el cableado principal. Su precio suele situarse alrededor de los 9.000 €.
- Pinza o efecto final: el fabricante ofrece pinzas paralelas, pinzas neumáticas y herramientas adicionales como motores lineales y cámaras de visión. El coste típico oscila entre 600 y 3.000 €.
- Fuente de alimentación y accesorios: soportes, bases de montaje, extensiones de cableado y elementos de fijación. Ya incluido con el brazo robótico.
- Software y licencias: el ecosistema de *UFactory*, ROS2 y Movelt2 es de código abierto, por lo que no existen costes de licencia. Sin embargo, pueden existir costes asociados a software de terceros o módulos adicionales.

El coste directo total estimado para un sistema funcional básico se sitúa alrededor de 12.000 €.

8.2. Costes indirectos

Además del coste del robot, existen costes asociados a su integración en un entorno real o de laboratorio:

- Integración con ROS2 y Movelt2: aunque el fabricante proporciona paquetes ROS2 preconfigurados, la integración con la aplicación final puede requerir entre 20 y 60 horas de trabajo técnico.
- Infraestructura: mesa de trabajo, protecciones, sensores adicionales o cámaras externas.
- Formación: capacitación del personal en ROS2, Movelt2 y programación del robot.
- Mantenimiento: aunque el *xArm6* no requiere mantenimiento intensivo, se recomienda una revisión anual y posibles sustituciones de piezas menores.

8.3. Relación coste–beneficio

El *xArm6* destaca por su relación coste–prestaciones en comparación con manipuladores industriales tradicionales, cuyo precio suele situarse entre 25.000 y 60.000 €. Para aplicaciones de investigación, docencia, prototipado rápido o automatización ligera, el *xArm6* ofrece un coste significativamente inferior, integración nativa con ROS2 y MoveIt2, controladora cerrada y estable proporcionada por el fabricante, facilidad de uso mediante interfaz gráfica o nodos ROS2 y bajo coste de mantenimiento.

En consecuencia, el sistema resulta económicamente atractivo para entornos donde la precisión extrema o la carga elevada no son requisitos críticos, pero sí lo son la flexibilidad, la rapidez de integración y el coste reducido.

Anexo: código

Estudio cinemático

Cinemática directa con parámetros DH clásicos

```

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 la 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
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.

Cinemática directa con parámetros DH modificados

```

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 según la tabla DH modificada
6 % Link([theta d a alpha sigma offset], 'modified')
7 L1_link = Link([0 267 0 0 0], 'modified'); % Eslabón 1
8 L2_link = Link([0 0 -pi2 0 T2_offset], 'modified'); % Eslabón 2
9 L3_link = Link([0 0 a2 0 0 T3_offset], 'modified'); % Eslabón 3
10 L4_link = Link([0 342.5 77.5 -pi2 0 0], 'modified'); % Eslabón 4
11 L5_link = Link([0 0 pi2 0 0], 'modified'); % Eslabón 5
12 L6_link = Link([0 97 76 -pi2 0 0], 'modified'); % Eslabón 6
13
14 % Crear el robot
15 robot = SerialLink([L1_link L2_link L3_link L4_link L5_link L6_link], ...
16     'name', 'xArm6', 'modified');
17
18 % Visualización del robot con q_i=0
19 robot.plot([0 0 0 0 0 0]);
20
21 % Vector de articulaciones
22 q = [q1 q2 q3 q4 q5 q6];
23
24 % Construir la transformación total manualmente con las matrices .T
25 T01 = simplify(L1_link.A(q1).T);
26 T12 = simplify(L2_link.A(q2).T);
27 T23 = simplify(L3_link.A(q3).T);
28 T34 = simplify(L4_link.A(q4).T);
29 T45 = simplify(L5_link.A(q5).T);
30 T56 = simplify(L6_link.A(q6).T);
31
32 % Transformación total simbólica
33 T06_sym = simplify(T01 * T12 * T23 * T34 * T45 * T56);
34
35 % Sustitución numérica
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 = vpa(subs(T06_sym, S), 10);
42 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.

Cinemática inversa

```

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 según la tabla DH modificada
6 L1 = Link([0 267 0 0 0], 'modified'); % Eslabón 1
7 L2 = Link([0 0 -pi2 0 T2_offset], 'modified'); % Eslabón 2
8 L3 = Link([0 0 a2 0 0 T3_offset], 'modified'); % Eslabón 3
9 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified'); % Eslabón 4
10 L5 = Link([0 0 pi2 0 0], 'modified'); % Eslabón 5
11 L6 = Link([0 97 76 -pi2 0 0], 'modified'); % Eslabón 6
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 disp('==> ECUACIONES DE POSICIÓN p06 ==>');
27 disp('p_x ='); pretty(p06(1))
28 disp('p_y ='); pretty(p06(2))
29 disp('p_z ='); pretty(p06(3))
30
31 disp(' ');
32 disp('==> ECUACIONES DE ORIENTACIÓN R06 ==>');
33 pretty(R06)

```

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

```

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 según la tabla DH modificada
6 L1 = Link([0 267 0 0 0 0], 'modified'); % Eslabón 1
7 L2 = Link([0 0 0 -pi2 0 T2_offset], 'modified'); % Eslabón 2
8 L3 = Link([0 0 a2 0 0 T3_offset], 'modified'); % Eslabón 3
9 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified'); % Eslabón 4
10 L5 = Link([0 0 0 pi2 0 0], 'modified'); % Eslabón 5
11 L6 = Link([0 97 76 -pi2 0 0], 'modified'); % Eslabón 6
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 p06 = T06(1:3,4);
24
25 % Jacobiano lineal (para cinemática inversa)
26 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
27
28 % Convertir a funciones numéricas
29 p_fun = matlabFunction(p06, 'Vars', ...
30 {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});
31 J_fun = matlabFunction(Jv, 'Vars', ...
32 {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});
33 T_fun = matlabFunction(T06, 'Vars', ...
34 {q1, q2, q3, q4, q5, q6, a2, T2_offset, T3_offset});
35
36 % Parámetros numéricos del robot
37 a2_val = 289.48866;
38 T2_offset_val = deg2rad(-79.34995);
39 T3_offset_val = deg2rad(79.34995);
40
41 % ---
42 % Cinemática inversa numérica (-NEWTONRAPHSON)
43 % ---
44 % Posición deseada (se sabe que corresponde a q_i = 0 con esos offsets)
45 p_d = [207; 0; 112];
46
47 % Configuración inicial
48 qk = zeros(6,1); % punto de partida
49 tol = 1e-6;
50 maxIter = 50;
51
52 % Límites matemáticos de las articulaciones (incluyendo offsets en 2 y 3)

```

```

53 % Rangos físicos (tabla 1):
54 % J1: ±360°
55 % J2: -117° -- 116° (físicos)
56 % J3: -219° -- 10° (físicos)
57 % J4: ±360°
58 % J5: -97° -- 180°
59 % J6: ±360°
60 %
61 % Convertimos a límites "matemáticos" para q2 y q3 (sin offset):
62 % q2_mat [q2_fis_min - offset2, q2_fis_max - offset2]
63 % q3_mat [q3_fis_min - offset3, q3_fis_max - offset3]
64 q1_min = -360; q1_max = 360;
65 q2_min = -117 - rad2deg(T2_offset_val); % grados matemáticos
66 q2_max = 116 - rad2deg(T2_offset_val);
67 q3_min = -219 - rad2deg(T3_offset_val);
68 q3_max = 10 - rad2deg(T3_offset_val);
69 q4_min = -360; q4_max = 360;
70 q5_min = -97; q5_max = 180;
71 q6_min = -360; q6_max = 360;
72
73 q_min = deg2rad([q1_min, q2_min, q3_min, q4_min, q5_min, q6_min])';
74 q_max = deg2rad([q1_max, q2_max, q3_max, q4_max, q5_max, q6_max])';
75
76 % Centro del rango (para penalización suave)
77 q_mid = (q_min + q_max) / 2;
78
79 % Ganancia del término de penalización (joint limit avoidance)
80 K = 0.005;
81
82 for k = 1:maxIter
83
84     % Evaluar posición y Jacobiano
85     % (incluyen offsets vía a2_val, T2_offset_val, T3_offset_val)
86     p_now = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), ...
87                     a2_val, T2_offset_val, T3_offset_val); % 3x1
88     J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), ...
89                     a2_val, T2_offset_val, T3_offset_val); % 3x6
90
91     % Error
92     e = p_d - p_now;
93
94     if norm(e) < tol
95         break
96     end
97
98     % Gradiente de la función para evitar límites
99     gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);
100
101    % Paso -NewtonRaphson (pseudoinversa) + penalización suave
102    dq = pinv(J_now) * e - K * gradH;
103
104    % Actualizar
105    qk = qk + dq;
106 end
107
108 % Redondear solución para limpiar ruido numérico
109 q_solution = round(qk, 12);
110 % ---
111
112 % Verificación: evaluar T06(q*)
113 T_check = T_fun(q_solution(1), q_solution(2), q_solution(3), ...
114                 q_solution(4), q_solution(5), q_solution(6), ...
115                 a2_val, T2_offset_val, T3_offset_val);
116 p_check = T_check(1:3,4);
117
118 disp('==> SOLUCIÓN DE CINEMÁTICA INVERSA (en radianes) ==>')
119 disp(q_solution.)
120
121 disp('==> SOLUCIÓN DE CINEMÁTICA INVERSA (en grados) ==>')
122 disp(rad2deg(q_solution.))
123
124 % Si se quieren ver los ángulos FÍSICOS de las articulaciones 2 y 3:
125 q_phys = q_solution;
126 q_phys(2) = q_phys(2) + T2_offset_val;
127 q_phys(3) = q_phys(3) + T3_offset_val;

