

Máster Universitario en Robótica y Automatización

*Trabajo Fin de Máster*

**“Integración del robot colaborativo IIWA  
en un sistema de evaluación objetiva de  
espasticidad”**

---

Ana Casanova Jaquete

Tutor

Alberto Jardón Huete

Leganés, 24 de Noviembre de 2020



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**



## **RESUMEN**

En la actualidad, el campo de la robótica de rehabilitación se ha centrado en el desarrollo de sistemas para la intervención durante el proceso de recuperación del paciente. Sin embargo, una parte fundamental del ciclo de rehabilitación consiste en evaluar de forma correcta el nivel de afección de los trastornos neuromotores. Uno de los desórdenes neurológicos más difíciles de estimar es la espasticidad, un trastorno sensorimotor que produce rigidez muscular dependiente de la velocidad de movimiento. Su evaluación se realiza tradicionalmente de manera manual, llevando a resultados poco precisos y con una alta dependencia del criterio del terapeuta.

En este contexto, el proyecto Roboespas presenta un sistema de evaluación de la espasticidad basado en el uso de un robot colaborativo, y que pretende estimar un descriptor espástico que aporte objetividad al diagnóstico. Este sistema se ha enfocado a la extremidad superior, y consta de tres bloques: la movilización del brazo del paciente siguiendo una trayectoria personalizada, el modelado biomecánico del usuario, y la estimación del descriptor espástico utilizando los dos anteriores.

Este trabajo fin de máster se centra en el desarrollo, implementación e integración del primer bloque del sistema. El proceso de movilización asistido por robot se ha dividido en dos procedimientos consecutivos: la generación de una trayectoria adaptada a las características del paciente, y la posterior reproducción de la misma. Por un lado, la personalización de la trayectoria se ha conseguido capturando el movimiento natural de flexo-extensión de codo del paciente, y ajustando esa captura para cumplir unos objetivos de continuidad, suavidad y forma requeridos; que aseguran tanto la comodidad y seguridad del usuario, como la correcta movilización de la extremidad.

Por otro lado, para la correcta reproducción a diferentes velocidades de las trayectorias ajustadas se han utilizado dos interfaces de comunicación con el robot: iiwa\_stack y FRI. Utilizando la primera interfaz, se han probado dos posibles tipos de control: en posición articular y en velocidad articular; y utilizando la segunda, se han llevado a cabo experimentos utilizando un control en posición articular con y sin realimentación en posición cartesiana. Finalmente, se han analizado los datos obtenidos con estos cuatro modos de reproducción de trayectorias y se han discutido sus ventajas e inconvenientes.

**Palabras clave:** Espasticidad, Extremidad superior, Robot colaborativo, IIWA



## ABSTRACT

Nowadays, the rehabilitation robotics field is mainly focused on developing intervention systems for improving the patient's recovery process. However, a fundamental part of the rehabilitation cycle is to correctly assess neuromotor disorders. One of the most difficult neurological dysfunctions to estimate is spasticity, a sensorimotor disorder that produces velocity-dependent muscle stiffness. Its assessment is traditionally carried out manually by clinicians, leading to low-resolution results with a high dependency on the therapist criteria.

In this context, Roboespas presents a spasticity assessment system by means of a collaborative robot, which aims to obtain a spasticity descriptor that provides objectiveness to its diagnosis. This system has focused on the upper limb, and consists of three blocks: the mobilization of the patient's arm following a personalized trajectory, the biomechanical modeling of the user, and the estimation of the spastic descriptor using the previous two.

This master's thesis is centered on the development, implementation, and integration of the first block of the system. The robot-assisted mobilization process has been divided into two consecutive procedures: the generation of a trajectory adapted to the patient's characteristics, and its subsequent reproduction. On the one hand, the personalization of the trajectory has been achieved by capturing the natural elbow flexo-extension movement of the patient, and adjusting that capture to meet the required continuity, smoothness and shape objectives for the trajectory; which ensures both the comfort and safety of the user, and the correct mobilization of the limb.

On the other hand, for the correct reproduction of the adjusted trajectories at different speeds, two communication interfaces have been tested: iiwa\_stack and FRI. Regarding the first one, two possible command modes have been examined: joint position and joint velocity; and regarding the second, executions using joint position commands with and without feedback have been carried out. Finally, the data obtained with these four trajectory reproduction modes have been analyzed and their advantages and disadvantages have been discussed.

**Keywords:** Spasticity, Upper limb, Collaborative robot, IIWA



# ÍNDICE GENERAL

1. INTRODUCCIÓN . . . . .	1
1.1. Diagnóstico clínico tradicional . . . . .	1
1.2. Espasticidad: definición, causas y tratamiento. . . . .	2
1.3. Escalas clínicas de evaluación de espasticidad . . . . .	3
1.4. Uso de sistemas instrumentados para la evaluación objetiva de espasticidad . .	4
1.5. Proyecto Roboespas . . . . .	5
1.6. Objetivos del trabajo . . . . .	6
1.7. Organización del documento . . . . .	6
2. FUNDAMENTOS TEÓRICOS . . . . .	8
2.1. Control cinemático . . . . .	9
2.1.1. Trayectorias. . . . .	9
2.2. Screw theory: Principios matemáticos . . . . .	11
2.2.1. Screw Theory: Notación . . . . .	14
2.2.2. Cinemática directa . . . . .	16
2.2.3. Cinemática inversa . . . . .	16
2.2.4. Cinemática diferencial . . . . .	17
2.2.5. Cinemática diferencial inversa . . . . .	18
3. ROBOESPAS: DESCRIPCIÓN DE SUBSISTEMAS E INTEGRACIÓN . . . . .	19
3.1. Movilización pasiva asistida por robot . . . . .	21
3.2. Modelado biomecánico del brazo del paciente . . . . .	23
3.3. Estimación del descriptor espástico . . . . .	25
3.4. Integración de subsistemas . . . . .	26
4. SISTEMA DE MOVILIZACIÓN ASISTIDA POR ROBOT . . . . .	29
4.1. Generación de la trayectoria personalizada . . . . .	30
4.1.1. Captura de la trayectoria inicial. . . . .	31
4.1.2. Ajuste de la trayectoria . . . . .	34
4.2. Reproducción de trayectorias . . . . .	54
4.2.1. Opción 1: iiwa_stack. . . . .	56

4.2.2. Opción 2: Fast Robot Interface . . . . .	63
4.3. Captura de datos generados . . . . .	70
4.3.1. Datos proporcionados por el robot . . . . .	70
4.3.2. Datos proporcionados por los sensores Trigno . . . . .	71
4.3.3. Datos proporcionados por el sensor fuerza-par . . . . .	71
4.3.4. Otros datos incluidos en la base de datos . . . . .	72
5. ANÁLISIS DE RESULTADOS . . . . .	73
5.1. Retardo entre la trayectoria comandada y la ejecutada . . . . .	74
5.2. Error de precisión entre la trayectoria comandada y la ejecutada . . . . .	78
5.3. Error de repetibilidad: Diferencia entre las ejecuciones y la ejecución media .	83
5.4. Comparativa iiwa_stack y FRI . . . . .	86
6. CONCLUSIONES Y TRABAJO FUTURO . . . . .	88
A. PUESTA EN MARCHA DEL PAQUETE IIWA_STACK . . . . .	90
B. PUESTA EN MARCHA DE FAST ROBOT INTERFACE . . . . .	93
C. ESQUEMA DEL PAQUETE DE ROS DESARROLLADO . . . . .	95
D. ESQUEMA UML DEL PAQUETE DE MATLAB DESARROLLADO. . . . .	98
E. ESQUEMA DE LA INTERACCIÓN ENTRE MATLAB Y ROS . . . . .	101
F. INTERFAZ DE USUARIO. . . . .	103
BIBLIOGRAFÍA . . . . .	106



## ÍNDICE DE FIGURAS

2.1	Soportes de Cardán como representación de los ángulos de Euler XYZ . . . . .	10
2.2	Homeomorfismo de una variedad $\mathbb{R}^3$ a un subconjunto $\mathbb{R}^2$ . . . . .	12
2.3	Espacio de rotación 2R como grupo de Lie . . . . .	12
2.4	Algebra de Lie del grupo de Lie del espacio de configuraciones 2R . . . . .	13
2.5	CD y CI de un robot 2R con grupos de Lie . . . . .	14
3.1	Subsistemas de Roboespas . . . . .	19
3.2	Movilización asistida por robot . . . . .	21
3.3	Modelo biomecánico de extremidad superior . . . . .	24
3.4	Bloque 2: Modelo biomecánico . . . . .	25
3.5	Bloque 3: Estimación del descriptor espástico . . . . .	26
3.6	Integración de los subsistemas de Roboespas . . . . .	27
4.1	Bloque 1: Movilización asistida por robot . . . . .	29
4.2	Sistemas de coordenadas durante la interacción . . . . .	32
4.3	Resumen del ajuste de trayectorias . . . . .	34
4.4	Remuestreo de trayectoria . . . . .	35
4.5	Twists y pose de la herramienta en la configuración inicial . . . . .	36
4.6	CD de la trayectoria capturada . . . . .	38
4.7	Eliminación de las pausas inicial y final . . . . .	40
4.8	Segmentos y mediatrices para el ajuste a un arco de circunferencia . . . . .	42
4.9	Obtención del centro del arco de circunferencia . . . . .	43
4.10	Resultado de la adaptación a un arco de circunferencia . . . . .	43
4.11	Configuraciones articulares de la trayectoria ajustada . . . . .	48
4.12	Error introducido por el cálculo de la CDI . . . . .	49
4.13	Spline cúbico . . . . .	51
4.14	Error cartesiano introducido por el spline . . . . .	52
4.15	Trayectoria reflejada . . . . .	53
4.16	Trayectoria 3D capturada, adaptada y tras el spline . . . . .	54

4.17	Trayectoria articular ejecutada con iiwa_stack a 15 °/s . . . . .	57
4.18	Problema de vibración introducida por el iiwa_stack . . . . .	58
4.19	Perfil de velocidad trapezoidal generado por el iiwa_stack . . . . .	58
4.20	Trayectoria articular ejecutada con iiwa_stack a 45 °/s . . . . .	59
4.21	Error cartesiano de ejecución con iiwa_stack a 45 °/s . . . . .	60
4.22	Trayectoria de velocidad articular con iiwa_stack a 45 °/s . . . . .	61
4.23	Error de ejecución de velocidad articular con iiwa_stack a 15 °/s . . . . .	62
4.24	Error cartesiano de ejecución con iiwa_stack a 15 °/s . . . . .	62
4.25	Problema de separar el control de la comunicación con FRI . . . . .	64
4.26	Error cartesiano de ejecución con FRI a 15 °/s . . . . .	66
4.27	Bucle de control . . . . .	67
4.28	Geometría del bucle de control implementado . . . . .	68
4.29	Influencia de $K_p$ en el error de precisión . . . . .	69
5.1	Resumen de error de precisión y repetibilidad . . . . .	74
5.2	Sincronización de trayectoria comandada y ejecutada . . . . .	75
5.3	Resumen de los retardos obtenidos según el modo de control . . . . .	75
5.4	Error de precisión. . . . .	79
5.5	Error de precisión en el modo FRI realimentado . . . . .	81
5.6	Error de repetibilidad . . . . .	85
A.1	Interacción ROS-Sunrise iiwa_stack iiwa_stack . . . . .	90
C.1	Esquema UML de iiwa_command_stack_node . . . . .	95
C.2	Esquema UML de iiwa_command_fri_node . . . . .	96
C.3	Esquema UML de IiwaMsgTransformer . . . . .	97
D.1	Esquema UML de GUI, FTsensor, DelsysSensors . . . . .	98
D.2	Esquema UML de la clase IiwaControl . . . . .	99
D.3	Esquema UML de la interfaz IiwaCommand . . . . .	100
E.1	Esquema de interacción entre MATLAB y ROS con iiwa_stack . . . . .	101
E.2	Esquema de interacción entre MATLAB y ROS con FRI . . . . .	102

F.1	Interfaz de captura de datos . . . . .	103
F.2	Interfaz de reproducción de trayectorias . . . . .	104
F.3	Interfaz de presentación de gráficas de datos . . . . .	105



## **ÍNDICE DE TABLAS**

4.1	Datos tomados manualmente. . . . .	72
5.1	Resumen de los retardos obtenidos en los experimentos realizados. . . . .	76
5.2	Resumen de la precisión obtenida en los experimentos realizados. . . . .	80
5.3	Resumen de la repetibilidad obtenida en los experimentos realizados. . . . .	84
5.4	Comparativa de características de FRI y iiwa_stack. . . . .	87



## **ACRÓNIMOS**

CD Cinemática Directa

CDD Cinemática Diferencial Directa

CDI Cinemática Diferencial Inversa

CI Cinemática Inversa

EMG Electromiográfico

FRI Fast Robot Interface

GDL Grados de libertad

IIWA Intelligent Industrial Work Assistant

MAS Modified Ashworth Scale

ROS Robot Operating System

SDC Sistema de coordenadas

ST Screw Theory

UDP User Datagram Protocol



# 1. INTRODUCCIÓN

La robótica es una rama de la ingeniería que juega un papel fundamental en campos muy diversos de la sociedad, desde el sector industrial hasta la vida doméstica, pasando por aplicaciones militares, educativas y de la salud. Su uso en el ámbito clínico es muy extenso, pudiéndose distinguir principalmente tres ramas: la robótica médica, enfocada a la cirugía y tratamientos específicos; la robótica asistencial, más orientada hacia la ayuda a pacientes y a personal clínico en tareas relacionadas con los cuidados; y la robótica de rehabilitación, que busca la recuperación de una funcionalidad perdida tras lesiones significativas [1].

El proceso de rehabilitación consta de cuatro partes principales: la evaluación o diagnóstico de la afección, el diseño y asignación de un tratamiento, la intervención sobre el paciente y finalmente, la evaluación de la efectividad del protocolo de intervención [2].

En este contexto, se presenta una aplicación de la robótica de rehabilitación enfocada al diagnóstico, en la que se utiliza la tecnología para la evaluación objetiva de un trastorno motor: la espasticidad.

## 1.1. Diagnóstico clínico tradicional

El diagnóstico de los trastornos sensomotores se realiza habitualmente mediante el uso de escalas clínicas, con el objetivo de estandarizar los resultados y reducir la posible discrepancia entre valoraciones realizadas por diferentes operadores [3]. Estas escalas se suelen basar en la división en movimientos o tareas específicas que evalúan capacidades motoras o cognitivas concretas. Uno de los procesos de diagnóstico más utilizados es el test de Fugl-Meyer [4], [5], que mide el desempeño del paciente en diferentes tareas de cognición y movimiento tanto de extremidades superiores como inferiores. Otros tests se enfocan en funciones más específicas, cuantificando el rendimiento del paciente en ejercicios concretos. El ‘Box and Blocks Test’ [6], por ejemplo, evalúa la destreza manual y coordinación mediante el conteo de los cubos que el paciente es capaz de transportar de una caja a otra.

Pese al amplio uso de las escalas de medida, el proceso de evaluación clínico habitual sigue siendo manual, siendo el terapeuta el que valora cómo de bien se realiza cada tarea. La utilización de herramientas tecnológicas fiables puede reducir la subjetividad de este proceso, permitiendo cuantificar con precisión el grado de afección del paciente. Un diagnóstico objetivo llevará a una mejor comprensión de la enfermedad, así como del modo en que esta está influenciando al enfermo, y permitirá en última instancia la elaboración de tratamientos personalizados.

La tecnología ya ha tratado en numerosas ocasiones de sistematizar dichos procesos de evaluación. Los *serious games*, videojuegos enfocados a un propósito diferente al de la mera diversión, se han aplicado para reproducir procesos de evaluación basados en escalas clínicas clásicas, como en el caso de la virtualización del ‘Box and Blocks Test’ [7] o del ‘Fugl-Meyer Test’ [8]. Otras aplicaciones han sacrificado portabilidad para ganar objetividad, mediante la utilización de todo tipo de instrumentación robótica como herramientas de medida. De este modo, Rocon *et al.* [9] aprovecha la sensorización ya presente en un exoesqueleto para realizar un diagnóstico objetivo del temblor corporal debido a trastornos neurológicos y mejorar así la posterior rehabilitación.

Sin embargo, fuera de los laboratorios, son pocos los diagnósticos clínicos que se hacen utilizando tecnología específica, y se mantienen en la mayoría de los casos los procesos manuales, como es el caso del diagnóstico de la espasticidad. A continuación, se explicará qué es la espasticidad, cómo se suele evaluar, y cómo se están aplicando las nuevas tecnologías a la mejora del diagnóstico.

## 1.2. Espasticidad: definición, causas y tratamiento

La espasticidad se caracteriza por producir una rigidez muscular puntual o generalizada, que puede producir dolor y reducir el rango de movimiento. Se puede dar en cualquier parte del cuerpo, llegando a causar dificultad para hablar o caminar y suele ser crónica [10]. Se diferencia de la rigidez muscular en que esta no es dependiente de la velocidad, mientras que la espasticidad se muestra en mayor o menor grado en función de la rapidez con que se realiza el movimiento.

Es uno de los trastornos motores más comunes tras accidentes de lesión de médula espinal, con una prevalencia de alrededor del 70 % [11]. Afecta en gran medida tanto en la infancia, debido principalmente a la parálisis cerebral, como en adultos, como consecuencia de traumatismos craneoencefálicos o como síntoma de enfermedades tan conocidas como la esclerosis múltiple o la esclerosis lateral amiotrófica (ELA). La causa específica de la espasticidad es el daño en la médula espinal, en la parte del cerebro que controla los movimientos involuntarios o en los nervios que los conectan.