```

```

128
129 disp('==> ÁNGULOS FÍSICOS (q + offset) EN GRADOS ==')
130 disp(rad2deg(q_phys).')
131
132 disp('==> POSICIÓN OBTENIDA ==')
133 disp(p_check)
134
135 disp('==> POSICIÓN DESEADA ==')
136 disp(p_d)
137
138 disp('==> ERROR VECTORIAL ==')
139 disp(e)
140
141 final_error = norm(e);
142 disp('==> ERROR FINAL ==')
143 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*.

Cinemática diferencial y Jacobiano

```

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 según la tabla DH modificada
6 L1 = Link([0 267 0 0 0 0], 'modified'); % Eslabón 1
7 L2 = Link([0 0 -pi2 0 T2_offset], 'modified'); % Eslabón 2
8 L3 = Link([0 a2 0 0 T3_offset], 'modified'); % Eslabón 3
9 L4 = Link([0 342.5 77.5 -pi2 0 0], 'modified'); % Eslabón 4
10 L5 = Link([0 0 pi2 0 0], 'modified'); % Eslabón 5
11 L6 = Link([0 97 76 -pi2 0 0], 'modified'); % Eslabón 6
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 disp('==> ECUACIONES DE POSICIÓN p06 ==>');
27 disp('p_x ='); pretty(p06(1))
28 disp('p_y ='); pretty(p06(2))
29 disp('p_z ='); pretty(p06(3))
30
31 disp(' ');
32 disp('==> ECUACIONES DE ORIENTACIÓN R06 ==>');
33 pretty(R06)
34
35 % Jacobiano geométrico (Jv + Jw)
36 % Jacobiano lineal usando jacobian
37 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
38
39 % Ejes z_i en el marco base
40 z0 = [0;0;1];
41 T01_ = T01;
42 T02_ = T01*T12;
43 T03_ = T01*T12*T23;
44 T04_ = T01*T12*T23*T34;
45 T05_ = T01*T12*T23*T34*T45;
46
47 z1 = T01_(1:3,3);
48 z2 = T02_(1:3,3);
49 z3 = T03_(1:3,3);
50 z4 = T04_(1:3,3);
51 z5 = T05_(1:3,3);
52
53 % Jacobiano angular
54 Jw = [z0 z1 z2 z3 z4 z5];
55
56 % Jacobiano geométrico completo
57 J = simplify([Jv; Jw]);
58
59 disp('==> JACOBIANO GEOMÉTRICO ==>');
60 pretty(J)
61
62 % Velocidades simbólicas
63 syms dq1 dq2 dq3 dq4 dq5 dq6 real
64 dq = [dq1; dq2; dq3; dq4; dq5; dq6];
65
66 % Contribución angular de cada articulación
67 omega1 = simplify(Jw(:,1) * dq1);
68 omega2 = simplify(Jw(:,2) * dq2);
69 omega3 = simplify(Jw(:,3) * dq3);
70 omega4 = simplify(Jw(:,4) * dq4);
71 omega5 = simplify(Jw(:,5) * dq5);
72 omega6 = simplify(Jw(:,6) * dq6);
73
74 % Velocidades del efecto
75 vel = simplify(J * dq);
76 v_linear = vel(1:3);
77 v_angular = vel(4:6);
78
79 disp('==> VELOCIDAD ANGULAR APORTADA POR CADA ARTICULACIÓN ==>');
80 disp('omega_1 ='); pretty(omega1)
81 disp(' '); disp('omega_2 ='); pretty(omega2)
82 disp(' '); disp('omega_3 ='); pretty(omega3)
83 disp(' '); disp('omega_4 ='); pretty(omega4)
84 disp(' '); disp('omega_5 ='); pretty(omega5)
85 disp(' '); disp('omega_6 ='); pretty(omega6)
86
87 disp(' ');
88 disp('==> VELOCIDAD LINEAL DEL EFECTO ==>');
89 pretty(v_linear)
90
91 disp(' ');
92 disp('==> VELOCIDAD ANGULAR DEL EFECTO ==>');
93 pretty(v_angular)
94
95 % Evaluación numérica de T06, J y velocidades
96 disp(' ');
97 disp('=====');

```

```

98 disp('==> EVALUACIÓN NUMÉRICA DE T06, J Y VELOCIDADES      ==>');
99 disp('====='); 
100 % Valores numéricos
101 S.q1 = 0; S.q2 = 0; S.q3 = 0; S.q4 = 0; S.q5 = 0; S.q6 = 0;
102 S.a2 = 289.48866;
103 S.T2_offset = deg2rad(-79.34995);
104 S.T3_offset = deg2rad(79.34995);
105
106 disp(' ');
107 disp('==> VALORES USADOS PARA LA EVALUACIÓN NUMÉRICA ==>');
108 fprintf('q1 = %.4f rad\n', S.q1);
109 fprintf('q2 = %.4f rad\n', S.q2);
110 fprintf('q3 = %.4f rad\n', S.q3);
111 fprintf('q4 = %.4f rad\n', S.q4);
112 fprintf('q5 = %.4f rad\n', S.q5);
113 fprintf('q6 = %.4f rad\n', S.q6);
114 fprintf('a2 = %.5f mm\n', S.a2);
115 fprintf('T2_offset = %.4f rad (%.2f°)\n', S.T2_offset, rad2deg(S.T2_offset));
116 fprintf('T3_offset = %.4f rad (%.2f°)\n', S.T3_offset, rad2deg(S.T3_offset));
117
118 % Transformación numérica
119 T06_num = vpa(subs(T06, S), 10);
120 disp(' ');
121 disp('==> TRANSFORMACIÓN HOMOGÉNEA T_0^6 NUMÉRICA ==>');
122 disp(T06_num);
123
124 % Jacobiano numérico
125 J_num = vpa(subs(J, S), 10);
126 disp(' ');
127 disp('==> JACOBIANO GEOMÉTRICO NUMÉRICO ==>');
128 disp(J_num);
129
130 % Velocidades articulares numéricas
131 dq_val = [0.2; 0.1; -0.15; 0.05; 0.1; -0.05];
132 vel_num = double(J_num * dq_val);
133
134 v_linear_num = vel_num(1:3);
135 v_angular_num = vel_num(4:6);
136
137 disp(' ');
138 disp('==> VELOCIDAD LINEAL DEL EFECTOR (numérica) ==>');
139 disp(v_linear_num.');
140
141 disp(' ');
142 disp('==> VELOCIDAD ANGULAR DEL EFECTOR (numérica) ==>');
143 disp(v_angular_num.');
144
```

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

Generación de trayectorias

```

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 % Definición de eslabones según la tabla DH modificada
6 L1 = Link([0      267    0      0      0  0], 'modified');           % Eslabón 1
7 L2 = Link([0      0      0      -pi2   0  T2_offset], 'modified'); % Eslabón 2
8 L3 = Link([0      0      a2     0      0  T3_offset], 'modified'); % Eslabón 3
9 L4 = Link([0      342.5  77.5   -pi2   0  0], 'modified');        % Eslabón 4
10 L5 = Link([0     0      0      pi2    0  0], 'modified');         % Eslabón 5
11 L6 = Link([0     97     76     -pi2   0  0], 'modified');         % Eslabón 6
12
13 % Transformación completa
14 T01 = L1.A(q1).T; T12 = L2.A(q2).T; T23 = L3.A(q3).T;
15 T34 = L4.A(q4).T; T45 = L5.A(q5).T; T56 = L6.A(q6).T;
16 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
17 p06 = T06(1:3,4);
18
19 % Funciones numéricas
20 T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
```

```

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

```

```

96 P(k,:) = Tnow(1:3,4)';
97 end
98
99 figure;
100 plot3(P(:,1),P(:,2),P(:,3),'b-','LineWidth',2);
101 grid on; xlabel('X'); ylabel('Y'); zlabel('Z');
102 title('Trayectoria del efecto final (quintica con IPTP)');
103
104 % Imprimir posiciones inicial y final
105 % Valores físicos (sumando offset en q2 y q3)
106 q0_phys = q0;
107 q0_phys(2) = q0_phys(2) + T2_offset_val;
108 q0_phys(3) = q0_phys(3) + T3_offset_val;
109
110 qf_phys = qf;
111 qf_phys(2) = qf_phys(2) + T2_offset_val;
112 qf_phys(3) = qf_phys(3) + T3_offset_val;
113
114 disp('== Valores iniciales físicos (grados) ==');
115 disp(rad2deg(q0_phys));
116
117 disp('== Valores finales físicos (grados) ==');
118 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.

Control cinemático

```

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 % Definición de eslabones según la tabla DH modificada
6 L1 = Link([0      267      0      0      0 0], 'modified');           % Eslabón 1
7 L2 = Link([0      0      -pi2      0  T2_offset], 'modified'); % Eslabón 2
8 L3 = Link([0      0      a2      0      0  T3_offset], 'modified'); % Eslabón 3
9 L4 = Link([0      342.5    77.5  -pi2      0 0], 'modified');    % Eslabón 4
10 L5 = Link([0      0      0      pi2      0 0], 'modified');     % Eslabón 5
11 L6 = Link([0      97      76  -pi2      0 0], 'modified');     % Eslabón 6
12
13 % Transformaciones y Jacobiano
14 T01 = L1.A(q1).T; T12 = L2.A(q2).T; T23 = L3.A(q3).T;
15 T34 = L4.A(q4).T; T45 = L5.A(q5).T; T56 = L6.A(q6).T;
16 T06 = simplify(T01 * T12 * T23 * T34 * T45 * T56);
17 p06 = T06(1:3,4);
18 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
19
20 % Funciones numéricas
21 p_fun = matlabFunction(p06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
22 J_fun = matlabFunction(Jv, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
23 T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6,a2,T2_offset,T3_offset});
24
25 % Parámetros del robot
26 a2_val        = 289.48866;
27 T2_offset_val = deg2rad(-79.34995);
28 T3_offset_val = deg2rad(79.34995);
29
30 % Límites físicos matemáticos (deg)
31 q1_min = -360;      q1_max = 360;
32 q2_min = -117 - rad2deg(T2_offset_val);
33 q2_max = 116 - rad2deg(T2_offset_val);
34 q3_min = -219 - rad2deg(T3_offset_val);
35 q3_max = 10 - rad2deg(T3_offset_val);
36 q4_min = -360;      q4_max = 360;

```

```

37 q5_min = -97;      q5_max = 180;
38 q6_min = -360;     q6_max = 360;
39
40 q_min = deg2rad([q1_min, q2_min, q3_min, q4_min, q5_min, q6_min] );
41 q_max = deg2rad([q1_max, q2_max, q3_max, q4_max, q5_max, q6_max] );
42
43 % Entrada: articulaciones actuales, pose final y duración
44 dlg = inputdlg( ...
45     {'Articulaciones actuales [q1 q2 q3 q4 q5 q6] en grados (matemático):', ...
46     'Posición final del efecto [x y z] en mm:', ...
47     'Duración nominal de la trayectoria [s]:'}, ...
48     'Entrada de movimiento', [1 70], {'0 0 0 0 0 0', '350 150 250', '5'});
49
50 if isempty(dlg)
51     q0_deg = input('Introduce [q1 q2 q3 q4 q5 q6] en grados (matemático): ');
52     p_goal = input('Introduce [x y z] en mm para la posición final del efecto: ');
53     T      = input('Introduce la duración nominal de la trayectoria [s]: ');
54 else
55     q0_deg = str2num(dlg{1});
56     p_goal = str2num(dlg{2});
57     T      = str2double(dlg{3});
58 end
59
60 assert(numel(q0_deg)==6, 'Debes introducir 6 valores articulares.');
61 assert(numel(p_goal)==3, 'Debes introducir 3 valores cartesianas [x y z].');
62
63 % Convertir a radianes y columna
64 q_init = deg2rad(q0_deg(:));
65 p_goal = p_goal(:);
66
67 % Pose inicial cartesiana derivada de las articulaciones actuales
68 T_init = T_fun(q_init(1), q_init(2), q_init(3), q_init(4), q_init(5), q_init(6), ...
69             a2_val, T2_offset_val, T3_offset_val);
70 p_init = T_init(1:3,4);
71
72 % Verificación de alcanzabilidad (radio de acción)
73 reach_max = 700; % mm (xArm6)
74 d_goal = norm(p_goal);
75
76 if d_goal > reach_max
77     fprintf('Objetivo a %.1f mm, fuera del alcance máximo de %.1f mm -> NO realizable\n', ...
78         d_goal, reach_max);
79     return; % detener el programa aquí
80 end
81
82 fprintf('Objetivo a %.1f mm, dentro del alcance máximo %.1f mm -> Realizable\n', ...
83         d_goal, reach_max);
84
85 % IK numérica con penalización de límites
86 tol = 1e-6; maxIter = 80; K = 0.005; % penalización
87 q_mid = (q_min + q_max)/2;
88
89 % Semilla: las articulaciones actuales
90 q_goal_seed = q_init;
91 q_goal = newtonIK(p_goal, q_goal_seed, p_fun, J_fun, ...
92                     a2_val, T2_offset_val, T3_offset_val, ...
93                     q_min, q_max, q_mid, K, tol, maxIter);
94
95 % Trayectoria quintica + escalado
96 n = 300; t = linspace(0,T,n);
97 Q = quinticTraj(q_init.', q_goal.', T, t);
98
99 % Límites dinámicos del xArm6 (usa valores del manual)
100 vel_max = deg2rad([180 180 180 180 180 180]); % rad/s
101 acc_max = deg2rad([1500 1500 1500 1500 1500 1500]); % rad/s^2
102
103 % Escalado: escalar tiempo si excede límites (recalcula la quintic con T escalado)
104 [Q, t, T_scaled] = scale_traj(Q, t, vel_max, acc_max);
105
106 % Cálculo de velocidades y aceleraciones
107 dt = t(2) - t(1); % intervalo de muestreo
108 dQ = diff(Q) / dt; % velocidades articulares (rad/s), tamaño (n-1) x 6
109 ddQ = diff(dQ) / dt; % aceleraciones articulares (rad/s^2), tamaño (n-2) x 6
110
111 % Velocidad cartesiana del efecto usando Jacobiano lineal

```

```

112 v_cart = zeros(n-1,3);
113 for k = 1:(n-1)
114     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);
115     v_cart(k,:) = (J_now * dQ(k,:)').'; % [vx vy vz] en mm/s si p_fun/T_fun están en mm
116 end
117
118 % Impresión de resultados (físicos y matemáticos)
119 q0 = q_init(:).';
120 qf = q_goal(:).';
121
122 % Valores físicos (sumando offsets en q2 y q3)
123 q0_phys = q0; q0_phys(2) = q0_phys(2) + T2_offset_val; q0_phys(3) = q0_phys(3) + T3_offset_val;
124 qf_phys = qf; qf_phys(2) = qf_phys(2) + T2_offset_val; qf_phys(3) = qf_phys(3) + T3_offset_val;
125
126 fprintf('\n==== Estado ====\n');
127 fprintf('Duración planificada (IPTP): %.2f s\n', T_scaled);
128
129 disp('==== Articulaciones iniciales (matemático) [deg] ==='); disp(rad2deg(q0));
130 disp('==== Articulaciones finales (matemático) [deg] ==='); disp(rad2deg(qf));
131 disp('==== Articulaciones iniciales (físico) [deg] ==='); disp(rad2deg(q0_phys));
132 disp('==== Articulaciones finales (físico) [deg] ==='); disp(rad2deg(qf_phys));
133 disp('==== Velocidades articulares iniciales [deg/s] ==='); disp(rad2deg(dQ(1,:)));
134 disp('==== Velocidades articulares finales [deg/s] ==='); disp(rad2deg(dQ(end,:)));
135 disp('==== Aceleraciones articulares iniciales [deg/s^2] ==='); disp(rad2deg(ddQ(1,:)));
136 disp('==== Aceleraciones articulares finales [deg/s^2] ==='); disp(rad2deg(ddQ(end,:)));
137
138 % Trayectoria del efecto final
139 P = zeros(n,3);
140 for k=1:n
141     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);
142     P(k,:) = Tnow(1:3,4).';
143 end
144
145 % Gráficas con anotaciones (posiciones físicas)
146 % Añadir offsets a J2 y J3 para graficar valores físicos
147 Q_phys = Q;
148 Q_phys(:,2) = Q_phys(:,2) + T2_offset_val; % offset J2
149 Q_phys(:,3) = Q_phys(:,3) + T3_offset_val; % offset J3
150
151 figure;
152 plot(t, rad2deg(Q_phys), 'LineWidth',1.5); hold on;
153 xlabel('Tiempo [s]'); ylabel('Ángulo [°]');
154 legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
155 title('Trayectoria articular (quintica + IPTP, valores físicos');
156
157 % Puntos inicio/fin visibles pero EXCLUIDOS de la leyenda
158 for i=1:6
159     plot(t(1), rad2deg(Q_phys(1,i)), 'go', 'MarkerFaceColor','g', 'HandleVisibility','off'); % inicio
160     text(t(1), rad2deg(Q_phys(1,i)), sprintf('%.1f', rad2deg(q0_phys(i))), ...
161         'VerticalAlignment','bottom','HorizontalAlignment','right');
162     plot(t(end), rad2deg(Q_phys(end,i)), 'ro', 'MarkerFaceColor','r', 'HandleVisibility','off'); % final
163     text(t(end), rad2deg(Q_phys(end,i)), sprintf('%.1f', rad2deg(qf_phys(i))), ...
164         'VerticalAlignment','top','HorizontalAlignment','left');
165 end
166
167 % Gráficas de velocidades y aceleraciones articulares
168 % Nota: dQ es de longitud n-1 y ddQ de n-2
169 figure;
170 plot(t(1:end-1), rad2deg(dQ), 'LineWidth',1.2); grid on;
171 xlabel('Tiempo [s]'); ylabel('Velocidad [°/s]');
172 legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
173 title('Velocidades articulares');
174
175 figure;
176 plot(t(1:end-2), rad2deg(ddQ), 'LineWidth',1.2); grid on;
177 xlabel('Tiempo [s]'); ylabel('Aceleración [°/s^2]');
178 legend('q1','q2','q3','q4','q5','q6','Location','bestoutside');
179 title('Aceleraciones articulares');
180
181 % Gráfica de velocidad y aceleración cartesiana del efecto
182 figure;
183 plot(t(1:end-1), v_cart, 'LineWidth',1.2); grid on;
184 xlabel('Tiempo [s]'); ylabel('Velocidad lineal [mm/s]');
185 legend('v_x','v_y','v_z','Location','bestoutside');
186 title('Velocidad cartesiana del efecto');