Existen diversos tratamientos, desde medicamentos específicos para reducir los espasmos y relajar la tensión muscular, hasta intervenciones quirúrgicas, pasando por ayudas ortopédicas y rutinas de rehabilitación. Todos estos tratamientos han demostrado ser útiles para la reducción de la afección, si se realizan en el momento adecuado y de la manera acertada. Es primordial por lo tanto la elaboración de un tratamiento personalizado lo antes posible, y realizar un seguimiento activo de la situación del paciente.

### **1.3. Escalas clínicas de evaluación de espasticidad**

Al igual que para otros trastornos motores, se ha tratado de estandarizar su estimación utilizando escalas de medida que tratan de definir el grado de afección de cada paciente de manera objetiva.

La escala más utilizada para la evaluación de la espasticidad es la escala modificada de Ashworth (MAS), que mide la resistencia muscular durante la movilización pasiva del paciente en todo su rango de movimiento [12]. Un movimiento pasivo es aquel que aplica el terapeuta u otro agente externo sobre el paciente, sin ser un movimiento voluntario de este. La escala de Ashworth evalúa la suavidad con la que se produce el movimiento a una velocidad constante en una escala de ‘0’ a ‘4’ puntos, desde un movimiento sin componente espástico hasta un movimiento prácticamente imposible de realizar debido a la rigidez de los músculos, pasando por niveles intermedios de progresivo aumento de la tensión residual muscular o hipertonia. La escala original de Ashworth se divide en cinco rangos, mientras que la escala modificada de Ashworth añade uno entre el ‘1’ y el ‘2’ para mayor precisión en la descripción de la espasticidad moderada.

Por otro lado, la escala modificada de Tardieu [13] mide en primer lugar el rango de movimiento a una velocidad muy baja, y posteriormente evalúa la respuesta muscular a un movimiento pasivo a máxima velocidad. Esta escala es más apropiada para diferenciar entre rigidez muscular y espasticidad, debido al análisis en velocidad llevado a cabo, principal diferencia entre una y otra.

Ambos métodos de evaluación dependen del terapeuta para movilizar el brazo del paciente, y de su juicio para estimar el grado de espasticidad en la escala elegida a través de su tacto y percepción particular. Pese a que la escala sea la misma para diversos pacientes o diferentes momentos del mismo paciente, se introduce la subjetividad de la valoración del terapeuta. Existe la posibilidad de que, para un mismo paciente, un terapeuta diagnostique un grado de espasticidad determinado, y otro terapeuta o él mismo en otro momento elija otro grado de espasticidad diferente.

Además, aunque no hubiera un problema de exactitud en el diagnóstico, es inevitable el problema de la resolución. Estas escalas no tienen más de cinco o seis puntos porque la sensibilidad que ofrece el cuerpo humano a pequeños cambios en la rigidez muscular de otro humano es limitada. Introducir sistemas tecnológicos sensorizados en el proceso de diagnóstico permite obtener escalas de alta resolución capaces de detectar una pequeña mejora o deterioro en la funcionalidad motora del paciente, al tiempo que limitan la variabilidad del juicio humano.

No obstante, cabe recordar que esto no implica prescindir del personal clínico, solamente proporcionar herramientas que permitan obtener un diagnóstico mucho más completo posibilitando así el desarrollo de terapias personalizadas al estado concreto de la afección.

## 1.4. Uso de sistemas instrumentados para la evaluación objetiva de espasticidad

Los sistemas de diagnóstico de espasticidad que añaden instrumentación de diversa índole pueden clasificarse en dos grandes grupos: los que mantienen la componente humana, encargándose el fisioterapeuta de la movilización del paciente, pero añadiendo ciertos sensores para medir diversas señales; y los que incluyen también una tecnología encargada de movilizar al paciente. Estos últimos se pueden dividir a su vez en dos subgrupos, los que utilizan exoesqueletos para la movilización y los que utilizan brazos robóticos o sistemas basados en efecto final.

En el primer grupo, en el que la movilización la realiza el terapeuta, destaca el trabajo de Kim *et al.* [14], que desarrolla un sistema portátil que a partir de datos electromiográficos y cinemáticos desarrolla un sistema objetivo de evaluación basado en el agrupamiento por k-medios de los pacientes por nivel espástico. Se ha comparado su desempeño con la escala modificada de Ashworth en un grupo de 15 pacientes hemiplégicos con resultados positivos. Sin embargo, otros estudios plantean no usar las señales electromiográficas, sino analizar el movimiento de la extremidad que se detecta por medio de sensores inerciales [15]. Con la información registrada -aceleración y orientación- se pueden aplicar técnicas avanzadas de *machine learning* para clasificar el movimiento y detectar el grado de movimiento espástico.

Por su parte, Wang *et al.* [16] evalúa la espasticidad analizando contracciones voluntarias isométricas máximas y movimientos pasivos isoquinéticos. Se miden ciertos valores de las señales de par, posición y tiempo características para cada movimiento, y se correlacionan con el grado de espasticidad del paciente. Estas y otras aplicaciones sensorizadas pero sin movilización asistida por un robot [17], [18] tienen la ventaja de ser fácilmente portables, facilitando su uso clínico. Además, pese a incorporar sensores, el proceso de evaluación es muy similar al tradicional, no requiriendo una formación extensa del personal sanitario, facilitando la integración de las nuevas tecnologías en el ámbito clínico.

Por otro lado, los sistemas basados en el uso de exoesqueletos mantienen estas propiedades al tiempo que incrementan la precisión en el diagnóstico, al controlar no solo las mediciones sino también la movilización. Posteraro *et al.* [19] describe el módulo NeuroExos, un ligero exoesqueleto que rodea el brazo del paciente y actúa para movilizarlo realizando un control simultáneo en posición y par. Se miden el par máximo de extensión y el ángulo máximo de extensión y se utilizan para relacionarlos con valores de la escala modificada de Ashworth. Tiene la ventaja de ser portátil, e incluye un modo para rehabilitación.

Calabro *et al.* [20] propone una alternativa similar, Armeo-Power, un exoesqueleto ergonómico que recoge desde el hombro hasta la muñeca y asiste el movimiento activo del paciente en un extenso espacio de trabajo. Además de medir la espasticidad, mide la excitabilidad cortical, así como la excitabilidad del circuito motor espinal. Estos sistemas y otros basados en exoesqueletos [21], [22] tienen una alta aplicabilidad en entornos clí-

nicos debido a su portabilidad, pero complican la tarea del terapeuta siendo necesaria una formación para aprender a utilizar el sistema. Además, tienen el gran inconveniente de restringir el movimiento del paciente, dándole una sensación de inseguridad al mantenerle sujeto a un sistema en el que si surge cualquier problema, no puede zafarse fácilmente.

Por contraposición, los sistemas basados en efector final no presentan esta desventaja, y suelen resultar más cómodos para el paciente por guardar ciertas similitudes con la evaluación realizada por un terapeuta. Fazekas *et al.* [23] propuso por primera vez utilizar un robot industrial no modificado para realizar fisioterapia pasiva en extremidades superiores. Se centró principalmente en la rehabilitación y seguimiento del estado de los pacientes.

Sin *et al.* [24] utiliza un sistema isocinético y sensores electromiográficos para medir el ángulo de reflejo o ángulo de sacudida, relacionado a su vez con el grado de espasticidad. Estas aplicaciones basadas en efector final tienen la desventaja de ser menos portátiles y de no capturar los movimientos del paciente de manera precisa, ya que solo se cuenta con las medidas realizadas en el punto de contacto con el usuario. Se puede compensar esta desventaja añadiendo sensores iniciales y utilizando modelos biomecánicos precisos. Un análisis extenso del estado del arte actual de los sistemas de evaluación objetiva de espasticidad asistidos por robot puede encontrarse en [25].

## 1.5. Proyecto Roboespas

En este contexto, surge el proyecto Roboespas, un proyecto que busca la valoración objetiva de la espasticidad en miembros superiores utilizando un sistema basado en efector final. La utilización de un brazo robótico colaborativo permite una repetibilidad entre trayectorias de diagnóstico no presente en las movilizaciones realizadas por terapeutas, al tiempo que proporciona al paciente una comodidad no alcanzable con soluciones basadas en exoesqueletos.

Se compensa la escasez de datos recogidos del brazo del paciente a través del efector final añadiendo unos pequeños sensores iniciales, que no afectan a la comodidad del proceso por ser inalámbricos y de sencilla colocación. Son de gran utilidad para recoger datos cinemáticos del paciente, y proporcionan también datos electromiográficos que serán clave para estimar la espasticidad con mayor precisión. Con estos mismos elementos, se podría detectar una compensación postural durante la ejecución del movimiento.

El diagnóstico se basa en un modelo biomecánico personalizable, que permite ejecutar simulaciones de la movilización del brazo realizada, estimar las medidas de electromiografía que se deberían obtener para ciertos valores espásticos, y en base a su correlación con las medidas reales, estimar el grado de espasticidad del paciente. Las ventajas de este sistema frente a otros son claras: la resolución en la medida del grado de espasticidad puede llegar a ser muy superior debido al uso de un modelo biomecánico preciso, al tiempo que se mantiene la comodidad y la seguridad del paciente en todo momento.

## **1.6. Objetivos del trabajo**

Este trabajo fin de máster se centra en desarrollar el sistema que permite la movilización asistida del brazo del paciente por el robot colaborativo, siguiendo una trayectoria personalizada de flexo-extensión de codo. Se busca automatizar el método manual de evaluación de espasticidad, que debe contemplar los siguientes aspectos:

- Personalización: Las trayectorias de movilización deben poder adaptarse a cada paciente, para absorber las diferencias en sus características físicas, como su rango de movimiento o longitud de la extremidad.
- Continuidad y suavidad del movimiento: Las trayectorias no deben contener interrupciones más allá de las requeridas por el terapeuta, como pausas entre flexión y extensión o pausas entre flexo-extensiones completas. Además, la movilización debe ser suave, sin aceleraciones bruscas del robot ni situaciones que puedan poner en peligro la integridad del paciente.
- Simetría: La primera mitad de la trayectoria (flexión) debe ser idéntica pero en sentido contrario a la segunda mitad (extensión), por lo que se puede decir que la trayectoria total debe contener dos partes simétricas entre sí.
- Ajuste en velocidad: Dado que la espasticidad depende de la velocidad, para facilitar el posterior análisis de los datos obtenidos se debe establecer una velocidad constante a lo largo de todo el recorrido.
- Reproducibilidad: La trayectoria de movilización debe parecerse a la ajustada personalizada para el paciente.
- Repetibilidad: Las trayectorias ejecutadas para un mismo paciente deben ser comparables entre sí.

Además, se requiere integrar los bloques del proyecto relativos al modelo biomecánico y la estimación de la espasticidad en la aplicación principal y solventar las problemáticas derivadas de esta tarea.

## **1.7. Organización del documento**

En el presente capítulo 1 se han presentado la motivación y objetivos de este trabajo fin de máster. Para la implementación y control de las trayectorias del robot se han necesitado ciertos conocimientos previos de cinemática, control y ajuste de trayectorias que se resumen en el capítulo 2. En el capítulo 3 se explica detalladamente el funcionamiento interno del proyecto Roboespas y en el capítulo 4 se desarrolla la parte específica en la que se centra este trabajo. Para la ejecución de trayectorias en el robot se han contemplado

dos posibilidades: la utilización de un paquete ya existente llamado iiwa\_stack [26] y el desarrollo propio de una librería que permite controlar el robot utilizando la interfaz de comunicación *Fast Robot Interface* (FRI).

En el capítulo 5 se analizan las ventajas e inconvenientes de estos dos métodos y se discute sobre otros problemas encontrados y cómo se han solucionado. Finalmente, en el capítulo 6 se resumen las conclusiones de este trabajo y se habla sobre las posibilidades que abre para trabajos futuros. Se han incluido como anexos las partes más técnicas de puesta en marcha de los dos posibles modos de control del IIWA y la descripción del código desarrollado tanto en ROS como en MATLAB ®.

## 2. FUNDAMENTOS TEÓRICOS

La cinemática es una rama de la física desarrollada originalmente en la mecánica clásica, que busca describir el movimiento de puntos, cuerpos rígidos o sistemas de estos, sin considerar las fuerzas necesarias para moverlos [27]. Está íntimamente relacionada con las matemáticas, y ha sido uno de los principales objetos de estudio en la robótica desde sus orígenes.

La cinemática en la robótica se enfoca en el estudio del movimiento de sistemas multiarticulados y de robots móviles [28]. En este caso, nos centraremos en la cinemática de sistemas multiarticulados, y más concretamente de manipuladores, por ser de aplicación en este trabajo. Los manipuladores robóticos, en este contexto, se pueden definir como cadenas de sólidos rígidos o eslabones conectados entre sí por articulaciones y cuyos ejes están actuados. Las articulaciones robóticas son conexiones que imponen ciertas restricciones respecto a su movimiento, y pueden ser de traslación, revolución, helicoidales, cilíndricas, esféricas o planares [29]. En los robots industriales y colaborativos más utilizados en el mercado, la articulación más común es la de rotación, sencilla de actuar con un único motor.

Los robots manipuladores se utilizan en multitud de aplicaciones: cargar objetos pesados, realizar tareas peligrosas, repetitivas, teleoperadas o que requieran de una precisión exhaustiva. En prácticamente todos sus usos se requiere conocer la posición y orientación de la herramienta respecto a un sistema de coordenadas fijo. Este problema se conoce como el problema de la cinemática directa, que busca describir la pose del efecto final en base a las posiciones de las articulaciones de la cadena cinemática [30].

La resolución del problema cinemático directo no suele ser complicado utilizando métodos geométricos para robots con pocos grados de libertad o métodos sistemáticos como el algoritmo de Denavit-Hartenberg [31] para cadenas más largas. Sin embargo, el problema de la cinemática inversa es tema de debate aún a día de hoy. El algoritmo más extendido es el de Denavit-Hartenberg [32], aunque también hay otras soluciones basadas en algoritmos genéticos [33], en descomposiciones geométricas [34] o en redes neuronales [35]. En este trabajo, se hablará sobre la resolución de la cinemática inversa utilizando *Screw Theory*, así como de su utilización para otros problemas cinemáticos.

En esta sección se introducirán en primer lugar ciertas ideas generales relativas al control cinemático, como la definición de trayectorias articulares y cartesianas, la aplicación de polinomios interpoladores en este contexto y una introducción al uso de estrategias de control básicas. A continuación se hablará sobre los principios matemáticos, la notación y la resolución de los problemas cinemáticos clásicos utilizando *Screw Theory*.

## 2.1. Control cinemático

El control cinemático describe qué posiciones deben tener las articulaciones del robot en cada instante para cumplir los objetivos que se impongan, como el punto de destino o la trayectoria cartesiana que debe seguir el efecto final. Se deben tener en cuenta también las limitaciones articulares del manipulador para la planificación [30].

### 2.1.1. Trayectorias

Se describe una trayectoria como el recorrido que sigue alguien o algo a lo largo del tiempo para ir desde un punto a otro. En el caso de los manipuladores, se suelen distinguir dos tipos de trayectorias: articulares y cartesianas.

#### Trayectorias articulares

La trayectoria articular de un manipulador de  $n$  grados de libertad describe las  $n$  configuraciones articulares del robot a lo largo del tiempo. Por lo tanto, la trayectoria articular de un manipulador completo se puede dividir en  $n$  subtrayectorias articulares, cada una referida al camino que sigue cada una de las articulaciones. Según cómo sean estas se distinguen tres tipos de movimientos:

- Movimiento eje a eje: Mientras una articulación se mueve, las demás están paradas. Hasta que una articulación no llegue a su punto objetivo la otra no comenzará a moverse.
- Movimiento simultáneo de ejes: Todas las articulaciones comienzan a moverse a la vez, pero no necesariamente terminan al mismo tiempo.
- Movimiento isócrono: Se diseña la trayectoria de cada articulación para asegurar que todas comiencen y terminen al mismo tiempo, siendo este el de la articulación que más recorrido debe hacer o más lento se mueva.

Además, las trayectorias articulares deben ser continuas para que se puedan realizar, es decir, la posición de un instante tiene que ser próxima a la posición del instante anterior.

#### Trayectorias cartesianas

Una trayectoria cartesiana representa la posición y orientación de la herramienta o efecto final a lo largo del tiempo. Al igual que con las trayectorias articulares, se debe garantizar la continuidad de las trayectorias cartesianas para que sean alcanzables, es decir, la pose en un instante tiene que ser próxima a la pose en el instante anterior.

Mientras que asegurar la continuidad en posición se resuelve de manera intuitiva, comprobando que la resta de posiciones contiguas no es superior a determinado umbral,

asegurar la continuidad en orientación puede no ser tan sencillo debido a la notación en la que se describe la misma. Utilizando la notación por *screws*, la orientación queda representada en forma de ángulos de Euler, en la que valores distanciados  $360^\circ$  pueden ser contiguos entre sí. Para esta tarea, además de para calcular velocidades de rotación y errores en orientación, será necesario un método de resta de orientaciones válido.

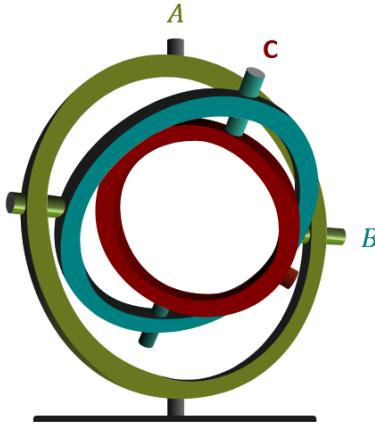


Fig. 2.1. Soportes de Cardán representativos de la notación en ángulos de Euler XYZ.