```

```

187
188 a_cart = diff(v_cart) / dt;           % aceleración cartesiana (mm/s^2), ...
189 %     tamaño (n-2) x 3
190 figure;
191 plot(t(1:end-2), a_cart, 'LineWidth',1.2); grid on;
192 xlabel('Tiempo [s]'); ylabel('Aceleración lineal [mm/s^2]');
193 legend('a_x','a_y','a_z','Location','bestoutside');
194 title('Aceleración cartesiana del efecto');
195
196 % Trayectoria del efecto final (posición)
197 figure;
198 plot3(P(:,1),P(:,2),P(:,3),'b-','LineWidth',2); hold on;
199 plot3(P(1,1),P(1,2),P(1,3),'go','MarkerFaceColor','g'); text(P(1,1),P(1,2),P(1,3),'Inicio', ...
200 'VerticalAlignment','bottom');
201 plot3(P(end,1),P(end,2),P(end,3),'ro','MarkerFaceColor','r'); text(P(end,1),P(end,2),P(end,3), ...
202 'Final','VerticalAlignment','top');
203 grid on; xlabel('X [mm]'); ylabel('Y [mm]'); zlabel('Z [mm]');
204 title('Trayectoria del efecto final');
205
206 % Funciones auxiliares
207 function q_sol = newtonIK(p_d, q_seed, p_fun, J_fun, a2_val, T2_offset_val, T3_offset_val, q_min, ...
208 q_max, q_mid, K, tol, maxIter)
209 % -NewtonRaphson SIN clamping (solo penalización suave de límites)
210 qk = q_seed(:);
211 for it = 1:maxIter
212     p_now = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), a2_val, T2_offset_val, T3_offset_val);
213     J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6), a2_val, T2_offset_val, T3_offset_val);
214     e = p_d - p_now;
215     if norm(e) < tol, break; end
216     gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);
217     dq = pinv(J_now) * e - K * gradH;
218     qk = qk + dq;
219 end
220 q_sol = qk;
221 end
222
223 function Q = quinticTraj(q0, qf, T, t)
224 % Polinomio quíntico con vel y acc cero en extremos
225 n = numel(t); Q = zeros(n, numel(q0));
226 for i=1:numel(q0)
227     a0 = q0(i); a1 = 0; a2 = 0;
228     a3 = 10*(qf(i)-q0(i))/T^3;
229     a4 = -15*(qf(i)-q0(i))/T^4;
230     a5 = 6*(qf(i)-q0(i))/T^5;
231     Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
232 end
233
234
235 function [Q_out, t_out, T_scaled] = scale_traj(Q, t, vel_max, acc_max)
236 % Escalado temporal para respetar límites de velocidad y aceleración
237 dt = t(2) - t(1);
238 dQ = diff(Q)/dt;
239 ddQ = diff(dQ)/dt;
240 scale_v = max(max(abs(dQ) ./ vel_max));
241 scale_a = max(max(abs(ddQ) ./ acc_max));
242 scale = max([1, scale_v, sqrt(scale_a)]);
243 if scale > 1
244     T_scaled = t(end) * scale;
245     t_out = linspace(0, T_scaled, numel(t));
246     % Recalcular la quintic con el nuevo T para mantener vel/acc cero en extremos
247     Q_out = zeros(size(Q));
248     q0 = Q(1,:); qf = Q(end,:);
249     for i=1:size(Q,2)
250         a0 = q0(i); a1 = 0; a2 = 0;
251         a3 = 10*(qf(i)-q0(i))/T_scaled^3;
252         a4 = -15*(qf(i)-q0(i))/T_scaled^4;
253         a5 = 6*(qf(i)-q0(i))/T_scaled^5;
254         Q_out(:,i) = a0 + a1*t_out + a2*t_out.^2 + a3*t_out.^3 + a4*t_out.^4 + a5*t_out.^5;
255     end
256 else
257     Q_out = Q; t_out = t; T_scaled = t(end);
258 end
259 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.

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.

```

1 clear all; close all; clc;
2
3 % Offsets articulares (en radianes)
4 T2_offset_val = deg2rad(-79.34995);
5 T3_offset_val = deg2rad(79.34995);
6
7 % Parámetro geométrico
8 a2_val = 0.28948866; % en metros
9
10 % Definición de eslabones según la tabla DH modificada, en SI
11 L1 = Link([0 0.267 0 0 0 0], 'modified'); % Eslabón 1
12 L2 = Link([0 0 0 -pi/2 0 T2_offset_val], 'modified'); % Eslabón 2
13 L3 = Link([0 0 a2_val 0 0 T3_offset_val], 'modified'); % Eslabón 3
14 L4 = Link([0 0.3425 0.0775 -pi/2 0 0], 'modified'); % Eslabón 4
15 L5 = Link([0 0 0 +pi/2 0 0], 'modified'); % Eslabón 5
16 L6 = Link([0 0.097 0.076 -pi/2 0 0], 'modified'); % Eslabón 6
17
18 % Crear el robot
19 robot = SerialLink([L1 L2 L3 L4 L5 L6], 'name', 'xArm6_DHmod');
20
21 % Visualización del robot con q_i=0
22 % q = [0 0 0 0 0 0];
23 % robot.plot(q);
24
25 % Parámetros dinámicos
26 robot.links(1).m = 2.177;
27 robot.links(1).r = [0.00015 0.02724 -0.01357];
28 robot.links(1).I = [0.005433 0.004684 0.0031118 9.864e-06 -2.68e-05 -0.000826936];
29
30 robot.links(2).m = 2.011;
31 robot.links(2).r = [0.0367 -0.22088 0.03356];
32 robot.links(2).I = [0.0271342 0.0053854 0.0262093 0.004736 0.00068673 -0.0047834];
33
34 robot.links(3).m = 1.725;
35 robot.links(3).r = [0.06977 0.1135 0.01163];
36 robot.links(3).I = [0.006085 0.0036652 0.0057045 -0.0015 0.0009558 0.0018091];
37
38 robot.links(4).m = 1.211;
39 robot.links(4).r = [-0.0002 0.02 -0.026];
40 robot.links(4).I = [0.0046981 0.0042541 0.00123664 -6.486e-06 -1.404e-05 -0.0002877];
41
42 robot.links(5).m = 1.206;
43 robot.links(5).r = [0.06387 0.02928 0.0035];
44 robot.links(5).I = [0.0013483 0.00175694 0.002207 -0.00042677 0.00028758 0.0001244];
45
46 robot.links(6).m = 0.170;
47 robot.links(6).r = [0 -0.00677 -0.01098];
48 robot.links(6).I = [9.3e-05 5.87e-05 0.000132 0 0 -3.6e-06];
49
50 for i = 1:6
51     robot.links(i).G = 1;
52     robot.links(i).Jm = 0;
53     robot.links(i).B = 0;
54     robot.links(i).Tc = [0 0];
55 end
56
57 % Gravedad en SI
58 grav = [0; 0; -9.81];
59
60 % Configuración articular (ejemplo)
61 q = [pi/4 pi/4 pi/4 pi/4 pi/4];
62
63 % Velocidades aleatorias dentro de ±2 rad/s
64 qd = [1.2, -0.8, 0.5, -1.0, 0.7, -0.6]; % rad/s

```

```

65
66 % Aceleraciones aleatorias dentro de ±10 rad/s²
67 qdd = [ 5.0, -3.5, 2.0, -4.0, 6.0, -2.5]; % rad/s²
68
69 % Cálculo de pares dinámicos
70 tau = robot.rne(q, qd, qdd, grav);
71 disp('Par calculado en cada articulación [Nm]:');
72 disp(tau);

```

Código 14: Cálculo de pares de los actuadores.

```

1 Par calculado en cada articulación [Nm]:
2   0.7081   -3.7259   -10.6529    -0.3223    -0.6966    -0.0006

```

Código 15: Salida del cálculo de pares de los actuadores.

Controlador

```
1 clear; clc; close all;
2
3 % Modelo del robot
4 [robot, p_fun, J_fun, T_fun, a2_val, T2_offset_val, T3_offset_val] = model();
5 robot.plot([0 0 0 0 0 0]);
6
7 % Entrada usuario
8 dlg = inputdlg( ...
9     {'Articulaciones actuales [q1 q2 q3 q4 q5 q6] en grados (matemático):', ...
10    'Posición final del efecto [x y z] en mm:', ...
11    'Entrada de movimiento', [1 70], {'0 0 0 0 0 0', '350 150 250'}});
12
13 if isempty(dlg)
14     q0_deg      = input('Introduce [q1 q2 q3 q4 q5 q6] en grados (matemático): ');
15     p_goal_mm   = input('Introduce [x y z] en mm para la posición final del efecto: ');
16 else
17     q0_deg      = str2num(dlg{1}); %#ok<ST2NM>
18     p_goal_mm   = str2num(dlg{2}); %#ok<ST2NM>
19 end
20
21 assert(numel(q0_deg)==6);
22 assert(numel(p_goal_mm)==3);
23
24 q_init      = deg2rad(q0_deg(:));    % rad
25 p_goal_m    = p_goal_mm(:) / 1000;   % m
26
27 grav = [0; 0; -9.81];
28
29 % Planificador de trayectoria con escalado replanificación
30 % para cumplir los límites
31 vel_max = deg2rad([180 180 180 180 180 180]); % rad/s
32 acc_max = deg2rad([1500 1500 1500 1500 1500 1500]); % rad/s^2
33 tau_max = [50 50 40 12 12 8]; % Nm
34
35 [Q_des, dQ_des, ddQ_des, t, P, T_final] = kinematic_planner( ...
36     robot, p_fun, J_fun, q_init, p_goal_m, vel_max, acc_max);
37
38 [Q_torque, dQ_torque, ddQ_torque, t_torque, tau_all, tau_peak_final, k_opt] = ...
39 dynamic_replanner(robot, Q_des, dQ_des, ddQ_des, t, tau_max);
40
41 % Controlador PID
42 Kp = [100 100 100 10 4 2];
43 Kd = [20 20 20 20 15 5];
44 Ki = [0 0 0 0 0 0];
45
46 [q_real, dq_real, tau_hist] = controller(robot, Q_des, dQ_des, ddQ_des, t, grav, Kp, Kd, Ki);
47
48
49 % Impresión de estados inicial/final y máximos
50 % 1) Posición articular inicial (ya en q_init)
51 q_init_rad = q_init(:)';
52 q_init_deg = rad2deg(q_init_rad);
53
54 % 2) Posición cartesiana inicial
55 T_init = T_fun(q_init(1), q_init(2), q_init(3), ...
56                 q_init(4), q_init(5), q_init(6));
57 p_init_m  = T_init(1:3,4);
58 p_init_mm = p_init_m.' * 1000;
59
60 % 3) Posición articular final (tomamos la final tras retiming)
61 q_final_rad = Q_torque(end,:);
62 q_final_deg = rad2deg(q_final_rad);
63
64 % 4) Posición cartesiana final
65 T_final_fk = T_fun(q_final_rad(1), q_final_rad(2), q_final_rad(3), ...
66                     q_final_rad(4), q_final_rad(5), q_final_rad(6));
67 p_final_m  = T_final_fk(1:3,4);
68 p_final_mm = p_final_m.' * 1000;
69
70 % 5) Tiempo total de trayectoria (tras retiming por torque)
71 T_traj = t_torque(end);
72
73 % 6) Máximos reales
```

```

74 vel_peak = max(abs(dQ_torque), [], 1);
75 acc_peak = max(abs(ddQ_torque), [], 1);
76 tau_peak = max(abs(tau_all), [], 1);
77
78 fprintf('=====\\n');
79 fprintf('          RESUMEN DE LA TRAYECTORIA\\n');
80 fprintf('=====\\n');
81
82 fprintf('\\nTiempo total de trayectoria (tras torque): %.4f s\\n', T_traj);
83
84 fprintf('\\nPosición articular inicial [rad]:\\n');
85 disp(q_init_rad);
86 fprintf('Posición articular inicial [deg]:\\n');
87 disp(q_init_deg);
88
89 fprintf('Posición cartesiana inicial [mm]:\\n');
90 disp(p_init_mm);
91
92 fprintf('Posición articular final [rad]:\\n');
93 disp(q_final_rad);
94 fprintf('Posición articular final [deg]:\\n');
95 disp(q_final_deg);
96
97 fprintf('Posición cartesiana final [mm]:\\n');
98 disp(p_final_mm);
99
100 % Máximos reales
101 vel_peak = max(abs(dQ_torque), [], 1);
102 acc_peak = max(abs(ddQ_torque), [], 1);
103 tau_peak = max(abs(tau_all), [], 1);
104
105 fprintf('=====\\n');
106 fprintf('      MÁXIMOS DE LA TRAYECTORIA\\n');
107 fprintf('=====\\n');
108
109 fprintf('Velocidades máximas alcanzadas en trayectoria calculada [rad/s]:\\n');
110 disp(vel_peak);
111 fprintf('\\nVelocidades máximas teóricas [rad/s]:\\n');
112 disp(vel_max);
113
114 fprintf('Aceleraciones máximas alcanzadas en trayectoria calculada [rad/s^2]:\\n');
115 disp(acc_peak);
116 fprintf('\\nAceleraciones máximas teóricas [rad/s^2]:\\n');
117 disp(acc_max);
118
119 fprintf('Torques máximos alcanzados en trayectoria calculada [Nm]:\\n');
120 disp(tau_peak);
121 fprintf('\\nTorques máximos teóricos [Nm]:\\n');
122 disp(tau_max);
123
124 fprintf('=====\\n\\n');
125
126 % Gráficas completas
127 % 1) Trayectoria articular óptima
128 figure;
129 plot(t_torque, Q_torque, 'LineWidth', 1.4);
130 title('Trayectoria articular óptima (Q_torque)');
131 xlabel('Tiempo [s]');
132 ylabel('q [rad]');
133 grid on;
134 legend('q1','q2','q3','q4','q5','q6');
135
136 % 2) Velocidades articulares
137 figure;
138 plot(t_torque, dQ_torque, 'LineWidth', 1.4);
139 title('Velocidades articulares óptimas');
140 xlabel('Tiempo [s]');
141 ylabel('dq/dt [rad/s]');
142 grid on;
143 legend('dq1','dq2','dq3','dq4','dq5','dq6');
144
145 % 3) Aceleraciones articulares
146 figure;
147 plot(t_torque, ddQ_torque, 'LineWidth', 1.4);
148 title('Aceleraciones articulares óptimas');

```

```

149 xlabel('Tiempo [s]');
150 ylabel('d^2q/dt^2 [rad/s^2]');
151 grid on;
152 legend('ddq1','ddq2','ddq3','ddq4','ddq5','ddq6');
153
154 % 4) Trayectoria cartesiana
155 figure;
156 plot3(P(:,1), P(:,2), P(:,3), 'LineWidth', 1.5);
157 grid on;
158 xlabel('X [mm]');
159 ylabel('Y [mm]');
160 zlabel('Z [mm]');
161 title('Trayectoria cartesiana (quintic)');
162
163 % 5) Par articular a lo largo del tiempo
164 %
165 figure;
166 plot(t_torque, tau_all, 'LineWidth', 1.4);
167 title('Par articular \tau(t)');
168 xlabel('Tiempo [s]');
169 ylabel('Torque [Nm]');
170 grid on;
171 legend('tau1','tau2','tau3','tau4','tau5','tau6');
172
173 % Límites de par
174 hold on;
175 for i = 1:6
176     yline( tau_max(i), '--', 'Color', [0.5 0.5 0.5]);
177     yline(-tau_max(i), '--', 'Color', [0.5 0.5 0.5]);
178 end
179 hold off;
180 %
181 figure;
182 plot(t_torque, tau_all, 'LineWidth', 1.4);
183 title('Par articular \tau(t)');
184 xlabel('Tiempo [s]');
185 ylabel('Torque [Nm]');
186 grid on;
187 legend('tau1','tau2','tau3','tau4','tau5','tau6');

```

Código 16: Programa principal.