La notación en ángulos de Euler consiste en describir la orientación en el espacio de un sólido rígido con tres números, en el caso de la Figura 2.1. El primero, llamado *A* en todo el documento, describe la rotación del sólido rígido respecto al eje *X* de su propio sistema de coordenadas. El segundo número, *B*, describe la rotación del sistema de coordenadas respecto a su eje *Y* una vez ya ha sido girado entorno a su eje *X*. El tercer número, *C*, describe la rotación respecto a su eje *Z* habiendo este sido girado entorno al eje *X* primero y al eje *Y* después las cantidades *B* y *C*. La representación mediante soportes de Cardán [36] de la Figura 2.1 resulta muy útil para visualizar esta transformación. La notación en ángulos de Euler no siempre es así, y existen alternativas que representan la rotación entorno a los ejes *YZY*, por ejemplo, o cualquier combinación entre los ejes.

Los ángulos *A*, *B* y *C* pueden estar comprendidos entre  $0^\circ$  y  $360^\circ$ , entre  $-180^\circ$  y  $180^\circ$  o en el rango que se elija que tenga un total de  $360^\circ$  o  $2\pi$  radianes. Es importante que ambas orientaciones con las que se opere estén descritas en el mismo rango o dará lugar a resultados incorrectos. Además, se ha de tener en cuenta que un número fuera de ese rango no es inválido, sino simplemente se deberá ajustar correctamente. Por ejemplo,  $361^\circ$  en un sistema de  $0^\circ$  a  $360^\circ$  sería equivalente a  $1^\circ$ .

Resolver el problema de la resta de orientaciones podría simplificarse erróneamente y restar independientemente sus valores *A*, *B* y *C*, y concluir así que la velocidad necesaria para ir de  $(A_1, B_1, C_1)$  a  $(A_2, B_2, C_2)$  sería equivalente a restar y dividir entre el tiempo:  $(A_1 - A_2, B_1 - B_2, C_1 - C_2)/t$ . Pero no se debe olvidar que la rotación *B*<sub>1</sub>, por ejemplo, se hace respecto del sistema ya girado  $A_1^\circ$  en el eje *X*; al modificar *A*<sub>1</sub>, el valor de *B*<sub>1</sub> ya no es representativo. Este problema se extiende a todos los que requieran sumar o restar orientaciones o rotaciones descritas en ángulos de Euler.

Para solucionarlo, se utiliza la notación por matrices de rotación. En esta notación, se describe la orientación de cada eje de rotación como un vector de tres dimensiones en la base del sistema no rotado. Al haber tres ejes de rotación, se contará un total de 9 parámetros, dispuestos en forma de matriz de tamaño  $3 \times 3$ . Las matrices de rotación tienen, entre otras, la característica de que la multiplicación de matrices de rotación sí representa la concatenación de rotaciones, simplificando el cálculo de rotaciones inversas y de rotación de posiciones.

La solución al problema de la resta de dos orientaciones  $ori_1$  y  $ori_2$  consiste por tanto en transformarlas en matrices de rotación (Ec. 2.1), cada una de ellas representativa de la rotación necesaria para transformar el sistema de coordenadas de la base ( $B$ ) en dicha orientación,  $R_{B1}$ ,  $R_{B2}$ . La resta de posiciones se refiere a la transformación de  $ori_1$  a  $ori_2$ , o lo que es equivalente, la matriz de rotación  $R_{12}$ . Para hallarla, se utilizarán las propiedades de las matrices de rotación.

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos A & \sin A \\ 0 & -\sin A & \cos A \end{bmatrix} \begin{bmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{bmatrix} \begin{bmatrix} \cos C & -\sin C & 0 \\ \sin C & \cos C & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.1)$$

$$R_{1B} = R_{B1}^T \quad (2.2)$$

$$R_{12} = R_{1B} \cdot R_{B2} \quad (2.3)$$

La matriz de rotación resultante se transformará a su expresión en ángulos de Euler [37] y comprobaremos que los valores  $A$ ,  $B$  y  $C$  resultantes no coinciden con la simple resta de los términos. Este problema va a surgir en varias situaciones diferentes durante el desarrollo del trabajo, y es muy importante no olvidar resolverlo de la manera adecuada.

## 2.2. Screw theory: Principios matemáticos

La teoría helicoidal, más conocida por su nombre en inglés *Screw Theory*, se basa en el teorema de Chasles de 1830 [38], que dividía todo movimiento de un sólido rígido en una traslación a través de una recta y, a continuación, una rotación en torno a ese mismo eje. Fue Robert Ball el que en 1876 [39] se percató de que uniendo ambos movimientos de traslación y rotación en un solo vector se daba lugar a un espacio vectorial con propiedades matemáticas muy útiles para el cálculo de la cinemática y la dinámica de los sólidos rígidos. Para entender las peculiaridades de la teoría helicoidal, es necesario presentar antes los conceptos matemáticos de variedad y grupo, y cómo dan origen al término de grupo de Lie, y su álgebra, el álgebra de Lie.

- En primer lugar, una variedad o *manifold* es un espacio topológico [40] cuyos objetos contenidos cumplen localmente las propiedades de un espacio euclídeo. En una variedad, para cada objeto existe una función homeomórfica  $\phi$ , que es continua e

invertible, y que permite mapear el entorno o vecindad del objeto a un subespacio del espacio euclídeo.

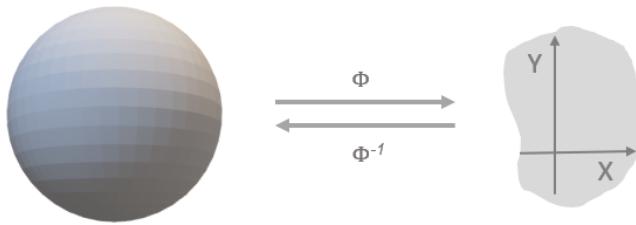


Fig. 2.2. Homeomorfismo de una variedad  $\mathbb{R}^3$  a un subconjunto del espacio euclídeo  $\mathbb{R}^2$ .

Esta definición se entiende mejor con el ejemplo de un globo (Fig. 2.2), que, pese a tener tres dimensiones, se puede recorrer todo su espacio avanzando a través de solo dos coordenadas: latitud y longitud. Localmente, se dice que un globo se comporta como un espacio euclídeo de 2 dimensiones. Sin embargo, si dibujaras un triángulo muy grande en la esfera la suma de sus ángulos no sumaría 180 grados, porque fuera de la localidad la variedad no se comporta como un espacio euclídeo.

- En segundo lugar, un grupo es una estructura algebraica formada por un conjunto y una operación interna que combina cualquier par de elementos para componer un tercero, dentro del mismo conjunto, y que satisface las propiedades asociativa, existencia de elemento neutro y elemento simétrico [41].

En base a ambos conceptos, se definen los grupos de Lie, que además de cumplir las propiedades de grupo, son variedades topológicas diferenciables, por lo que existe tanto una función continua que permite mapear la variedad al espacio euclídeo, como unas operaciones de grupo que son también continuas [42].

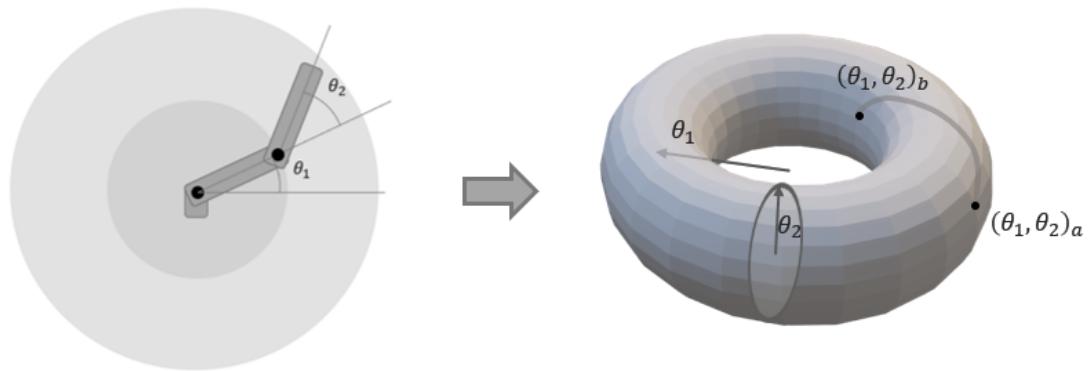


Fig. 2.3. El espacio de rotación 2R como grupo de Lie.

El espacio de rotación de un robot de dos grados de libertad se puede representar como un toroide, como se ve en la Figura 2.3, de manera que para cada ángulo  $\theta_1$  y para cada

ángulo  $\theta_2$  del espacio de trabajo del robot hay un elemento del espacio del toroide. Este espacio es asimilable a un espacio euclídeo en su localidad, de la misma manera que el globo del anterior ejemplo, por lo que es una variedad, y existe una función continua y diferenciable que mapea el espacio a un espacio euclídeo. Además, sus operaciones de grupo son continuas ya que te permiten desplazarte dentro de ese mismo espacio. Para describir el movimiento del robot desde una posición inicial  $(\theta_1, \theta_2)_a$  un incremento de  $(\theta_1, \theta_2)_{inc}$ , se puede seguir un camino continuo en la variedad hasta llegar al resultado, que es una posición nueva  $(\theta_1, \theta_2)_b$  que sigue estando dentro de la variedad. Por lo tanto, el espacio de trabajo de un robot de dos grados de libertad es un grupo de Lie.

Para cada grupo de Lie, se define su álgebra de Lie como el espacio vectorial tangente al elemento identidad del grupo [43]. Este espacio vectorial es capaz de representar de manera completa la estructura del grupo, suponiendo el resto de elementos del grupo como elementos cercanos infinitesimalmente. El corchete de Lie representa la operación conmutadora de dos elementos del espacio vectorial, similar al operador producto vectorial en espacios no euclídeos.

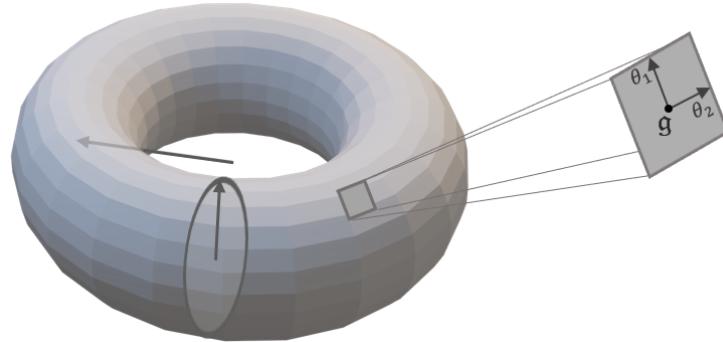


Fig. 2.4. Álgebra de Lie del grupo de Lie del espacio de configuraciones de un robot 2R.  
Plano tangente al elemento identidad  $g$  del grupo de Lie y dos vectores base  $\theta_1$  y  $\theta_2$ .

Finalmente, llegamos a la definición de la aplicación exponencial, conocida como *exponential map*, un mapa que permite reconstruir la totalidad del grupo de Lie  $G$  a partir de su álgebra  $\mathfrak{g}$  y su elemento identidad  $e$  [44]. En el caso de los brazos robóticos, el elemento identidad será una configuración articular elegida para la cual se conozca la pose de su herramienta.

El espacio de configuraciones de un robot manipulador de  $n$  grados de libertad es un grupo de Lie de  $n$  dimensiones [45]. En el ejemplo expuesto anteriormente, el espacio de configuraciones de un manipulador con dos grados de libertad era un toroide  $T^2$ . La cinemática directa (CD) de este robot mapea el espacio de configuraciones  $T^2$  a un espacio descrito por dos coordenadas de orientación y dos de posición conocido como el grupo euclidiano especial  $SE(2)$ . La cinemática inversa (CI) describe para cada pose de la herramienta su configuración articular utilizando la configuración inicial  $H(0)$ , como se ve en la Figura 2.5.

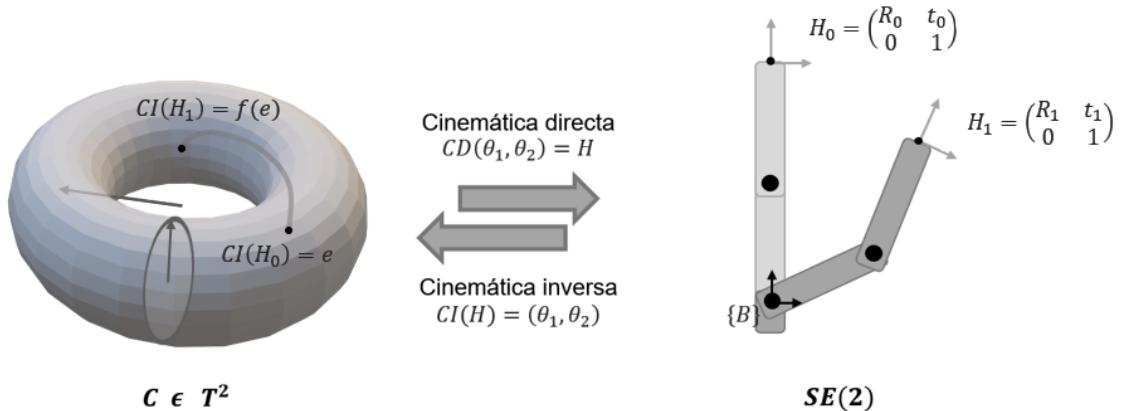


Fig. 2.5. Cinemática directa e inversa de un robot 2R con grupos de Lie.

El espacio de trabajo del IIWA se puede representar como un toroide de siete dimensiones, por tener siete articulaciones de rotación ( $C_{IIWA} \in T^7 = S^1 \times \dots \times S^7$ ), y es un grupo de Lie al igual que el toroide bidimensional que se ha expuesto como ejemplo. El espacio que contiene las posibles poses de la herramienta del IIWA está descrito por tres coordenadas de posición y tres de orientación ( $S^3 \times R^3 = SE(3)$ ). La cinemática directa permite hallar una pose de la herramienta en el sistema de coordenadas de la base  $H_{\{B\}}$  en función de sus configuraciones articulares  $\theta_1, \theta_2, \dots, \theta_7$ , y la cinemática inversa, al contrario.

$$(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7) \xrightarrow{CD} H_{\{B\}} \quad (2.4)$$

$$H_{\{B\}} \xrightarrow{CI} (\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7) \quad (2.5)$$

Antes de describir la resolución de los problemas de cinemática mediante *Screw Theory*, es necesario introducir ciertos conceptos de la aplicación práctica de los grupos de Lie y su álgebra a la descripción del movimiento de los sólidos rígidos.

### 2.2.1. Screw Theory: Notación

Un movimiento helicoidal  $\xi$  o *screw* describe la transformación de un sólido rígido en el espacio a través de una traslación  $v$  y una rotación  $\omega$ . La representación infinitesimal de un *screw* se conoce como un *twist* o eje helicoidal  $\hat{\xi}$  [46].

$$\xi = \theta \cdot \hat{\xi} = \theta \cdot \begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} X \\ Y \\ Z \\ A \\ B \\ C \end{pmatrix} \quad \begin{matrix} v, \omega \in \mathbb{R}^3 \\ X, Y, Z, A, B, C \in \mathbb{R} \\ \theta \in \mathbb{R} \end{matrix} \quad (2.6)$$

No se debe confundir la magnitud  $\theta \in \mathbb{R}$  con una configuración articular de un robot de  $n$  grados de libertad  $\theta \in \mathbb{R}^n$ . Una matriz de transformación homogénea equivalente a un

movimiento helicoidal puede derivarse mediante el uso del exponencial:

$$e^{\hat{\omega}\theta} = \begin{bmatrix} e^{\hat{\omega}\theta} & (I_3 - e^{\hat{\omega}\theta})(\omega \times v) + \omega\omega^T v\theta \\ 0 & 1 \end{bmatrix} \quad (2.7)$$

donde  $I_3$  es la matriz identidad de  $3 \times 3$ , y  $e^{\hat{\omega}\theta}$  describe una rotación pura descrita como una matriz de  $3 \times 3$ , calculable mediante la fórmula de Rodrigues:

$$e^{\hat{\omega}\theta} = I_3 + \hat{\omega}\sin(\theta) + \hat{\omega}^2 \cdot (1 - \cos(\theta)) = R_\omega(\theta) \quad (2.8)$$

donde  $\hat{\omega}$  es una matriz antisimétrica calculada a partir del vector unitario de rotación  $\omega$ .

$$\omega = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \in \mathbb{R}^3 \quad (2.9)$$

$$\hat{\omega} = \begin{bmatrix} 0 & -C & B \\ C & 0 & -A \\ -B & A & 0 \end{bmatrix} \quad (2.10)$$

A continuación, se resolverán los problemas cinemáticos clásicos utilizando *Screw Theory*: cinemática directa, cinemática inversa, cinemática diferencial directa y cinemática diferencial inversa, ampliamente debatidos en trabajos anteriores [47]-[49].

### 2.2.2. Cinemática directa utilizando Screw Theory

La cinemática directa como problema general trata de describir la pose cartesiana de la herramienta a partir de su configuración articular.

$$\theta \xrightarrow{CD} \xi \quad (2.11)$$

Haciendo uso de la aplicación exponencial descrita en la sección 2.2 se puede hallar la cinemática directa de cualquier configuración del robot a partir de, únicamente, una configuración inicial para la cual se conozca la posición y orientación de la herramienta. Se hace mediante el uso del producto de exponenciales [50], en inglés *Product of Exponentials* (POE), que resuelve la CD para cualquier manipulador de  $n$  grados de libertad utilizando notación exponencial:

$$H_{\{B\}}(\theta) = POE(\theta, n) \cdot H_{\{B\}}(0) \quad (2.12)$$

$$POE(\theta, n) = e^{\hat{\xi}_1 \theta_1} \cdot \dots \cdot e^{\hat{\xi}_n \theta_n} \quad (2.13)$$

donde  $(\hat{\xi}_1, \dots, \hat{\xi}_n)$  son los ejes helicoidales o *twists* del manipulador en la posición elegida como configuración inicial. El *twist* asignado a una articulación de traslación pura en el eje X, por ejemplo, será  $(1, 0, 0, 0, 0, 0)^T$ , y el de una articulación de rotación pura en el eje X  $(0, 0, 0, 1, 0, 0)^T$ . En todo caso deben ser vectores unitarios.  $H_{\{B\}}(0)$  se refiere a la pose de la herramienta descrita en el sistema de coordenadas de la base cuando el robot se encuentra en dicha configuración inicial.