```

1 function [robot, p_fun, J_fun, T_fun, a2_val, T2_offset_val, T3_offset_val] = model()
2
3     % Parámetros geométricos
4     T2_offset_val = deg2rad(-79.34995); % Eslabón 1
5     T3_offset_val = deg2rad(79.34995); % Eslabón 2
6     a2_val         = 0.28948866; % metros % Eslabón 3
7
8     % Construcción del robot (DH modificado)
9     L1 = Link([0 0.267 0 0 0 0], 'modified'); % Eslabón 1
10    L2 = Link([0 0 0 -pi/2 0 T2_offset_val], 'modified'); % Eslabón 2
11    L3 = Link([0 0 a2_val 0 0 T3_offset_val], 'modified'); % Eslabón 3
12    L4 = Link([0 0.3425 0.0775 -pi/2 0 0], 'modified'); % Eslabón 4
13    L5 = Link([0 0 0 pi/2 0 0], 'modified'); % Eslabón 5
14    L6 = Link([0 0.097 0.076 -pi/2 0 0], 'modified'); % Eslabón 6
15
16     % Crear el robot
17     robot = SerialLink([L1 L2 L3 L4 L5 L6], 'name', 'xArm6');
18
19     % Definir los límites articulares
20     robot qlim = deg2rad([
21         -360 360 % q1
22         -117 116 % q2
23         -219 10 % q3
24         -360 360 % q4
25         -97 180 % q5
26         -360 360 % q6
27     ]);
28
29     % Parámetros dinámicos
30     robot.links(1).m = 2.177;
31     robot.links(1).r = [0.00015 0.02724 -0.01357];
32     robot.links(1).I = [0.005433 0.004684 0.0031118 9.864e-06 -2.68e-05 -0.000826936];
33

```

```

34 robot.links(2).m = 2.011;
35 robot.links(2).r = [0.0367 -0.22088 0.03356];
36 robot.links(2).I = [0.0271342 0.0053854 0.0262093 0.004736 0.00068673 -0.0047834];
37
38 robot.links(3).m = 1.725;
39 robot.links(3).r = [0.06977 0.1135 0.01163];
40 robot.links(3).I = [0.006085 0.0036652 0.0057045 -0.0015 0.0009558 0.0018091];
41
42 robot.links(4).m = 1.211;
43 robot.links(4).r = [-0.0002 0.02 -0.026];
44 robot.links(4).I = [0.0046981 0.0042541 0.00123664 -6.486e-06 -1.404e-05 -0.0002877];
45
46 robot.links(5).m = 1.206;
47 robot.links(5).r = [0.06387 0.02928 0.0035];
48 robot.links(5).I = [0.0013483 0.00175694 0.002207 -0.00042677 0.00028758 0.0001244];
49
50 robot.links(6).m = 0.170;
51 robot.links(6).r = [0 -0.00677 -0.01098];
52 robot.links(6).I = [9.3e-05 5.87e-05 0.000132 0 0 -3.6e-06];
53
54 for i = 1:6
55     robot.links(i).G = 1;
56     robot.links(i).Jm = 0;
57     robot.links(i).B = 0;
58     robot.links(i).Tc = [0 0];
59 end
60
61 % Generación de funciones numéricas
62 syms q1 q2 q3 q4 q5 q6 real
63
64 % Transformaciones simbólicas
65 T01 = L1.A(q1).T;
66 T12 = L2.A(q2).T;
67 T23 = L3.A(q3).T;
68 T34 = L4.A(q4).T;
69 T45 = L5.A(q5).T;
70 T56 = L6.A(q6).T;
71
72 T06 = simplify(T01*T12*T23*T34*T45*T56);
73 p06 = T06(1:3,4);
74 Jv = jacobian(p06, [q1 q2 q3 q4 q5 q6]);
75
76 % Funciones numéricas
77 p_fun = matlabFunction(p06, 'Vars', {q1,q2,q3,q4,q5,q6});
78 J_fun = matlabFunction(Jv, 'Vars', {q1,q2,q3,q4,q5,q6});
79 T_fun = matlabFunction(T06, 'Vars', {q1,q2,q3,q4,q5,q6});
80
81 end

```

Código 17: Modelado.

```

1 function [Q_des, dQ_des, ddQ_des, t, P, T_final] = kinematic_planner( ...
2     robot, p_fun, J_fun, T_fun, q_init, p_goal_m, vel_max, acc_max)
3
4 % Límites articulares (en deg)
5 q_min = robot.qlim(:,1);
6 q_max = robot.qlim(:,2);
7 q_mid = (q_min + q_max)/2;
8
9 % Verificación de alcanzabilidad
10 reach_max_m = 0.7; % 700 mm
11 if norm(p.goal_m) > reach_max_m
12     fprintf('Objetivo fuera del alcance -> NO realizable\n');
13     Q_des = []; dQ_des = []; ddQ_des = []; t = []; P = []; T_final = [];
14     return
15 end
16
17 % IK numérica
18 tol      = 1e-6;
19 maxIter = 80;
20 K        = 0.005;
21
22 q_goal = newtonIK(p.goal_m, q_init, p_fun, J_fun, ...
23                     q_min, q_max, q_mid, K, tol, maxIter);
24

```

```

25 if any(isnan(q_goal))
26     error('IK no encontró solución válida.');
27 end
28
29 % Trayectoria quíntica base + escalado
30 n = 300;
31 T0 = 1.0; % tiempo base mínimo (se reescalara luego)
32 t0 = linspace(0, T0, n);
33
34 % Quíntica base entre q_init y q_goal
35 Q0 = quinticTraj(q_init.', q_goal.', T0, t0);
36
37 % Escalado óptimo de tiempo para cumplir vel/acc (estilo TOTG)
38 [Q_des, t, T_final] = scale_traj(Q0, t0, vel_max, acc_max);
39
40 % Derivadas numéricas
41 dt = t(2) - t(1);
42 dQ_des = diff(Q_des) / dt;
43 ddQ_des = diff(dQ_des) / dt;
44
45 % Trayectoria cartesiana del efecto
46 P = zeros(numel(t), 3);
47 for k = 1:numel(t)
48     Tnow = T_fun(Q_des(k,1), Q_des(k,2), Q_des(k,3), ...
49                   Q_des(k,4), Q_des(k,5), Q_des(k,6));
50     p_m = Tnow(1:3,4); % metros
51     P(k,:) = (p_m * 1000).'; % mm para graficar
52 end
53
54 end
55
56 % Funciones auxiliares
57 function q_sol = newtonIK(p_d_m, q_seed, p_fun, J_fun, ...
58                             q_min, q_max, q_mid, K, tol, maxIter)
59
60 qk = q_seed(:);
61 for it = 1:maxIter %#ok<NASGU>
62     p_now_m = p_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6));
63     J_now = J_fun(qk(1), qk(2), qk(3), qk(4), qk(5), qk(6));
64
65     e = p_d_m - p_now_m;
66     if norm(e) < tol, break; end
67
68     gradH = 2 * (qk - q_mid) ./ ((q_max - q_min).^2);
69     dq = pinv(J_now) * e - K * gradH;
70
71     qk = qk + dq;
72 end
73 q_sol = qk;
74 end
75
76 function Q = quinticTraj(q0, qf, T, t)
77 n = numel(t);
78 Q = zeros(n, numel(q0));
79 for i = 1:numel(q0)
80     a0 = q0(i); a1 = 0; a2 = 0;
81     a3 = 10*(qf(i)-q0(i))/T^3;
82     a4 = -15*(qf(i)-q0(i))/T^4;
83     a5 = 6*(qf(i)-q0(i))/T^5;
84     Q(:,i) = a0 + a1*t + a2*t.^2 + a3*t.^3 + a4*t.^4 + a5*t.^5;
85 end
86 end
87
88 function [Q_out, t_out, T_scaled] = scale_traj(Q, t, vel_max, acc_max)
89 dt = t(2) - t(1);
90 dQ = diff(Q)/dt;
91 ddQ = diff(dQ)/dt;
92
93 % Factor mínimo de escalado para cumplir límites
94 scale_v = max(max(abs(dQ) ./ vel_max));
95 scale_a = max(max(abs(ddQ) ./ acc_max));
96 scale = max([1, scale_v, sqrt(scale_a)]);
97
98 if scale > 1
99     T_scaled = t(end) * scale;

```

```

100 t_out      = linspace(0, T_scaled, numel(t));
101
102 Q_out = zeros(size(Q));
103 q0    = Q(1,:);
104 qf    = Q(end,:);
105 for i = 1:size(Q,2)
106     a0 = q0(i); a1 = 0; a2 = 0;
107     a3 = 10*(qf(i)-q0(i))/T_scaled^3;
108     a4 = -15*(qf(i)-q0(i))/T_scaled^4;
109     a5 = 6*(qf(i)-q0(i))/T_scaled^5;
110     Q_out(:,i) = a0 + a1*t_out + a2*t_out.^2 + a3*t_out.^3 + ...
111         a4*t_out.^4 + a5*t_out.^5;
112 end
113 else
114     Q_out = Q;
115     t_out = t;
116     T_scaled = t(end);
117 end
118 end

```

Código 18: Planificador cinemático.