### 2.2.3. Cinemática inversa utilizando Screw Theory

La resolución de la cinemática inversa para un robot no redundante, que tenga menos grados de libertad de los necesarios para describir la posición de su herramienta, se hace habitualmente mediante el uso de subproblemas geométricos, situaciones que se dan habitualmente en cinemática de manipuladores cuya solución se conoce. Mientras que otros subproblemas [51] se centran en las articulaciones de traslación, los subproblemas de Paden-Kahan [52] resuelven ciertas situaciones relativas a articulaciones de rotación, como por ejemplo:

- Rotación de un punto en torno a un eje (PK1): Proporciona la solución  $\theta$  para el problema  $e^{\hat{\xi}\theta} p = q$ , siendo  $p$  y  $q$  puntos del espacio  $p, q \in \mathbb{R}^3$ .
- Rotación de un punto en torno a dos ejes consecutivos (PK2): Proporciona la solución  $(\theta_1, \theta_2)$  tal que  $e^{\hat{\xi}_1 \theta_1} e^{\hat{\xi}_2 \theta_2} p = q$ , siendo  $p$  y  $q$  puntos del espacio  $p, q \in \mathbb{R}^3$ .

Al ser procedimientos geométricos, la solución que proporcionan es exacta, y la combinación de subproblemas también da por tanto una solución exacta, algo que pocas veces se consigue en resoluciones de la cinemática inversa. Sin embargo, la sistematización de

la división de la geometría de un robot manipulador en subproblemas geométricos resulta complicada, y habitualmente debe ser realizada personalmente.

Para robots redundantes el problema se complica, ya que las soluciones de la CI para una pose cartesiana son teóricamente infinitas. Habitualmente se resuelve limitando las posibles soluciones fijando una o varias de sus articulaciones en una posición conveniente, y posteriormente resolviendo la CI mediante subproblemas geométricos.

#### 2.2.4. Cinemática diferencial directa utilizando *Screw Theory*

La cinemática diferencial directa (CDD) busca describir las velocidades de la herramienta en función de las velocidades articulares. Se hace mediante el uso de la matriz jacobiana geométrica  $J_{\{B\}}$ , que tendrá tantas columnas como articulaciones tenga el robot, y cada una de ellas describirá en el sistema de coordenadas de la base la influencia de la velocidad de cada articulación en la velocidad cartesiana del extremo del robot, y será dependiente de la configuración articular instantánea.

$$\dot{\theta} \xrightarrow{CDD} \dot{\xi} \quad (2.14)$$

$$\dot{\xi} = (\dot{X} \ \dot{Y} \ \dot{Z} \ \dot{A} \ \dot{B} \ \dot{C})^T = J_{\{B\}}(\theta) \cdot \dot{\theta} \quad (2.15)$$

El cálculo de la matriz jacobiana se hace utilizando de nuevo el POE y la notación en coordenadas adjuntas. Cada columna  $i \in (1, \dots, n)$  de la matriz jacobiana geométrica de un manipulador de  $n$  grados de libertad se describe como:

$$J_{\{B\}(:,i)} = \hat{\xi}_i \quad (2.16)$$

$$J_{\{B\}}(\theta)_{(:,i)} = adj(POE(\theta, i - 1)) \cdot \hat{\xi}_i \quad (2.17)$$

$$adj(H) = adj \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R & t \times R \\ 0 & R \end{bmatrix} \quad (2.18)$$

donde  $\hat{\xi}_i$  es el *twist* de la articulación  $i$  en su configuración inicial como se describía en la sección 2.2.2. La forma adjunta de una matriz es una herramienta matemática que permite aplicar una transformación sobre un *twist* sin necesidad de describir este en forma de matriz de transformación homogénea previamente. Se puede observar que la influencia de la velocidad articular del robot en la velocidad cartesiana del extremo del robot será dependiente de las posiciones de las articulaciones anteriores, pero no de las siguientes.

El hecho de que el cálculo de la matriz jacobiana dependa de la posición articular del robot hace que la CDD solo dé resultados precisos para velocidades instantáneas, ya que en cuanto el robot se desplace una distancia infinitesimal, el valor de la matriz jacobiana habrá dejado de ser preciso. Por este motivo, aunque en una trayectoria las velocidades articulares no cambien en todo el recorrido, las velocidades cartesianas sí lo hacen.

## 2.2.5. Cinemática diferencial inversa utilizando *Screw Theory*

La cinemática diferencial inversa se enfrenta al problema contrario: tratar de describir las velocidades articulares necesarias para que la herramienta del manipulador siga una velocidad cartesiana concreta. En este caso se utiliza la inversa de la matriz jacobiana geométrica descrita anteriormente.

$$\dot{\xi} \xrightarrow{CDI} \dot{\theta} \quad (2.19)$$

$$\dot{\theta} = J_{\{B\}}(\theta)^{-1} \cdot \dot{\xi} = J_{\{B\}}(\theta)^{-1} \cdot (\dot{X} \ \dot{Y} \ \dot{Z} \ \dot{A} \ \dot{B} \ \dot{C})^T \quad (2.20)$$

Se recuerda que la matriz jacobiana es dependiente de la posición articular actual del robot, por lo que aunque la velocidad cartesiana no cambie en toda una trayectoria, habrá que actualizar las velocidades articulares necesarias para seguir esa velocidad cartesiana continuamente teniendo en cuenta la posición articular instantánea.

Calcular la inversa de la matriz puede ser trivial en robots de 6 grados de libertad, por ser el jacobiano una matriz cuadrada de tamaño  $6 \times 6$  invertible. Sin embargo, para robots con cualquier otro número de articulaciones distinto de 6, nos enfrentamos al problema de calcular la inversa de una matriz no invertible. En estos casos se utiliza la pseudoinversa de Moore-Penrose [53], una generalización de la inversa de una matriz que tiene solución para matrices no cuadradas y se calcula de la siguiente manera para cualquier matriz  $A$ :

$$A^+ = (A^T A)^{-1} A^T \quad (2.21)$$

El cálculo de la CDI consistirá por tanto en calcular en primer lugar la jacobiana geométrica como describe la Ec. 2.17, hallar su inversa o pseudoinversa según el número de articulaciones del robot, y multiplicarla por la derecha por el vector de velocidades cartesianas de la herramienta. El resultado será el vector de velocidades articulares que debe seguir el robot para que en esa posición articular concreta el extremo siga la velocidad cartesiana descrita.

### 3. ROBOESPAS: DESCRIPCIÓN DE SUBSISTEMAS E INTEGRACIÓN

El propósito del proyecto Roboespas es obtener un descriptor objetivo de la espasticidad de la extremidad superior. El método propuesto se basa en la movilización del brazo del paciente en un entorno controlado utilizando un robot colaborativo, mientras se registran variables de la respuesta del brazo al movimiento. Es necesario efectuar varias repeticiones de la misma trayectoria a diferentes velocidades porque, como se ha mencionado con anterioridad, la espasticidad se muestra a partir de un determinado umbral de velocidad.

Con el objetivo de simplificar el sistema en su etapa de diseño, se ha centrado el proyecto en la movilización y evaluación del movimiento de flexo-extensión del codo. Este movimiento consta de dos partes: la flexión que se produce cuando, partiendo de una posición con el brazo completamente estirado, se contraen los músculos hasta acercar lo máximo posible el antebrazo al biceps; y la extensión, que se hará siguiendo el mismo camino pero en sentido contrario. Pese a que los experimentos y el sistema en general están diseñados para este movimiento, será sencillo realizar los ajustes necesarios para incluir otros movimientos de extremidades superiores como la prono-supinación de antebrazo.

Se ha dividido el sistema de evaluación en tres grandes bloques: la movilización controlada del brazo del paciente, el modelado biomecánico del brazo en movimiento, y la estimación del grado de espasticidad. En la Figura 3.1 se presentan los anteriores bloques y su interacción. El bloque de movilización con captura de datos es independiente de los otros dos y corresponde a la fase inicial de la evaluación. Sin embargo, los otros dos bloques interactúan entre sí, ya que la estimación del grado de espasticidad requiere el uso del modelo biomecánico, y éste a su vez necesita una primera estimación de la ganancia espástica para poder ejecutar las simulaciones, que estará establecida por el tercer bloque. Además, el modelo biomecánico utiliza los datos capturados durante la movilización del paciente. Estas relaciones se explican en profundidad en las siguientes subsecciones.

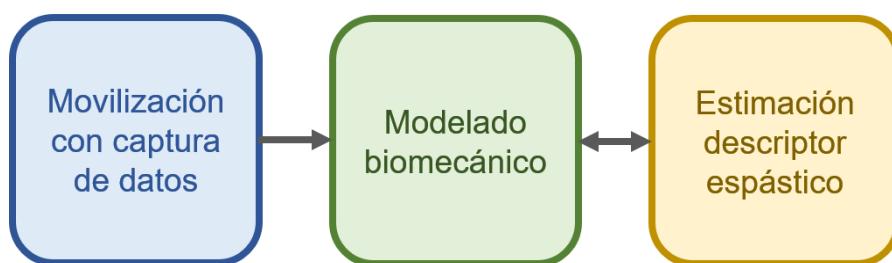


Fig. 3.1. Subsistemas en que se ha dividido el sistema Roboespas.

El robot utilizado para todo el proyecto ha sido el brazo robótico colaborativo *IIWA*, de sus siglas en inglés ‘Intelligent Industrial Work Assistant’, de KUKA Robotics [54]. Se ha escogido este brazo principalmente por ser colaborativo y tener un nivel de seguridad intrínseco elevado, condición indispensable para trabajar en entornos médicos, con pacientes vulnerables y con la necesidad de interactuar físicamente con ellos. Además, el brazo robótico IIWA ofrece ciertas ventajas respecto a otros brazos colaborativos del mercado:

- Cuenta con sensores de par precisos en cada una de sus articulaciones, que permiten estimar la fuerza externa detectada en la herramienta del robot, de utilidad para esta aplicación.
- Tiene siete articulaciones que proporcionan siete grados de libertad (GDL), algo poco común entre los brazos robóticos colaborativos, que permite al IIWA alcanzar una misma pose cartesiana con diferentes configuraciones del robot. Esto tiene grandes ventajas a la hora de reproducir trayectorias variadas, aunque en cierta manera complique los cálculos.

El modelo concreto escogido ha sido el LBR IIWA 14 R820, con una capacidad de carga de 14 kg y un alcance de 820 mm con respecto de su base, cuyo controlador se basa en el sistema operativo Sunrise OS.

Además del IIWA, entre el hardware utilizado también encontramos los sensores electromiográficos-inerciales *Trigno Wireless EMG Sensors* [55] de Delsys, y el sensor de fuerza-par *Delta SI-660-60 Sensor* [56] de ATI.

Respecto al software, el primer bloque del sistema, la captura, ajuste y reproducción de trayectorias controladas se ha concretado utilizando el framework ROS, la manera más sencilla que se ha encontrado de interactuar con el robot. Para cálculos de ajuste de trayectorias, el desarrollo de una interfaz de usuario, y para la comunicación con los sensores se ha utilizado el lenguaje de programación y entorno de desarrollo de MATLAB [57]. Además, los bloques de modelado biomecánico y estimación de la espasticidad han utilizado el programa OpenSim [58].

A continuación se desarrollarán brevemente los contenidos de cada uno de los bloques mencionados anteriormente. El bloque en el que se centra esta memoria es el de la movilización controlada del brazo del paciente. Tanto el modelado biomecánico como la estimación del descriptor espástico se han desarrollado en otros trabajos que se refieren a continuación. Además del trabajo relativo al primer bloque, también se ha desempeñado el trabajo de integración de las diferentes partes del proyecto. Por este motivo, es vital conocer el funcionamiento de cada parte y del proyecto en su conjunto.

### 3.1. Movilización pasiva asistida por robot

El primer bloque consiste en la movilización pasiva del brazo del paciente a diferentes velocidades, mientras se capturan ciertas variables del sistema. La razón de utilizar el robot colaborativo IIWA para llevar a cabo esta movilización es la búsqueda de la objetividad en diferentes aspectos:

- Se quiere controlar la velocidad a la que se ejecuta la trayectoria de la manera más segura posible, ya que la espasticidad que se muestra en cada movimiento varía en función de la velocidad de fibra de cada músculo, que a su vez es dependiente de la velocidad de movimiento del brazo.
- Además, para poder utilizar diferentes trayectorias ejecutadas a distintas velocidades, se busca una repetibilidad entre trayectorias, facilitada en gran medida por la utilización del robot.

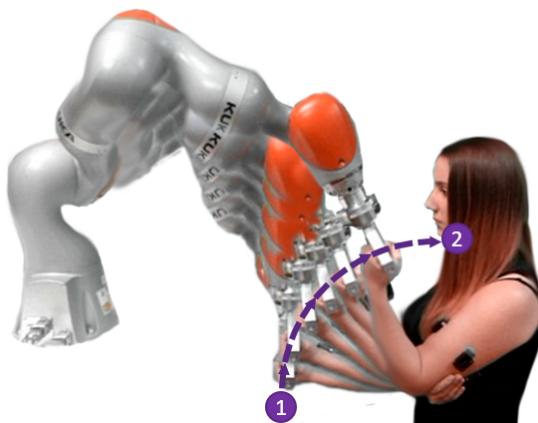


Fig. 3.2. Movilización asistida por robot.

La movilización controlada del brazo del paciente consistirá por tanto en la ejecución de una trayectoria por parte del robot, estando el paciente sujeto al efecto final de este. El movimiento del brazo del paciente será un movimiento pasivo, ya que no será él el que ponga el esfuerzo del movimiento, sino el robot.

La trayectoria elegida no será la misma para todos los pacientes, ya que queremos que el sistema sea lo más adaptable a la diversidad de usuarios posible. Cada paciente tendrá una longitud de antebrazo, una altura o una anchura de espalda diferente, incluso algún paciente requerirá realizar la movilización a una altura inferior debido al uso de silla de ruedas. Por estos motivos, antes de movilizar al paciente, necesitamos definir la trayectoria que se va a ejecutar, que deberá estar adaptada a la situación concreta del sujeto.

La trayectoria de partida para la generación de la trayectoria personalizada se capturará utilizando el robot colaborativo. Este se pondrá en modo gravedad compensada, en el cual el paciente podrá mover el robot libremente. Mientras tanto, se capturará el estado del robot en cada instante mientras realiza una trayectoria de flexión.

Una vez se haya capturado una trayectoria personalizada para cada paciente en coordenadas articulares, deberá ser transformada al sistema de coordenadas de la herramienta utilizando cinemática directa. A continuación, se hará el ajuste de la trayectoria teniendo en cuenta diferentes aspectos:

- La trayectoria ejecutada debe ser simétrica en coordenadas cartesianas, ya que la mano, conectada al efecto final, sigue el mismo camino en la flexión y en la extensión.
- Se ha observado que la mayoría de las trayectorias capturadas se asemejan a una trayectoria de arco de circunferencia, cuyo eje de giro se encuentra en la localización del codo del sujeto. Se busca por lo tanto el ajuste a un arco de circunferencia que encaje mejor con la trayectoria capturada para suavizar las vibraciones que puedan introducirse por movimientos del paciente durante la captura. Además, este ajuste simplifica el proceso de unificación de la velocidad angular de giro.
- La velocidad angular de giro del codo debe ser una concreta y controlada, elegida para cada ejecución.

Como el robot se debe comandar en coordenadas articulares, tras el ajuste, el siguiente paso consistirá en transformar la trayectoria cartesiana adaptada en una trayectoria articular. Además, esta trayectoria articular debe ajustarse a los límites del espacio de trabajo del robot, y a los límites de velocidad y aceleración articular máximos para garantizar un movimiento suave, que no genere fuerzas excesivas sobre el brazo del paciente.

Una vez se tenga la trayectoria que se quiere ejecutar para movilizar el brazo del paciente siguiendo una trayectoria de flexo-extensión perfectamente simétrica, a una velocidad continua y controlada y que se ajusta a las dimensiones del paciente, se procederá a comandar la trayectoria al robot. Debemos asegurarnos de que el robot siga las referencias con precisión para asegurar la seguridad del paciente. Para ello se han probado dos interfaces de comunicación con el robot: el iiwa\_stack y Fast Robot Interface (FRI). Esta parte se explicará en profundidad en la Sección 4.2, en la que se han analizado las diferentes estrategias y se han observado sus ventajas e inconvenientes.

Mientras se reproduce la trayectoria y el brazo del paciente está siendo movilizado, se debe realizar la captura de ciertas variables, que servirán para alimentar al modelo biomecánico. Se utilizan unos sensores electromiográficos inalámbricos, situados en ciertos músculos implicados en el movimiento de flexo-extensión. Los músculos escogidos y el protocolo de colocación de los sensores se detalla en los trabajos de Gordillo [59] y Sáenz [60].

En esta fase, se están capturando los siguientes datos:

- La posición cartesiana del efecto final del robot en cada instante de la ejecución de la trayectoria, que se relaciona de manera directa con la posición de la mano del paciente.
- Las posiciones relativas de los sensores electromiográficos (EMG) situados sobre el brazo del paciente. Esto supone información adicional de la trayectoria que sigue el brazo del paciente debido a la movilización.
- Las señales electromiográficas presentes en los músculos elegidos involucrados en la flexo-extensión. Esto será clave para el bloque 3 del sistema en el que se trata de predecir objetivamente la espasticidad del paciente.
- Las fuerzas articulares sentidas en todas las articulaciones del robot, que permitirá obtener las fuerzas cartesianas sentidas en el efecto final. Al estar la herramienta del IIWA únicamente en contacto con el brazo del paciente las fuerzas sentidas por el robot tendrán la misma magnitud que las fuerzas externas sentidas en la palma de la mano del paciente. Este dato se introducirá también al modelo biomecánico.
- Fuerza aplicada en el extremo del robot, utilizando el sensor fuerza-par adicional. Este sensor permite detectar de forma más fiable los esfuerzos que el paciente realiza durante la movilización.
- Detección de la compensación postural, por medio de un sistema independiente, que utilizando un sensor Kinect detecta la postura de las articulaciones del usuario y estima si el paciente está compensando el movimiento.

En el Capítulo 4 se desarrolla este apartado en mayor profundidad, ya que es el tema principal de esta memoria.

### **3.2. Modelado biomecánico del brazo del paciente**

Para esta parte se ha utilizado el programa OpenSim, un software de código abierto desarrollado por SimTK que permite realizar modelos biomecánicos precisos, así como simulaciones sobre estos modelos, y posterior análisis de las simulaciones realizadas. Si se quiere conocer con mayor precisión cómo se ha desarrollado este bloque, se describe mejor en los trabajos de Gordillo y Saénz. En ambos trabajos se han desarrollado modelos biomecánicos de la parte superior del torso humano, como se muestra en la Figura 3.3.

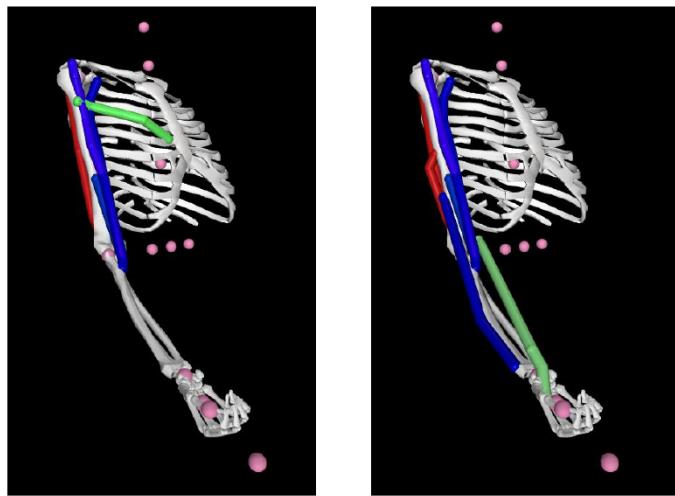


Fig. 3.3. Modelado biomecánico por Saenz[60] (izquierda) y Gordillo[59] (derecha).

El modelo incluye la estructura ósea de la parte superior de un cuerpo humano, así como los músculos implicados en la flexo-extensión de codo. Al ser código abierto, OpenSim permite modificar el funcionamiento de los modelos de los músculos utilizados, lo que modificará su comportamiento en simulación. Se han utilizado músculos basados en los modelos de Millard [61] y Thelen [62], y el trabajo de Saénz explica cómo se han modificado estos músculos para seguir el modelo de espasticidad de Hill [63]. Este modelo modificado incluye un descriptor espástico que solo se refleja en la actividad muscular cuando la velocidad de las fibras musculares supera cierto umbral predeterminado durante la realización del movimiento. Este umbral y el componente espástico introducido deberán especificarse antes de iniciar la simulación.

En la Figura 3.3 también se pueden observar unas esferas rosáceas que funcionan como marcadores. Se pueden leer sus posiciones cuando se ejecuta la simulación, o se pueden utilizar para fijar o dar una orientación de dónde deberían estar en cada marca de tiempo de la simulación. En este caso, se están utilizando como medio para añadir restricciones a la simulación indicando las posiciones que han de tener durante el movimiento.

Como se ha indicado en el bloque anterior, durante la movilización del paciente se está capturando tanto la posición de la mano como la posición de los sensores EMG. La extracción de las posiciones de los sensores a partir de los datos inerciales que estos proporcionan se describe en profundidad en el trabajo de Gordillo. Estos datos de posición se introducen al modelo biomecánico a través de los marcadores mencionados. Además, se introducen al modelo las fuerzas de contacto en la palma de la mano.

Una vez introducidas todas las variables capturadas, se ejecuta la simulación del movimiento realizado, y observaremos cómo el modelo biomecánico del brazo realiza un movimiento de flexo-extensión muy similar al realizado por el paciente. Además, se verá cómo la velocidad de la flexo-extensión producida en la simulación coincide con la velocidad a la que se produjo la movilización. Esto se debe a que todos los datos introducidos al modelo llevan sus respectivas marcas de tiempo.

Como salida de la simulación se obtienen diversos datos, como las velocidades de las fibras musculares en cada instante o las posiciones concretas de los músculos y huesos. El dato proporcionado que más nos interesa son las señales electromiográficas que deberían captarse por los sensores localizados en las marcas indicadas dada una ganancia espástica concreta.

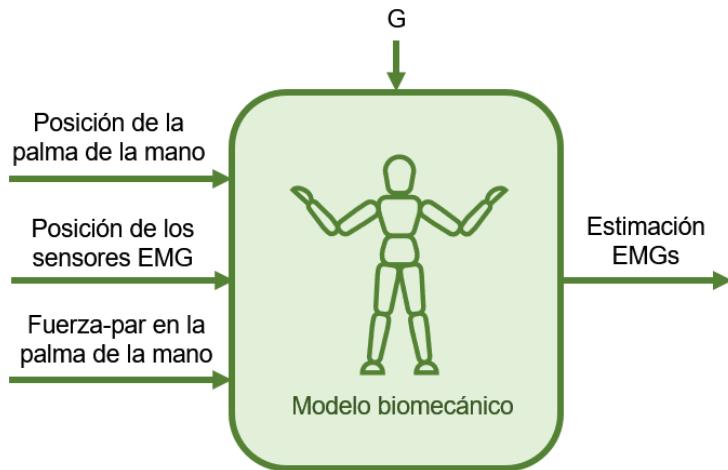


Fig. 3.4. Bloque 2: Modelo biomecánico.

Se deberán realizar varias simulaciones basadas en movilizaciones ejecutadas a varias velocidades y observar las señales electromiográficas estimadas.

### 3.3. Estimación del descriptor espástico

La estimación del descriptor de espasticidad se deberá realizar utilizando el modelo biomecánico. No es un bloque que se pueda independizar del anterior, aunque sean fases separadas del proyecto. En este caso, se seguirá utilizando OpenSim, pero en esta ocasión se hará desde la API de OpenSim para MATLAB, que permite ejecutar simulaciones y modificar parámetros de estas utilizando scripts. Esto facilitará en gran medida realizar varias simulaciones seguidas.

Para cada una de las velocidades a las que se ha realizado la movilización, y cada una de las movilizaciones producidas a esa velocidad, se han de ejecutar varias simulaciones con ganancias espásticas diferentes, y guardar las señales electromiográficas (EMG) predichas para cada una de ellas.

Con este volumen de datos, se puede comparar cada una de las señales EMG predichas con diferentes ganancias espásticas con las señales EMG capturadas durante la movilización real del paciente, y comprobar su grado de ajuste. La ganancia espástica que tenga mayor ajuste a todas las velocidades propuestas será el descriptor de espasticidad del paciente. Esta sería la opción más sencilla de entender pero la menos eficiente computacionalmente.

La alternativa propuesta por Velásquez en [64], y la que se integra en el proyecto Roboespas, introduce todo este volumen de datos generado en un sistema basado en *machine learning*. Una vez se entrena la red neuronal con la mayor cantidad de datos de que se disponga, se pueden introducir las señales EMG reales capturadas durante la movilización, y esta red devuelve el descriptor espástico más probable para ese paciente.

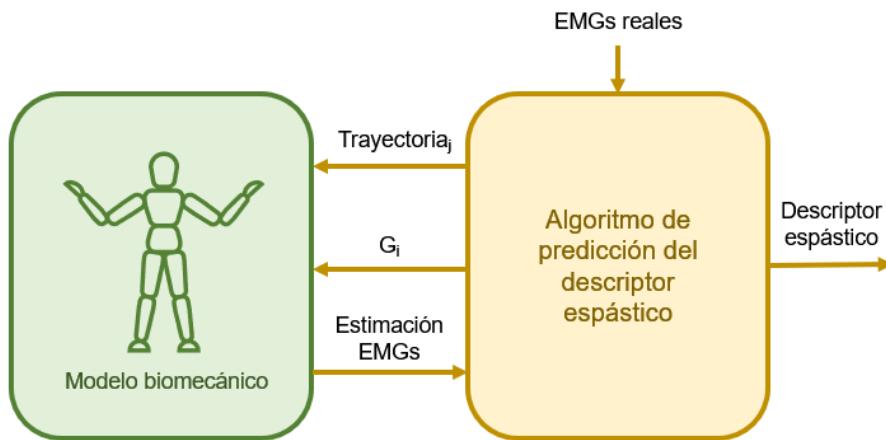


Fig. 3.5. Bloque 3: Estimación del descriptor espástico.

### 3.4. Integración de subsistemas

La variedad de tecnologías utilizadas en este proyecto hace que la integración de todas ellas en una única aplicación sea un trabajo complejo. Por un lado, tenemos el robot colaborativo *IIWA*, con el sistema operativo *Sunrise OS*, una arquitectura basada en *Windows Server*, pero que no permite la instalación de otros programas ajenos a los del robot. Por otro lado, tenemos los sensores electromiográficos-inerciales y el sensor de fuerza-par, cuyos fabricantes solo proporcionan software de control para *Windows*. Además, se necesita que la aplicación pueda interactuar con *OpenSim*, que también está disponible solo para *Windows*. Se requiere además una interfaz gráfica que permita comunicarse con toda la instrumentación mencionada, y mostrar datos de las capturas realizadas.

El software base escogido ha sido *MATLAB*, por su gran aporte en muchos aspectos de las condiciones del proyecto:

- Simplifica la programación de formulación matemática, muy presente tanto en la planificación del movimiento del robot como en el tratamiento de datos capturados por los sensores, proporcionando librerías y funciones de tratamiento de señales, generación e interpolación de trayectorias.
- Cuenta con interfaces de comunicación tanto con la tarjeta de comunicación del sensor fuerza-par *ATI SI-660-60*, como con los sensores electromiográficos-inerciales *Trigno Delsys*.

- Proporciona una herramienta para interactuar con OpenSim, permitiendo personalizar el modelo biomecánico y ejecutar simulaciones desde línea de comandos de MATLAB.
- Además de la programación por scripts, permite la programación orientada a objetos, proporcionando abstracción respecto al sistema global a los diferentes subsistemas y facilitando el desarrollo del proyecto.
- Tiene también una herramienta de diseño de interfaces gráficas llamada MATLAB App Designer, que permite la integración de otras clases y herramientas de MATLAB, como la visualización de gráficas.

La integración del control del robot en el sistema ha sido algo más compleja. La primera herramienta que se ha utilizado, *iiwa\_stack*, permite la comunicación con el sistema operativo del IIWA, el Sunrise OS, a través del framework ROS. Como MATLAB también permite la comunicación con ROS parece que no necesitamos más. Sin embargo, el paquete *iiwa\_stack* tiene una limitación de diseño, y es que sólo funciona correctamente si los mensajes al Sunrise OS se envían desde un ordenador con sistema operativo Ubuntu. Además, la generación y el envío de mensajes a ROS desde MATLAB se hace a una velocidad no lo suficientemente alta para asegurar que los comandos se envían en las marcas de tiempo establecidas.

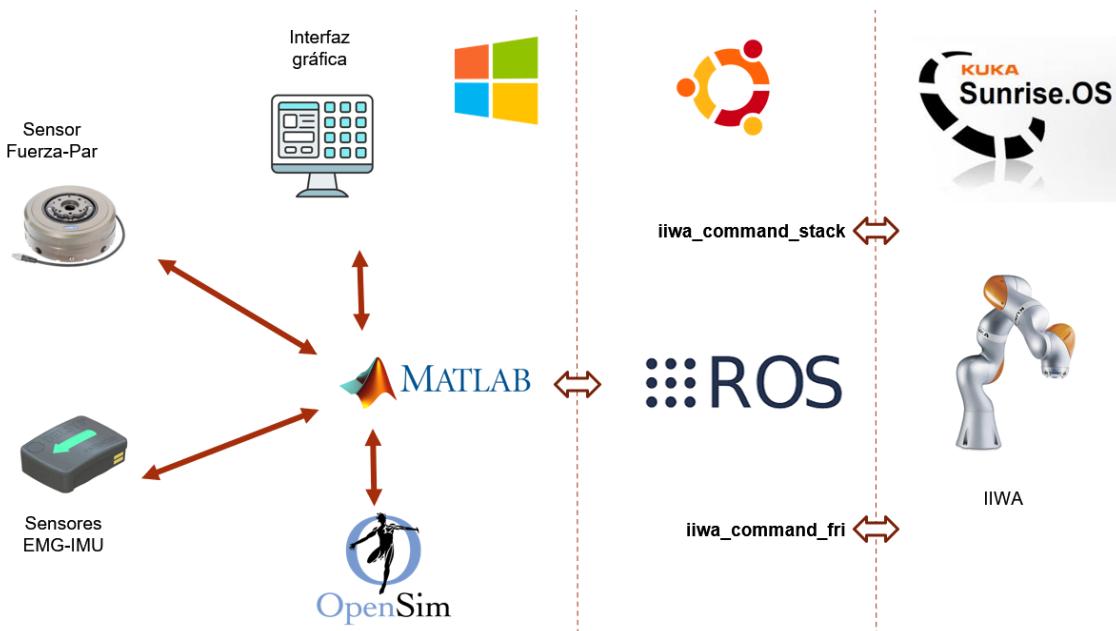


Fig. 3.6. Integración de los subsistemas de Roboespas en una única aplicación.

La solución ha sido crear un nodo intermedio denominado *iiwa\_command\_stack*, corriendo en un ordenador con Ubuntu, que recibe la trayectoria completa que se quiere ejecutar enviada a través de ROS desde otro ordenador con MATLAB-Windows, y envía los comandos individuales al Sunrise OS también a través de ROS. De este modo, se ha utilizado ROS como puente entre Windows, Ubuntu y Sunrise OS.

La segunda interfaz de comunicación con el robot utilizada, FRI, no está integrada en ROS, sino que se trata de una librería de C++ que envía mensajes a través de comunicación UDP, que son recibidos directamente por la controladora. Para simplificar la arquitectura global del sistema y permitir cambiar entre un modo de control del robot y otro de la manera más sencilla posible, se ha hecho un *wrapper* de la librería de C++ a un nodo de ROS, *iiwa\_command\_fri*, que al igual que el otro nodo descrito, recibe trayectorias completas y comanda posiciones individuales al robot, pero esta vez en vez de a través de ROS, a través de comunicación UDP. Por lo tanto, la comunicación con el robot desde MATLAB se realiza siempre a través de mensajes de ROS.

De esta manera, todas las tecnologías necesarias se han integrado en una única aplicación, cuya arquitectura se resume en la Figura 3.6. En el Anexo E se muestran los grafos de nodos de ROS cuando se utiliza *iiwa\_command\_stack*, y cuando se utiliza *iiwa\_command\_fri*.

A continuación, el trabajo se centra en la descripción de la implementación del primer bloque del sistema. Primero se desarrolla cómo se lleva a cabo la generación de la trayectoria personalizada, a continuación, se definen los métodos utilizados para la ejecución de la trayectoria en el robot, y finalmente se especifican los datos que se han capturado durante la movilización. Además, se ha desarrollado la interfaz de usuario, incluida en el anexo F.

## 4. SISTEMA DE MOVILIZACIÓN ASISTIDA POR ROBOT

El desarrollo principal del trabajo ha consistido en el diseño e implementación del sistema de movilización asistida por robot cuyas características principales se resumían en 3.1. El sistema se puede dividir en tres grandes bloques:

- Generación de trayectoria personalizada: Se grabará una primera trayectoria a partir de un movimiento de flexión, y se ajustará a las condiciones deseadas y a la velocidad establecida por el terapeuta.
- Reproducción de trayectoria: Movilización asistida por robot del brazo del paciente en un movimiento de flexo-extensión. Será clave el control cinemático.
- Captura de datos durante la movilización: Se capturan datos de ciertos sensores para su posterior análisis.

En la Figura 4.1 se muestra el esquema de interacción de este bloque con los diferentes dispositivos y con otros bloques presentes en Roboespas.