```

1 function [Q_torque, dQ_torque, ddQ_torque, t_torque, tau_all, tau_peak_final, k_opt, did_retime] = ...
2     dynamic_replanner(robot, Q_des, dQ_des, ddQ_des, t, tau_max)
3
4 % Ajuste de longitudes
5 % ddQ tiene N-2 muestras --> recortamos todo a esa longitud
6 Ndd = size(ddQ_des,1);
7 Q_des = Q_des(1:Ndd,:);
8 dQ_des = dQ_des(1:Ndd,:);
9 ddQ_des = ddQ_des(1:Ndd,:);
10 t = t(1:Ndd);
11
12 grav = [0;0;-9.81];
13
14 % Parámetros de búsqueda
15 k_low = 1.0;
16 k_high = 8.0;
17 tol = 1e-3;
18 max_iter = 20;
19
20 % Función interna para evaluar torque con factor k
21 function [tau_peak, tau_all, t_new, dQ_new, ddQ_new] = ...
22     eval_tau(robot, Q, dQ, ddQ, t, k, grav)
23
24 % Escalado temporal
25 t_new = t * k;
26 dQ_new = dQ / k;
27 ddQ_new = ddQ / (k^2);
28
29 % Evaluación del torque
30 N = size(Q,1);
31 tau_all = zeros(N,6);
32 for j = 1:N
33     tau_all(j,:) = robot.rne(Q(j,:), dQ_new(j,:), ddQ_new(j,:), grav);
34 end
35
36 tau_peak = max(abs(tau_all),[],1);
37 end
38
39 % Evaluar torque original (k = 1)
40 [tau_peak, ~, ~, ~, ~] = eval_tau(robot, Q_des, dQ_des, ddQ_des, t, 1.0, grav);
41
42 % Caso 1: no hay retiming
43 if all(tau_peak <= tau_max)
44     disp('La trayectoria original ya cumple límites de par.');
45     k_opt = 1.0;
46
47 % Evaluar torque original para graficarlo SIEMPRE
48 [tau_peak_final, tau_all, t_torque, dQ_torque, ddQ_torque] = ...
49     eval_tau(robot, Q_des, dQ_des, ddQ_des, t, 1.0, grav);
50
51 Q_torque = Q_des;
52 did_retime = false;
53 return;

```

```

54
55
56 % Caso 2: sí hay retiming
57 disp('Buscando factor de escalado óptimo...');

58
59 for iter = 1:max_iter
60     k_mid = (k_low + k_high) / 2;
61
62     tau_peak_mid = eval_tau(robot, Q_des, dQ_des, ddQ_des, t, k_mid, grav);
63
64     if all(tau_peak_mid <= tau_max)
65         k_high = k_mid;    % podemos reducir k
66     else
67         k_low = k_mid;    % necesitamos aumentar k
68     end
69
70     if abs(k_high - k_low) < tol
71         break;
72     end
73 end
74
75 k_opt = k_high;
76
77 % Construir trayectoria final
78 [tau_peak_final, tau_all, t_torque, dQ_torque, ddQ_torque] = ...
79     eval_tau(robot, Q_des, dQ_des, ddQ_des, t, k_opt, grav);
80
81 Q_torque = Q_des;
82 did_retime = true;
83
84 % Impresiones SOLO si hubo retiming
85 fprintf('\n==== RESULTADOS DEL RETIMING ====\n');
86 fprintf('Factor de escalado óptimo k = %.4f\n', k_opt);
87 fprintf('Nuevo tiempo total de trayectoria: %.4f s\n', t_torque(end));
88 fprintf('Tau máximo por articulación después del retiming:\n');
89 disp(tau_peak_final);
90
91 end

```

Código 19: Replanificador dinámico.

```

1 function [q_real_hist, dq_real_hist, tau_hist] = controller( ...
2     robot, Q_des, dQ_des, ddQ_des, t, grav, Kp, Kd, Ki)
3
4 % Ajuste de longitudes
5 N = size(dQ_des,1);
6
7 Q_des    = Q_des(1:N,:);
8 dQ_des   = dQ_des(1:N,:);
9 ddQ_des = ddQ_des(1:N,:);
10 dt = t(2) - t(1);
11 t      = t(1:N);
12
13 % Históricos
14 q_real_hist = zeros(N,6);
15 dq_real_hist = zeros(N,6);
16 tau_hist     = zeros(N,6);
17
18 % Estado inicial
19 q_real = Q_des(:,1);
20 dq_real = dQ_des(:,1);
21
22 % Integral del error
23 e_int = zeros(1,6);
24
25 % Perturbación externa (torque)
26 %tau_pert = @(k) 0.5 * sin(0.1*k) * ones(1,6);
27 tau_pert = @(k) 0.2 * randn(1,6);
28
29 robot.gravity = grav(:,1);
30
31 for k = 1:N
32
33     % Señales deseadas
34     q_d    = Q_des(k,:);

```

```

35 dq_d = dQ_des(k,:);
36 ddq_d = ddQ_des(k,:);
37
38 % Error
39 e = q_d - q_real;
40 edot = dq_d - dq_real;
41 e_int = e_int + e*dt;
42
43 % PID
44 ddq_ref = ddq_d + Kp.*e + Kd.*edot + Ki.*e_int;
45
46 % Torque PID
47 tau_pid = robot.rne(q_real, dq_real, ddq_ref);
48
49 % Perturbación física
50 tau_total = tau_pid + tau_pert(k);
51
52 % Dinámica directa
53 M = robot.inertia(q_real);
54 C = robot.coriolis(q_real, dq_real);
55 g = robot.gravload(q_real);
56
57 ddq_real = M \ (tau_total' - C*dq_real' - g');
58
59 % Integración
60 dq_real = dq_real + ddq_real'*dt;
61 q_real = q_real + dq_real*dt;
62
63 % Guardar
64 q_real_hist(k,:) = q_real;
65 dq_real_hist(k,:) = dq_real;
66 tau_hist(k,:) = tau_total;
67 end
68
69 % Gráficas
70 figure;
71 plot(t, Q_des, '--', 'LineWidth', 1.2); hold on;
72 plot(t, q_real_hist, 'LineWidth', 1.4);
73 title('Trayectorias deseadas vs reales (todas las articulaciones)');
74 xlabel('Tiempo [s]'); ylabel('Ángulo [rad]');
75 legend('q1_d','q2_d','q3_d','q4_d','q5_d','q6_d', ...
76 'q1_r','q2_r','q3_r','q4_r','q5_r','q6_r');
77 grid on;
78
79 % Una figura por articulación
80 for j = 1:6
81     figure;
82     plot(t, Q_des(:,j), '--', 'LineWidth', 1.2); hold on;
83     plot(t, q_real_hist(:,j), 'LineWidth', 1.4);
84     title(['Trayectoria - Articulación ', num2str(j)]);
85     xlabel('Tiempo [s]'); ylabel('Ángulo [rad]');
86     legend('Deseada','Real');
87     grid on;
88 end
89
90 end

```

Código 20: Controlador.

Programación básica de MoveIt2 con un nodo en C++

```
1 #include <rclcpp/rclcpp.hpp>
2 #include <rclcpp_action/rclcpp_action.hpp>
3 #include <moveit_msgs/action/move_group.hpp>
4 #include <geometry_msgs/msg/pose_stamped.hpp>
5 #include <vector>
6 #include <string>
7 #include <cmath>
8
9 using MoveGroup = moveit_msgs::action::MoveGroup;
10 using GoalHandleMoveGroup = rclcpp_action::ClientGoalHandle<MoveGroup>;
11
12 class XArm6ActionClient : public rclcpp::Node
13 {
14 public:
15     XArm6ActionClient()
16         : Node("xarm6_action_client")
17     {
18         // Cliente de la acción /move_action de MoveIt2
19         client_ = rclcpp_action::create_client<MoveGroup>(this, "/move_action");
20
21         RCLCPP_INFO(this->get_logger(), "Esperando al action server /move_action...");
22         client_->wait_for_action_server();
23         RCLCPP_INFO(this->get_logger(), "Action server disponible.");
24
25         run_sequence();
26     }
27
28 private:
29     rclcpp_action::Client<MoveGroup>::SharedPtr client_;
30
31     // Función para crear goals articulares
32     moveit_msgs::msg::Constraints make_joint_goal(
33         const std::vector<std::string>& names,
34         const std::vector<double>& positions_deg)
35     {
36         moveit_msgs::msg::Constraints constraints;
37
38         for (size_t i = 0; i < names.size(); ++i)
39         {
40             moveit_msgs::msg::JointConstraint jc;
41             jc.joint_name = names[i];
42             jc.position = positions_deg[i] * M_PI / 180.0; // grados --> radianes
43             jc.weight = 1.0;
44             constraints.joint_constraints.push_back(jc);
45         }
46
47         return constraints;
48     }
49
50     // Enviar un goal con velocidad y aceleración configurables
51     void send_goal_and_wait(
52         const moveit_msgs::msg::Constraints& constraints,
53         double vel_scaling,
54         double acc_scaling)
55     {
56         MoveGroup::Goal goal;
57         goal.request.group_name = "xarm6";
58
59         // Parámetros dinámicos
60         goal.request.max_velocity_scaling_factor = vel_scaling;
61         goal.request.max_acceleration_scaling_factor = acc_scaling;
62
63         goal.request.goal_constraints.push_back(constraints);
64
65         auto future_goal = client_->async_send_goal(goal);
66
67         if (rclcpp::spin_until_future_complete(this->get_node_base_interface(), future_goal)
68             != rclcpp::FutureReturnCode::SUCCESS)
69         {
70             RCLCPP_ERROR(this->get_logger(), "Error enviando goal");
71             return;
72         }
73     }
```