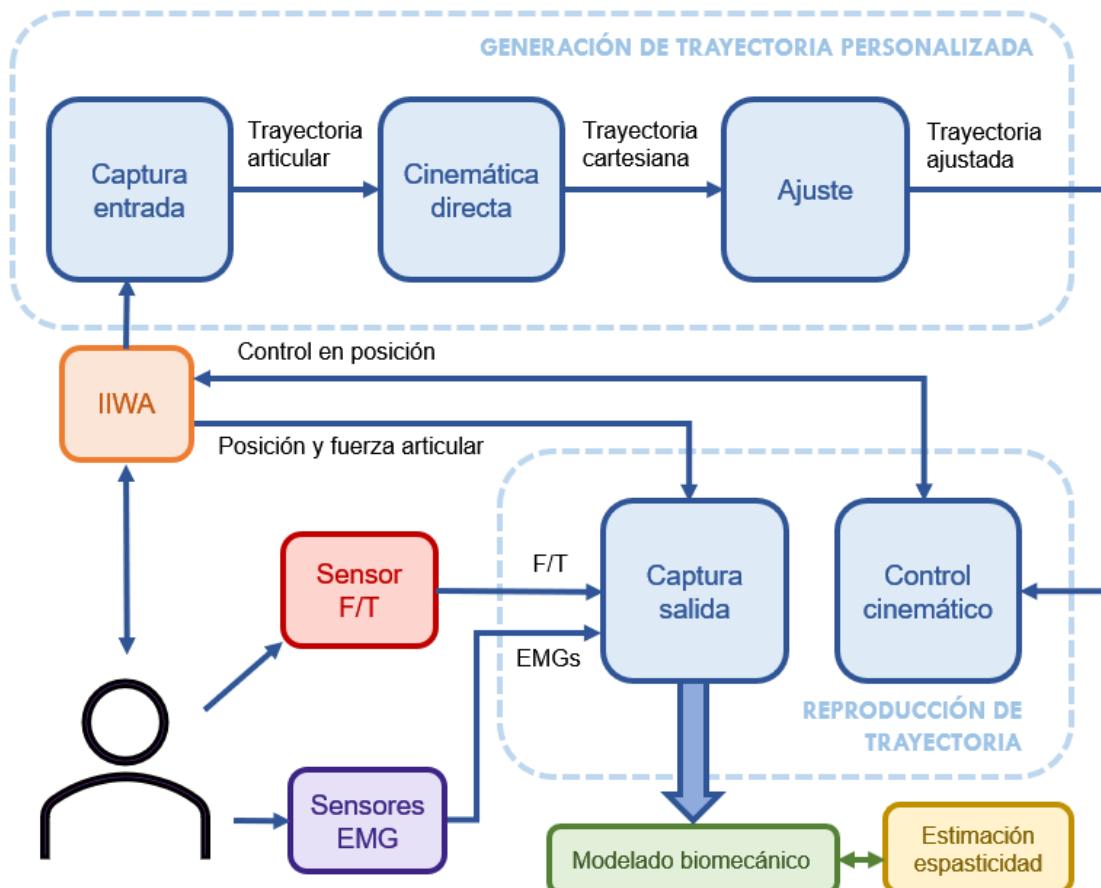


Fig. 4.1. Bloque 1: Movilización asistida por robot.

#### 4.1. Generación de la trayectoria personalizada

En esta sección se detalla con mayor precisión cómo se realiza la generación de una trayectoria personalizada para cada paciente, cumpliendo ciertos requisitos. Se quiere encontrar una trayectoria cartesiana que se ajuste a las condiciones requeridas, y posteriormente, transformar esta trayectoria cartesiana en una articular que pueda reproducir el robot.

La mano del paciente debe seguir una trayectoria que se pueda realizar sin forzar los límites articulares del brazo. Esto introduce unas restricciones, que se aplicarán de igual medida a la trayectoria que sigue el efecto final, ya que ambos estarán en contacto durante el momento de la movilización. Por lo tanto, la trayectoria cartesiana ajustada ejecutada por el robot debe cumplir:

- Las posiciones han de ajustarse a un arco de circunferencia, ya que es la trayectoria que sigue el centro de la palma de la mano cuando se realiza un movimiento de flexo-extensión.
- La trayectoria debe estar contenida en un plano.
- La velocidad seguida por el efecto final debe asegurar una velocidad angular escogida para la movilización que se vaya a llevar a cabo.
- La trayectoria debe ser completamente simétrica, siendo la flexo-extensión un movimiento que consta de dos partes cuyas trayectorias son iguales en sentido contrario.
- Las trayectorias de flexión y extensión por separado no deben incluir pausas antes o después de la trayectoria ni durante la trayectoria, ya que su velocidad angular respecto al punto de giro - el codo - debe ser continua.
- Se debe incluir una pausa entre la flexión y la extensión de duración variable.
- Se requiere por último conocer la trayectoria articular correspondiente a la trayectoria ajustada que cumpla los puntos anteriores.

Se consigue una trayectoria de estas características en dos fases:

1. Captura: Permite que la trayectoria generada esté adaptada a las proporciones del usuario y a su rango de movimiento.
2. Ajuste: Asegura que la trayectoria de movilización cumple las propiedades requeridas por los objetivos del trabajo.

#### 4.1.1. Captura de la trayectoria inicial

Con la intención de que la trayectoria generada se adecúe a las dimensiones del paciente y a su rango de movimiento, se le permite en primer lugar que realice una flexo-extensión libre, como él la haría naturalmente, mientras se graba el movimiento. Para capturar el movimiento del brazo del paciente utilizamos el robot colaborativo puesto en modo gravedad compensada, y pedimos al paciente que sostenga la herramienta del robot mientras realiza un movimiento de flexión. Será suficiente con capturar una única trayectoria de entrada para cada paciente, asumiendo que su cuerpo no cambia entre una movilización y otra.

Un robot se dice que está en modo gravedad compensada, cuando si algo externo ejerce una fuerza sobre él, éste se mueve en la dirección de la fuerza ejercida, y cuando se deja de ejercer fuerza alguna, el robot mantiene su posición [65]. De esta manera, se genera la sensación de que los frenos del robot están libres y el usuario puede mover el robot a dónde quiera. Para lograr este comportamiento, se ha utilizado el paquete `iiwa_stack` de Salvo Virga [26]. Este paquete tiene diversas funcionalidades, como leer el estado del robot, comandarle posiciones y velocidades, y cambiar ciertas configuraciones del robot. En el Anexo A se explica cómo se ha puesto en funcionamiento este paquete y qué posibilidades ofrece. Además, en el capítulo 5 se analizan sus ventajas e inconvenientes a la hora de utilizarlo para reproducir trayectorias. En esta sección se va a utilizar su funcionalidad para poner el robot en modo gravedad compensada sin grandes complicaciones. Hay dos opciones disponibles para elegir cómo se comporta el modo gravedad compensada del paquete `iiwa_stack`:

- (a) Modo gravedad compensada con restricciones articulares: Este modo permite liberar el movimiento de una o varias articulaciones en mayor o menor medida. Para cada articulación, se deben especificar los valores de *stiffness* y *damping*. El primero se refiere a la rigidez del robot, la cantidad de fuerza necesaria para mover el robot en cierta dirección: si se elige 0 el robot se moverá con la más leve presión que se ejerza, y si se elige 1 hará falta una fuerza enorme para mover el robot tan solo unos milímetros. Digamos que este parámetro simula lo que “pesa” mover cada articulación. El segundo parámetro controla la amortiguación del movimiento. Podríamos entenderlo imaginando un pequeño muelle atado por un extremo a la posición inicial del robot, y por otro a la posición actual del robot después de haberlo movido. Si este muelle es muy fino, al empujar el robot este continuará su movimiento lejos de la posición inicial sin restricción. Si el muelle es más grueso, tirará hacia la posición inicial, impidiendo al robot alejarse demasiado.
- (b) Modo gravedad compensada con restricciones cartesianas: En este caso, se controlan los mismos parámetros, pero no referidos a cada articulación del robot, sino a cada coordenada cartesiana. De esta manera, se puede restringir el movimiento en una coordenada cartesiana concreta subiendo el parámetro *stiffness* de esa coordenada cartesiana a un valor muy alto, siendo el máximo 5000 [N/m] para las coor-

denadas de posición y 300 [Nm/rad] para las coordenadas de orientación. Cuando se restringe el movimiento en modo gravedad compensada en un eje, no se refiere a un eje en el sistema de coordenadas de la base del robot, sino a un eje en el sistema de coordenadas de la herramienta en la pose en la que se activa el modo gravedad compensada. Por este motivo, será clave la posición inicial del robot al liberarlo. Se puede ver este sistema de coordenadas en la Figura 4.2.

Para simplificar el posterior ajuste de la trayectoria, se busca que esta esté contenida dentro de un plano paralelo al plano YZ del sistema de coordenadas de la base ( $SDC_B$ ). Para conseguir esto, se fija la pose inicial de la herramienta de tal manera que el eje X del sistema de coordenadas de la herramienta ( $SDC_H$ ) coincida con el eje Y del  $SDC_B$ , el eje X del  $SDC_H$  con el eje Y del  $SDC_B$  y el eje Z del  $SDC_H$  con el inverso del eje Z del  $SDC_B$ , tal y como se puede ver en la Figura 4.2.

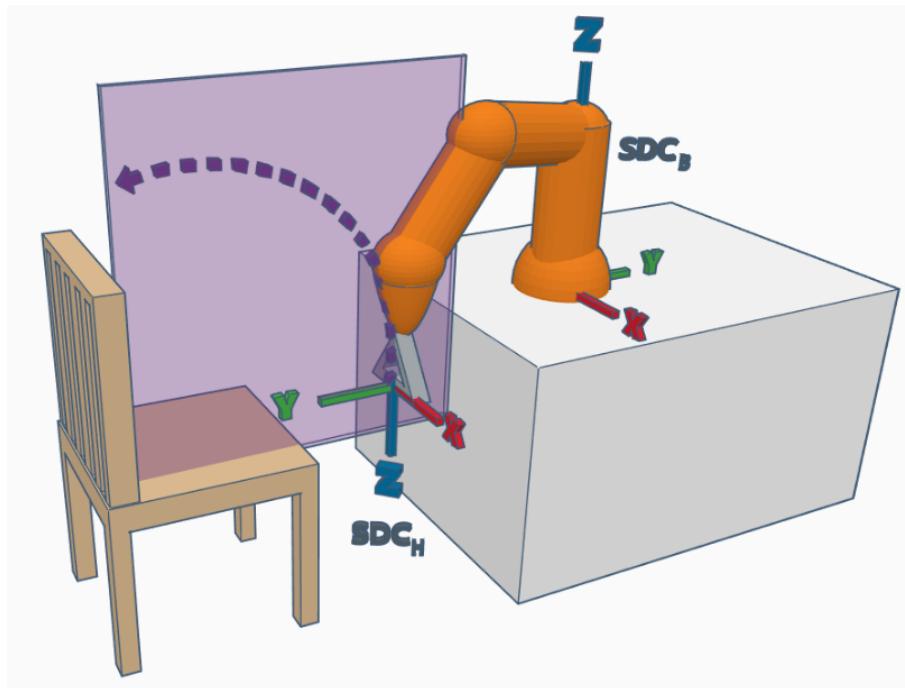


Fig. 4.2. Sistema de coordenadas de la herramienta ( $SDC_H$ ), sistema de coordenadas de la base ( $SDC_B$ ), ejemplo de una trayectoria capturada en morado, con la dirección de captura y el plano al que pertenece.

Por lo tanto, para conseguir que el movimiento se produzca en el plano YZ del  $SDC_B$  habrá que restringir el desplazamiento en el eje X, y la rotación entorno al eje Y y el eje Z (también llamadas coordenadas B y C) del  $SDC_H$ , subiendo el parámetro *stiffness* en estas coordenadas. Se permitirá el desplazamiento de la posición inicial, pero no la rotación, para conservar la correlación entre el  $SDC_H$  y el  $SDC_B$ .

La captura de la trayectoria base constará por lo tanto de dos momentos en los que se utiliza la funcionalidad de poner el robot en modo gravedad compensada:

1. Adecuación de la posición inicial a las necesidades del paciente: Se pone el robot en modo gravedad compensada liberando todas las coordenadas salvo las de rotación (A, B y C).
2. Captura de la trayectoria: Se pone el robot en modo gravedad compensada liberando solo las coordenadas YZ y la rotación en torno al eje X (coordenada A).

Tras poner el robot en modo gravedad compensada, se comienza a capturar la trayectoria articular que realiza, y se le indica al paciente que realice una trayectoria de flexión de codo sujetando la maneta del robot. El punto 1 dependerá de la posición inicial escogida por el paciente, y el punto 2 dependerá del rango de movimiento que tenga el paciente. Solo se capture la flexión porque la extensión será la misma trayectoria en sentido contrario. La velocidad de captura es irrelevante, ya que luego se adapta fijando una velocidad continua. Se debe cuidar que el paciente no flexione la muñeca ni desplace el codo durante la captura. Al finalizar la flexión, se para la captura de datos manualmente. Los datos se capturan desde el nodo de ROS *iiwa\_command\_stack*, cuyo funcionamiento se detalla en el Anexo C. Para iniciar la captura se utiliza el servicio *CaptureStart*, que no devuelve ningún dato; y para finalizar la captura se utiliza el servicio *CaptureStop*, que devuelve una variable de tipo *trajectory\_msgs/JointTrajectory* [66]. Ambos servicios se llaman desde la clase de MATLAB *IiwaControlRoboespas*, que a su vez utiliza la clase *IiwaCommandStack*, descritas ambas en el Anexo D.

A grandes rasgos, *IiwaCommandStack* sirve para controlar el robot (moverlo, capturar datos, ponerlo en modo gravedad compensada, ...) utilizando el paquete *iiwa\_stack*; y *IiwaControlRoboespas* utiliza la anterior, y tiene además funciones relativas a la utilización del robot IIWA dentro del proyecto Roboespas, como ajuste de trayectorias, tratamiento de datos y su representación mediante gráficas.

Una vez se obtiene la trayectoria de flexión capturada en coordenadas articulares como una variable de tipo *trajectory\_msgs/JointTrajectory*, se utiliza un servicio del nodo *msg\_transform\_helper*, que ayuda a transformar tipos de datos de ROS en matrices de MATLAB a alta velocidad. Su funcionamiento se detalla en el Anexo C. Sin la utilización de este nodo también se podría transformar el mensaje de ROS en una matriz desde el propio MATLAB, pero sería poco eficiente computacionalmente.

La transformación y ajuste de esta trayectoria se realiza en su totalidad con funciones de MATLAB sobre estructuras de MATLAB. Para mantener estas funciones ordenadas según su aplicación, se han ordenado por clases. En este caso, la trayectoria capturada se ha guardado en una clase de implementación propia llamada *IiwaTrajectory*, descrita en el Anexo D. En esta clase se implementan todas las funciones necesarias para el ajuste de la trayectoria, que se detallan a continuación.

#### 4.1.2. Ajuste de la trayectoria

El ajuste de la trayectoria a las condiciones enumeradas al comienzo de la sección 4.1 se hará en varias fases, resumidas en la Figura 4.3: se remuestreará la trayectoria articular porque sus marcas de tiempo no se capturan equidistantes, se hallará la correspondiente trayectoria cartesiana, se eliminarán las pausas iniciales y finales generadas por la activación y desactivación manual de la captura, se ajustará la trayectoria a un arco de circunferencia en torno al codo del paciente, se hallará la correspondiente trayectoria articular para la ajustada, se adecuará la trayectoria para que sea realizable por el robot y finalmente se reflejará la misma para generar una trayectoria de flexo-extensión. Se detalla cada una de ellas a continuación.

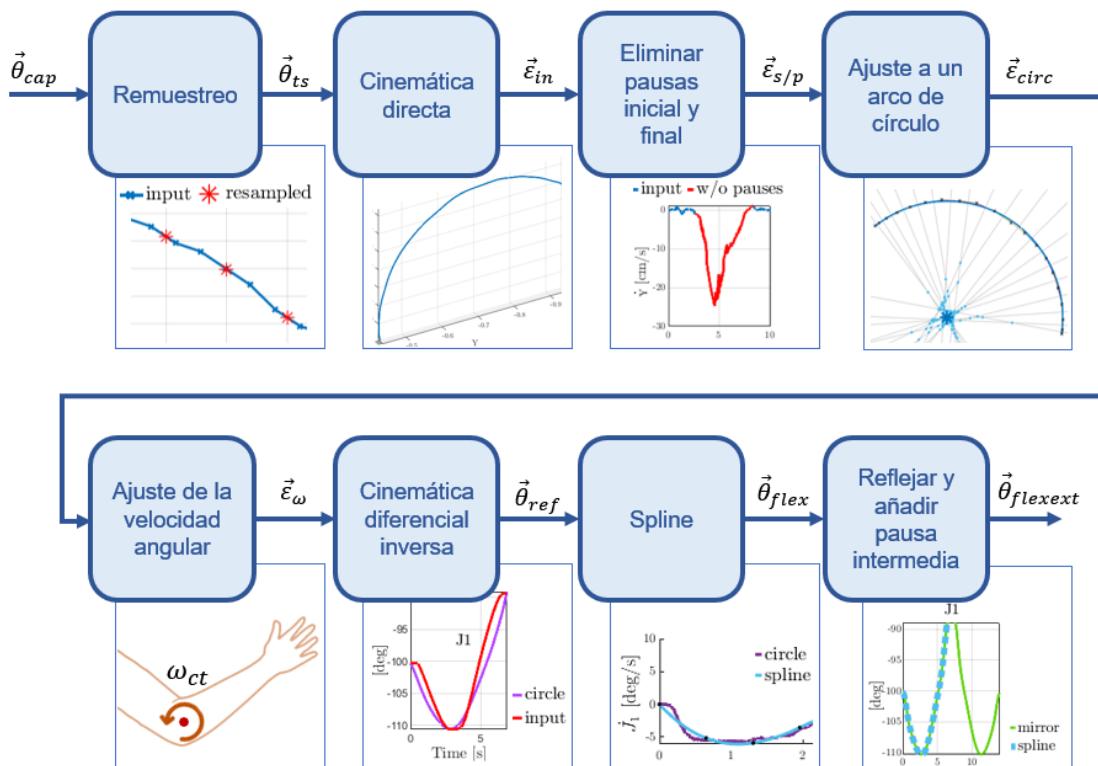


Fig. 4.3. División del ajuste de la trayectoria capturada en sus diferentes etapas.

#### Remuestreo

La captura realizada incluye todos los mensajes que se han recibido desde el paquete *iiwa\_stack*, que han sido generados por el programa de Java ejecutado en la controladora del IIWA. Se detalla el funcionamiento en el Anexo A. Los mensajes recibidos vienen cada uno con su propia marca de tiempo, que se almacena junto a la posición durante la captura de la trayectoria. Las marcas de tiempo recibidas no están necesariamente equidistantes, y esto puede complicar ciertos ajustes posteriores.

Por ello, lo primero que se hará con la trayectoria capturada será remuestreada a un

tiempo de muestreo fijo. El tiempo escogido será de 5 milisegundos, por ser el mínimo intervalo de control utilizado en la posterior reproducción de trayectorias, sección 4.2. Utilizar un tiempo de muestreo inferior en los ajustes de la trayectoria y remuestrearla al final sólo conllevaría tiempos de cómputo mayores al generarse una cantidad de puntos superior que se descartarían posteriormente. Además, se han observado diferentes capturas y en ningún caso se reciben mensajes distanciados más de 3 milisegundos, por lo que haciendo un muestreo a 5 milisegundos la desviación del error introducido por la interpolación será menor que si el periodo de lectura fuera inferior.

La implementación de este paso ha sido muy sencilla gracias a la clase `timeseries` de MATLAB. En primer lugar, se ha transformado cada una de las siete trayectorias articulares capturadas en una variable de tipo `timeseries`, que incluye también el vector de tiempos adquirido. A continuación, se ha utilizado la función `resample` que proporciona esta clase, introduciendo como entrada el nuevo vector de tiempos equidistantes. La función devuelve otra variable de tipo `timeseries` con la trayectoria articular remuestreada por interpolación lineal. En la Figura 4.4 se ve un fragmento de la trayectoria seguida por la primera articulación antes y después de remuestrearla.

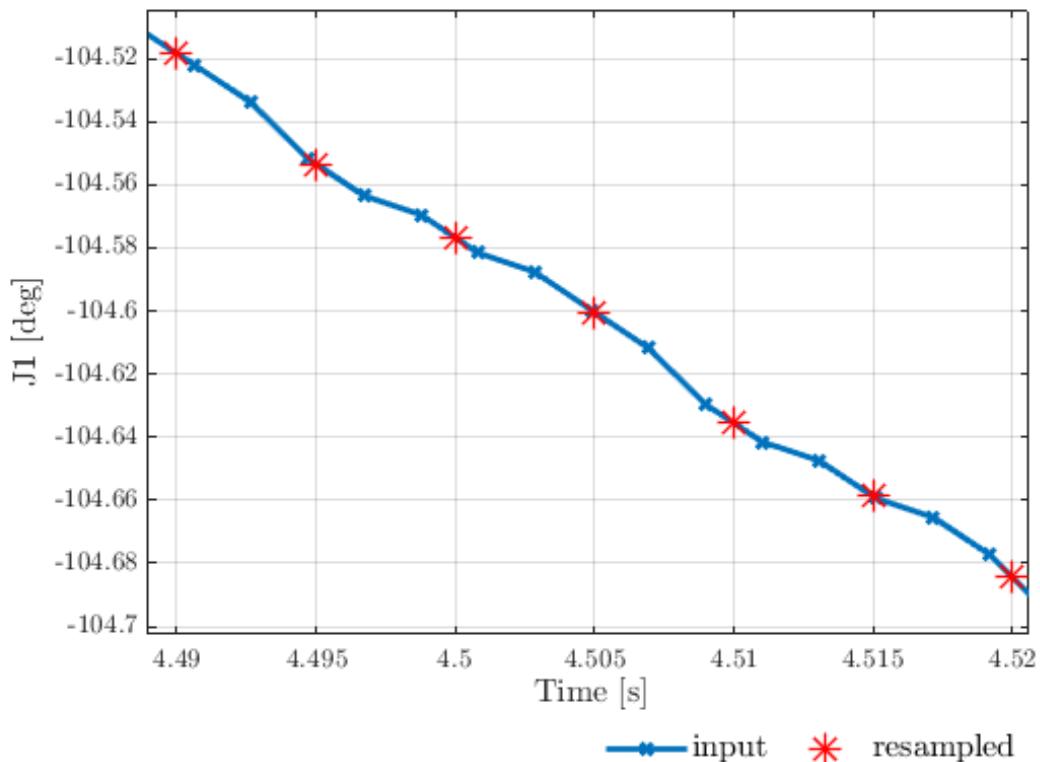


Fig. 4.4. Sección ampliada de la trayectoria articular de J1 y la señal remuestreada correspondiente.

## Cinemática directa

A continuación, se deben obtener las poses cartesianas correspondientes a cada una de las configuraciones articulares capturadas. Para cada punto de la trayectoria, se calcula la cinemática directa (CD) por *Screw Theory*, que toma como entrada una configuración articular  $\theta$ , y devuelve una pose cartesiana  $\xi$ , valiéndose de las propiedades matemáticas del álgebra de Lie del espacio de configuraciones del robot. En la sección 2.2.2, se describe la cinemática directa generalizada para cualquier robot.

Para el caso específico del IIWA, como tiene 7 articulaciones,  $\theta$  será un vector de tamaño  $1 \times 7$ , y la ecuación de la cinemática directa será por tanto:

$$H(\theta) = e^{\hat{\xi}\theta_1} \cdot e^{\hat{\xi}\theta_2} \cdot e^{\hat{\xi}\theta_3} \cdot e^{\hat{\xi}\theta_4} \cdot e^{\hat{\xi}\theta_5} \cdot e^{\hat{\xi}\theta_6} \cdot e^{\hat{\xi}\theta_7} \cdot H(0) \quad (4.1)$$

donde  $H(0)$  es la posición de la herramienta en el  $SDC_B$  en la configuración inicial escogida, que en este caso se toma con el brazo completamente estirado; y  $e^{\hat{\xi}\theta_j}$  es el exponente del *twist* de la articulación en la posición inicial multiplicado por la magnitud  $\theta_j$ ,  $j \in (1, 7)$ .

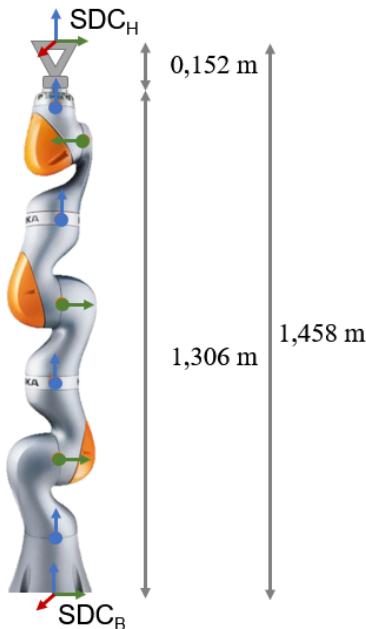


Fig. 4.5. Twists y pose de la herramienta en la configuración inicial.

Como se ve en la Figura 4.5, el  $SDC_H$  en esta posición coincide en orientación con el  $SDC_B$ , pero a cierta distancia en el eje Z. Por lo tanto, la matriz de transformación homogénea de la herramienta en el  $SDC_B$  será:

$$H(0) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1,458 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.2)$$

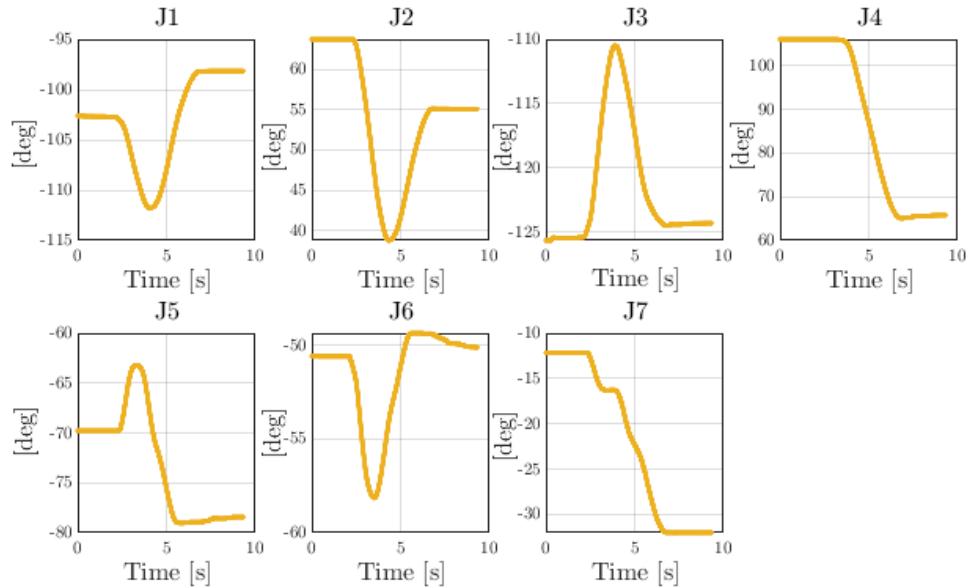
El exponencial  $e^{\hat{\xi}\theta_j}$  se calcula utilizando el mapa exponencial descrito en la Ec. 2.13, teniendo en cuenta las posiciones de los *twists* en la posición inicial que se muestran en la Fig. 4.5.

El resultado de la Ec. 4.1,  $H(\theta)$ , se transformará fácilmente en el screw  $\xi$  teniendo en cuenta que  $H(0)$  es una matriz de transformación homogénea compuesta por una matriz de rotación  $R$  y un vector de translación  $p$ .

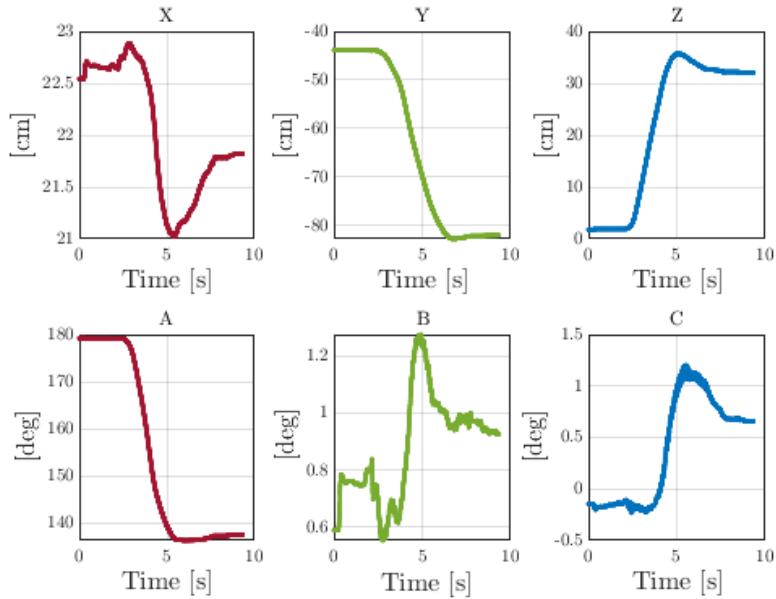
$$H(\theta) = \begin{pmatrix} R & p \\ 0 & 1 \end{pmatrix} \rightarrow \xi = \begin{pmatrix} p \\ \text{eul}(R) \end{pmatrix} \quad (4.3)$$

Este procedimiento se sigue para todas las configuraciones del vector de posiciones articulares capturadas  $\vec{\theta}$ , dando lugar a un vector de screws  $\vec{\xi}$ , que representan las poses cartesianas del robot durante la fase de captura de la trayectoria de entrada. El vector de tiempos de la trayectoria no se verá modificado en este paso.

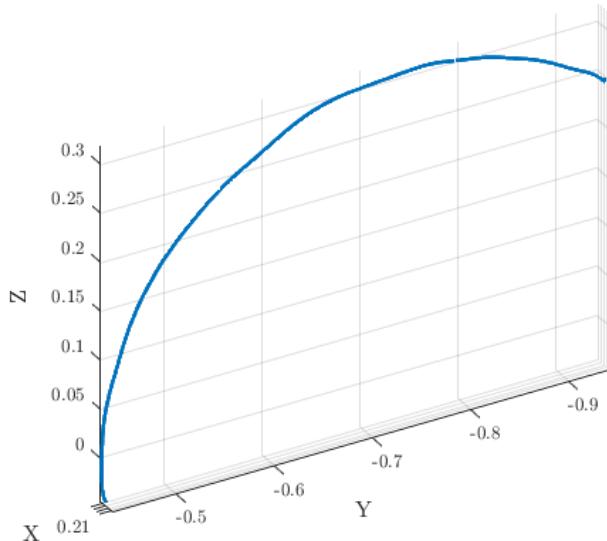
Todas las funciones relativas a *Screw Theory* se han incluido en una clase de MATLAB de métodos estáticos llamada `IiwaScrewTheory`, descrita en el Anexo D. También se han implementado la mayoría de las funciones en un namespace de C++ con el mismo nombre, descrito en el Anexo C, para las ocasiones en que se requiere su uso desde un nodo de ROS, como es el caso del control cinemático en velocidad cartesiana explicado en la sección 4.2.



(a) Posición articular capturada directamente del movimiento del paciente.



(b) Posición cartesiana correspondiente a la captura del movimiento del paciente.



(c) Posición cartesiana tridimensional correspondiente a la captura del movimiento del paciente.

Fig. 4.6. Entrada (a) y salida (b, c) de la CD aplicada sobre la trayectoria de entrada.

### Eliminación de las pausas inicial y final

La indicación de cuándo se inicia y cuándo finaliza la captura de la trayectoria se hace de manera manual, debiendo el terapeuta hacerle clic a un botón para cada tarea. Esto hace que desde que se comienza a capturar la trayectoria hasta que realmente el paciente empieza a moverse, pasa un cierto tiempo, y desde que se termina de ejecutar la trayectoria hasta que realmente finaliza la captura, hay otra pequeña pausa. Si no se eliminan estos intervalos de tiempo de la trayectoria capturada, los posteriores ajustes

podrían no resultar satisfactorios, y aunque así fuera, introduciría retrasos innecesarios en la movilización posterior del paciente. Por estos motivos, se deben eliminar en la medida de lo posible.

Para ello, se ha utilizado la cinemática diferencial, que a partir de las velocidades articulares devuelve las velocidades cartesianas en cada punto de la trayectoria. Se podrían calcular también por diferenciación de las posiciones cartesianas obtenidas anteriormente, pero al ser el dato de las configuraciones articulares el capturado, se ha considerado más apropiado que sea el utilizado para el cálculo. La cinemática diferencial directa (CDD) parte de las velocidades articulares, que se obtienen en este caso por diferenciación de las posiciones articulares, y utiliza después la matriz jacobiana geométrica para obtener las velocidades cartesianas.

Para obtener las velocidades articulares, en primer lugar se calcula una aproximación de las velocidades medias articulares en cada intervalo de tiempo por diferencias finitas, restando las posiciones articulares consecutivas y dividiéndolas entre la resta de marcas de tiempo sucesivas.

$$\dot{\theta}_{i_{med}} = \frac{\Delta\theta_i}{\Delta t_i} = \frac{\theta_{i+1} - \theta_i}{t_{i+1} - t_i} \quad i \in (1, n-1) \quad (4.4)$$

Las velocidades medias se pueden interpretar como las velocidades instantáneas en el punto medio entre dos marcas de tiempo. Para conseguir las velocidades articulares en cada una de las marcas de tiempo de la trayectoria original capturada, se calcula la media de las velocidades articulares medias. Además, se fijan las velocidades articulares inicial y final a cero.

$$\begin{aligned} \dot{\theta}_i &= \frac{\dot{\theta}_{i_{med}} + \dot{\theta}_{i+1_{med}}}{2} \quad i \in (2, n-1) \\ \dot{\theta}_1 &= \dot{\theta}_n = \vec{0} \end{aligned} \quad (4.5)$$

Una vez se obtienen las velocidades articulares en cada una de las marcas de tiempo, se computa el vector de velocidades cartesianas, utilizando el jacobiano geométrico actualizado en cada posición articular, cuyo cálculo se detalla en la Ec. 2.17.

$$\dot{\xi}_i = J_B(\theta) \cdot \dot{\theta}_i \quad (4.6)$$

Obtenidas las velocidades cartesianas, se procede al cálculo de la norma euclídea de la parte de cada velocidad cartesiana referida a la traslación, es decir, los 3 primeros elementos. Se ha utilizado la función de MATLAB `norm`.

$$\dot{\xi}_{euc_i} = \text{norm}(\dot{\xi}_{i(1..3)}) \quad (4.7)$$

Con este parámetro, ya se puede filtrar qué puntos tienen una velocidad cartesiana lo suficientemente alta para considerar que el paciente ya estaba moviendo el robot durante

la captura. Se ha considerado una velocidad mínima de 15 cm/s. Pese a que no es recomendable, se contempla la posibilidad de que el paciente pare en mitad de la trayectoria y continúe posteriormente, ya que se guardan los índices del primer instante desde el inicio y el último desde el final que superan ese umbral de velocidad cartesiana. Posteriormente, se eliminan todos los valores de posición, velocidad y se reajustan las marcas de tiempo al nuevo tamaño de la trayectoria. Los perfiles de velocidades cartesianas correspondientes a la trayectoria de entrada y a la trayectoria después de eliminar la pausa inicial y final se muestran en la Figura 4.7.