```

74     auto goal_handle = future_goal.get();
75     if (!goal_handle)
76     {
77         RCLCPP_ERROR(this->get_logger(), "Goal rechazada");
78         return;
79     }
80
81     auto result_future = client_->async_get_result(goal_handle);
82     rclcpp::spin_until_future_complete(this->get_node_base_interface(), result_future);
83
84     RCLCPP_INFO(this->get_logger(), "Goal completada.");
85 }
86
87 // Secuencia completa
88 void run_sequence()
89 {
90     // 1) Goal articular
91     auto goal1 = make_joint_goal(
92         {"joint1", "joint2", "joint3", "joint4", "joint5", "joint6"},
93         {-59.0, -23.0, -40.0, 0.0, 63.0, -59.0}
94     );
95
96     RCLCPP_INFO(this->get_logger(), "Enviando goal 1...");
97     send_goal_and_wait(goal1, 0.3, 0.2); // velocidad y aceleración personalizadas
98
99     // 2) Volver a cero
100    auto goal2 = make_joint_goal(
101        {"joint1", "joint2", "joint3", "joint4", "joint5", "joint6"},
102        {0, 0, 0, 0, 0, 0}
103    );
104
105    RCLCPP_INFO(this->get_logger(), "Enviando goal 2 (volver a cero)..."); 
106    send_goal_and_wait(goal2, 0.5, 0.3);
107
108    RCLCPP_INFO(this->get_logger(), "Secuencia completa.");
109 }
110
111 int main(int argc, char **argv)
112 {
113     rclcpp::init(argc, argv);
114     auto node = std::make_shared<XArm6ActionClient>();
115     rclcpp::shutdown();
116     return 0;
117 }
118 }
```

Código 21: Nodo C++ para controlar el robot con MoveIt2.

Referencias adicionales al material del Campus Virtual

- [1] diagrams.net. *draw.io – Herramienta de diagramación en línea*. Accedido: 2 de diciembre de 2025. 2025. URL: <https://www.drawio.com/>.
- [2] Inc. The MathWorks. *MATLAB Online*. <https://matlab.mathworks.com/>. Versión en línea de MATLAB accesible vía navegador. 2025.
- [3] zDynamics. *ROBOT DELTA: CONTROL DE LAS ARTICULACIONES*. <https://www.youtube.com/watch?v=m0zzhz-36Bw>. Accedido: 2 de diciembre de 2025. 2023.
- [4] UFactory. *xArm 6 Collaborative Robot*. <https://www.ufactory.us/product/ufactory-xarm-6>. Accedido: 8 de diciembre de 2025. 2025.
- [5] Steven Macenski et al. «Robot Operating System 2: Design, architecture, and uses in the wild». En: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: <https://www.science.org/doi/abs/10.1126/scirobotics.abm6074>.
- [6] Open Robotics y Gazebo Community. *Gazebo Harmonic*. <https://gazebosim.org/docs/harmonic>. Accedido: 8 de diciembre de 2025. 2025.
- [7] Open Robotics. *ROS 2 Jazzy Jalisco*. <https://docs.ros.org/en/jazzy/>. Accedido: 8 de diciembre de 2025. 2024.
- [8] xArm-Developer. *xArm ROS 2*. https://github.com/xArm-Developer/xarm_ros2. Repositorio en GitHub. Accedido: 8 de diciembre de 2025. 2025.
- [9] UFactory. *UFactory Docs, xArm User Manual V2.0.0*. Accedido: 8 de diciembre de 2025. 2023.
- [10] UFactory. *xArm Collaborative Robot*. <https://www.ufactory.cc/xarm-collaborative-robot/>. Accedido: 8 de diciembre de 2025. 2025.
- [11] Maroš Majchrák et al. «Analysis of harmonic gearbox tooth contact pressure». En: *IOP Conference Series: Materials Science and Engineering* 659 (oct. de 2019), pág. 012068. DOI: 10.1088/1757-899X/659/1/012068.
- [12] Antonio Barrientos et al. *Fundamentos de robótica*. spa. 2.^a ed. Madrid: McGraw-Hill Interamericana de España, 2007. ISBN: 9788448156367.
- [13] Intel Corporation. *Intel RealSense Depth Camera D435*. Datasheet y documentación técnica. 2018. URL: <https://www.intelrealsense.com/depth-camera-d435/>.
- [14] International Organization for Standardization. *ISO 9409-1:2004, Mechanical interfaces for industrial robots – Part 1: Plates for connection of end-effectors*. Norma internacional sobre interfaces mecánicas de robots industriales. 2004. URL: <https://www.iso.org/standard/35495.html>.
- [15] UFactory. *xArm Studio User Manual*. <https://www.ufactory.us/ufactory-studio>. Accedido: 9 de diciembre de 2025. 2025.
- [16] Peter I. Corke. «A Simple and Systematic Approach to Assigning Denavit–Hartenberg Parameters». En: *Proceedings of the IEEE International Conference on Robotics and Automation* (1996), págs. 1834–1839. URL: https://petercorke.com/doc/simple_systematic.pdf.

- [17] Peter Corke. *Robotics Toolbox*. 2025. URL: <https://petercorke.com/toolboxes/robotics-toolbox/>.
- [18] John J. Craig. *Introduction to Robotics: Mechanics and Control*. 3rd. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.
- [19] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control – The Computer Control of Robot Manipulators*. Inf. téc. NASA Technical Paper 1800. Washington, D.C.: NASA, 1986. URL: <https://ntrs.nasa.gov/api/citations/19860018481/downloads/19860018481.pdf>.
- [20] MoveIt Contributors. *MoveIt 2*. <https://moveit.ros.org>. Accessed: 2025-01-15. 2024.
- [21] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. London: Springer, 2009. ISBN: 978-1-84628-641-4.
- [22] Yoshihiko Nakamura. *Advanced Robotics: Redundancy and Optimization*. Addison-Wesley, 1991.
- [23] Samuel R. Buss y Jin-Su Kim. «Selectively damped least squares for inverse kinematics». En: *Journal of Graphics Tools* 10.3 (2004), págs. 37-49.
- [24] Charles W. Wampler. «Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods». En: *IEEE Transactions on Systems, Man, and Cybernetics* 16.1 (1986), págs. 93-101.
- [25] ros-controls. *ros2_control: A framework for robot control in ROS 2*. https://github.com/ros-controls/ros2_control. Accessed: December 17, 2025.
- [26] Mark W. Spong, Seth Hutchinson y M. Vidyasagar. *Robot Modeling and Control*. Wiley, 2006. ISBN: 978-0471649908.
- [27] James E Bobrow, Steven Dubowsky y John S Gibson. «Time-optimal control of robotic manipulators along specified paths». En: *The International Journal of Robotics Research* 4.3 (1985), págs. 3-17.
- [28] Kang G Shin y N McKay. «Minimum-time control of robotic manipulators with geometric path constraints». En: *IEEE Transactions on Automatic Control* 31.6 (1985), págs. 491-498.
- [29] Quang-Cuong Pham. «A new approach to time-optimal path parameterization based on reachability analysis». En: *Robotics: Science and Systems (RSS)*. 2018.
- [30] Damien Verschueren et al. «Time-optimal path tracking for robots: A convex optimization approach». En: *IEEE Transactions on Automatic Control* 54.10 (2009), págs. 2318-2327.
- [31] Tobias Kunz y Mike Stilman. «Time-Optimal Trajectory Generation for Path Following with Bounded Acceleration and Velocity». En: *Proceedings of Robotics: Science and Systems (RSS)*. Sydney, Australia, 2012.
- [32] Quang-Cuong Pham. «A New Approach to Time-Optimal Path Parameterization based on Reachability Analysis». En: *IEEE Transactions on Robotics* 34.3 (2018), págs. 645-659.