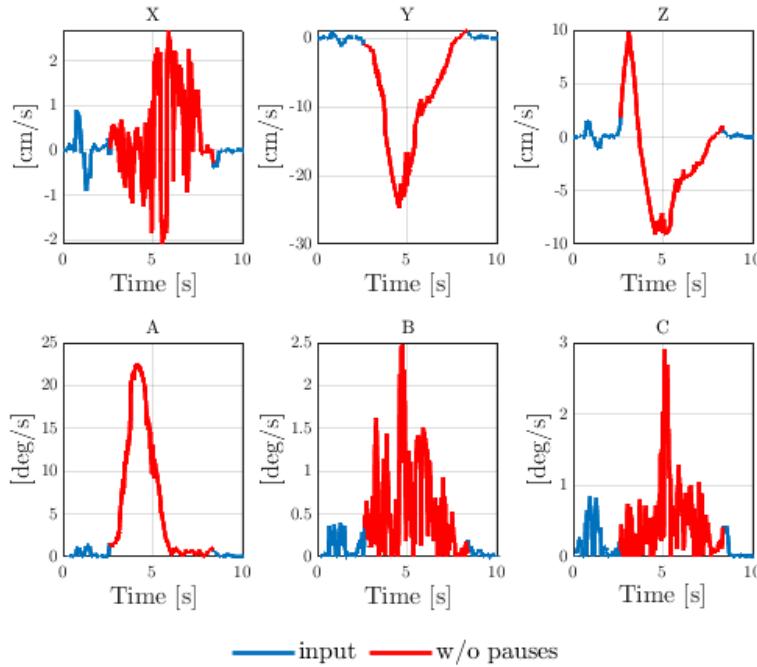


Fig. 4.7. Perfil de velocidades cartesianas antes y después de eliminar las pausas inicial y final.

La función `CompleteVelAcc` implementa el cálculo de las velocidades articulares, el cálculo de las velocidades cartesianas se recoge en la función `CompleteCartesian` y la función global en la función `DeleteInitialEndPauses`, todas ellas dentro de la clase de MATLAB `LiwaTrajectory`, que incluye además de estos y otros métodos, propiedades de la trayectoria como el vector de tiempos `t`, las posiciones articulares `q` y cartesianas `x`, las velocidades articulares `qdot` y cartesianas `xdot` y el número de puntos de la trayectoria `npoints`.

### Ajuste a un arco de circunferencia

La trayectoria grabada ha sido la ejecutada activamente por el paciente sobre el robot, manteniendo el codo en una posición fija y la muñeca sin flexionar. Esto implica que la palma de la mano del paciente debería haber estado durante toda la trayectoria a una distancia constante al codo, dibujando un arco de circunferencia. Se recuerda que la

trayectoria de la mano del paciente concuerda con la trayectoria que ha seguido la herramienta del robot y la cual ha sido grabada, por estar ésta sujetada por el paciente. Sin embargo, en la Figura 4.6c se observaba que la trayectoria dibujada no es exactamente un arco de circunferencia. Esto se puede deber a muchos motivos, pero el más probable es que el paciente se haya desplazado o girado una distancia muy corta durante la grabación de la trayectoria debido a tener que mover el robot con su propia fuerza.

Que haya pequeñas variaciones respecto a una trayectoria de arco de circunferencia no implica que la trayectoria no sea válida, ya que igualmente nos está proporcionando información muy útil respecto al paciente, como el rango de movimiento que es capaz de realizar, la medida aproximada de su antebrazo y una posición aproximada de dónde ha estado su codo durante la adquisición de datos. Podemos obtener todos estos datos de la trayectoria capturada, y luego reconstruirla con la forma que debería tener idealmente, aprovechando además para normalizar la velocidad angular.

En este paso se busca por lo tanto el circunferencia que más se ajuste a los puntos de la trayectoria y luego se halla el arco de circunferencia que empiece en el punto más cercano al inicial de la trayectoria grabada y finalice en el punto más cercano al último capturado.

Primero, se seleccionan entre 15 y 50 posiciones cartesianas equidistantiadas en el tiempo de la trayectoria capturada. En ese rango de puntos no cambia significativamente el resultado de centro y radio obtenidos para la mayoría de trayectorias, siendo el error un poco menor cuantos más puntos se cojan, pero cogiendo más puntos se incrementa demasiado el tiempo de cálculo siendo el resultado del algoritmo prácticamente el mismo. Esos  $n$  puntos generarán  $n - 1$  segmentos entre cada par de puntos. De cada uno de esos puntos se toman sólo las coordenadas de traslación que deberían estar cambiando en el tiempo, es decir, X y Z, como se observaba en la Figura 4.2. Para las ilustraciones se han utilizado tan solo 15 puntos con el objetivo de que se vea con mayor claridad el método seguido, pero en el método final se ha dejado como un parámetro configurable cuyo valor por defecto es 40.

A continuación, para cada uno de los segmentos, se calcula la recta perpendicular que pasa por el punto medio entre los dos extremos, es decir, la mediatrix.

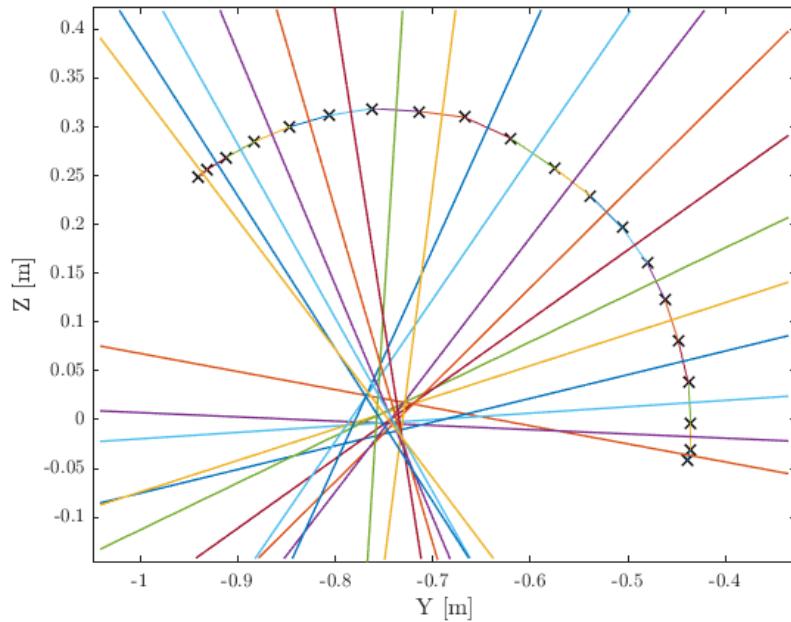


Fig. 4.8. Puntos, segmentos y mediatrices de los puntos seleccionados dentro del plano YZ.

El cálculo del centro de la circunferencia  $C$  se hace basándose en el método para hallar la circunferencia que pasa por tres puntos, que toma el punto de corte de las mediatrices de los segmentos, pero extendiéndolo a un número mayor de puntos. Se tomará como centro de la circunferencia el centroide de las intersecciones de las mediatrices entre sí, calculada como la media de las posiciones de los puntos de intersección  $p_{\cap_i}$ .

$$C = \frac{1}{n_{\cap}} \cdot \sum_{i=1}^{n_{\cap}} p_{\cap_i} \quad (4.8)$$

donde  $n_{\cap}$  es el número de intersecciones entre las  $n - 1$  rectas perpendiculares a los  $n - 1$  segmentos mencionados anteriormente,  $n_{\cap} = (n - 1) + (n - 2) + \dots + 3 + 2 + 1$ , y  $n$  es el número de puntos utilizados, que en este caso eran 50. Para hallar el radio del circunferencia, se calcula la media de las distancias desde cada punto de la trayectoria utilizado  $p_i$  al centro calculado  $C$ .

$$R = \frac{1}{n} \cdot \sum_{i=1}^n |p_i - C| \quad (4.9)$$

Por último, el principio y el final del arco de circunferencia se hallan intersecando las rectas que conectan el centro  $C$  y los puntos inicial y final de la trayectoria capturada con la circunferencia calculada descrita por su centro  $C$  y su radio  $R$ . Una vez descritos los parámetros del circunferencia  $C, R, p_{ini}, p_{fin}$ , se generan tantos puntos equidistantes en el espacio como puntos tuviera la trayectoria original, y se les asignan marcas temporales también equidistantes. Con este paso se está fijando la velocidad angular de toda la trayectoria a la velocidad angular media. Será de utilidad en el paso siguiente, ya que modificar la velocidad de la trayectoria completa será más sencillo al estar esta ya unificada.

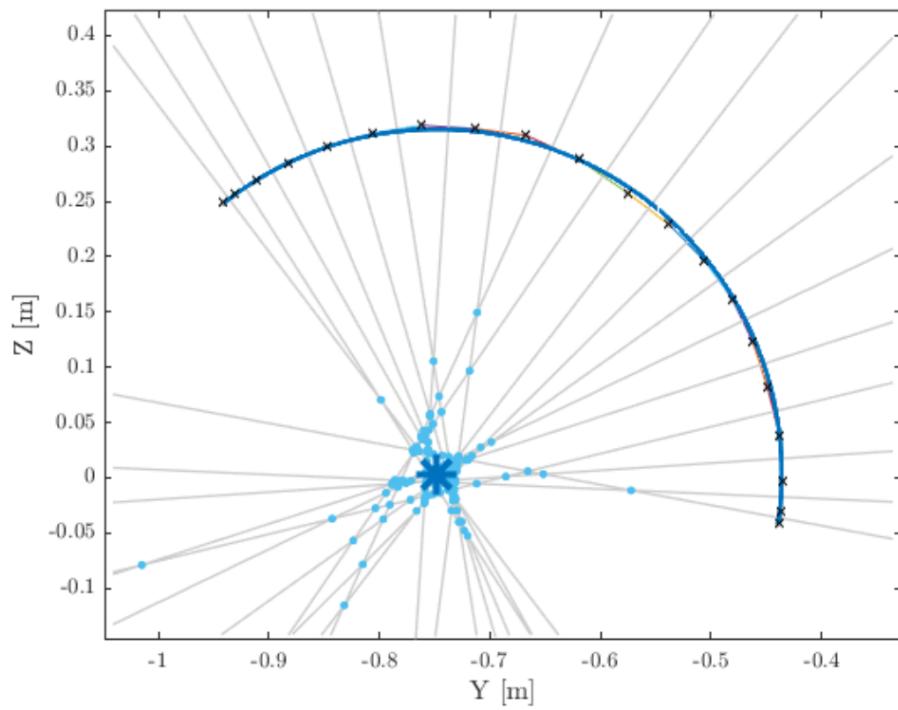


Fig. 4.9. Intersecciones en azul claro, centro y arco de circunferencia en azul oscuro.

Transformar el arco bidimensional en vectores seis-dimensionales de tipo *screw* será sencillo teniendo en cuenta que el movimiento pertenece al plano YZ y la herramienta solo rota en X. Se fija la posición X de toda la trayectoria a la posición en X inicial, la orientación entorno al eje X (coordenada A) se deja tal y como se capturó, y las coordenadas B y C se fijan para toda la trayectoria a la orientación en la posición inicial.

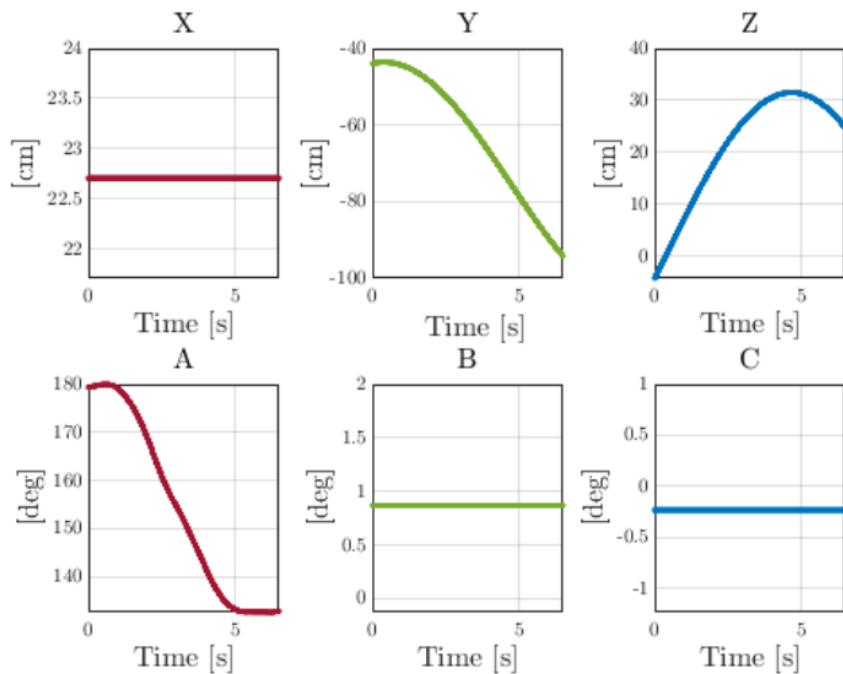


Fig. 4.10. Coordenadas cartesianas de la trayectoria tras ajustarla a un arco de circunferencia.

La función `FitToCircle` de la clase `IiwaTrajectory` recoge este paso del ajuste, devolviendo la trayectoria modificada y la velocidad angular en el plano YZ de la misma.

### Ajuste de la velocidad angular

Uno de los objetivos del proyecto mencionados en el capítulo 1 es que la velocidad de movilización del brazo del paciente sea regulable en velocidad. Con este fin, se introduce este paso en el ajuste de la trayectoria de la herramienta, que se modifica para adaptarla a la velocidad de giro requerida por el terapeuta. En la fase de ajuste de la trayectoria, se puede establecer cualquier velocidad de giro siempre que esta sea positiva, dada en grados por segundo. Sin embargo, posteriormente habrá que tener en cuenta que el robot no podrá reproducir la trayectoria a cualquier velocidad debido a limitaciones articulares y del control. Partiendo de una trayectoria cuya velocidad angular  $\omega_{in}$  es continua y conocida, y una velocidad angular requerida  $\omega_{out}$ , se calcula el factor de incremento de velocidad.

$$f_\omega = \frac{\omega_{in}}{\omega_{out}} \quad (4.10)$$

Basta con multiplicar cada una de las marcas de tiempo por este factor para cambiar implícitamente la velocidad de la trayectoria. Después, se volverá a ejecutar el algoritmo de remuestreo explicado anteriormente para conservar la frecuencia de muestreo previa.

$$t_i = t_i \cdot f_\omega \quad (4.11)$$

Con este paso se finaliza el ajuste en coordenadas cartesianas, habiendo obtenido una trayectoria que cumple los requisitos mencionados:

- Ajustada a las medidas del paciente por partir de la trayectoria capturada de su movimiento.
- Perteneciente a un solo plano.
- Ajustada a un arco de circunferencia.
- Con velocidad angular continua y conocida, fijada desde la interfaz.

A continuación, se debe transformar esta trayectoria en el espacio euclídeo a una trayectoria en el espacio de configuraciones, asegurando que se encuentra dentro de los límites articulares del robot y que no se generan movimientos bruscos que puedan dañar al paciente.

### Transformación a coordenadas articulares

Para poder comandar la trayectoria al robot es necesario encontrar previamente un vector de configuraciones articulares alcanzables que hagan que la herramienta se sitúe

en las posiciones deseadas en las marcas de tiempo indicadas. La primera idea que se viene a la cabeza para lograr esta transformación es utilizar la cinemática inversa, ya que como se ha explicado en la sección 2.2, con *screw theory* es posible obtener soluciones exactas para este problema. Sin embargo, el hecho de estar utilizando un robot redundante como el IIWA introduce ciertas complicaciones.

Para una pose cartesiana, en un robot redundante pueden existir infinitas configuraciones que coloquen la herramienta en la posición deseada. Aunque esto suele ser una ventaja, en el caso de la resolución de la cinemática inversa con *screw theory*, un método geométrico, nos vemos obligados a limitar la solución de alguna manera. La solución adoptada por Pardos-Gotor [67] para el IIWA consiste en fijar una de sus articulaciones siguiendo un criterio concreto y calcular la cinemática inversa para las otras seis.

Esto es una solución válida para cada una de las posiciones independientes de la trayectoria, sin embargo, no se garantiza que la configuración encontrada para cada pose esté en la vecindad de la configuración inmediatamente anterior. Se pueden producir cambios de configuración drásticos que serían irrealizables.

Se ha probado a hallar 32 soluciones diferentes para cada pose y a tratar de seleccionar de entre esas 32 una que asegurara que no había un cambio de codo entre posiciones consecutivas y que mantuviera alejado al robot de sus límites articulares. Sin embargo, no se encontró solución para todas las trayectorias, llegando en ocasiones a puntos en los que las 32 soluciones contenían un cambio de codo. Cuantas más soluciones calculemos más probabilidades habrá de asegurar que una de ellas sea continua a la anterior, pero en ningún caso tenemos la certeza de que una de ellas vaya a ser válida, a no ser que calculemos las infinitas soluciones.

Como esto no es una alternativa viable, se ha decidido utilizar la cinemática diferencial inversa (CDI) en lugar de la cinemática inversa, que mapea velocidades cartesianas a velocidades articulares utilizando la inversa de la matriz jacobiana geométrica. Integrando la velocidad articular en una posición articular conocida obtendremos cada una de las configuraciones articulares de la trayectoria articular que buscamos. Esta solución es útil además para realizar un control cinemático en velocidad articular.

En primer lugar, se deben calcular las velocidades cartesianas correspondientes a la trayectoria ajustada. En este caso no se puede utilizar el método basado en la CDD como se ha hecho anteriormente, porque no conocemos las posiciones ni las velocidades articulares para esta trayectoria. Por lo tanto, se ha de recurrir al cálculo por diferenciación de la posición cartesiana. Sin embargo, hay que tener en cuenta que no se pueden calcular incrementos de orientación por resta de vectores de orientación.

Para cada punto  $\xi_i$ , con componentes en posición  $v_i$  y orientación  $\omega_i$ , su velocidad cartesiana se puede calcular como:

$$\dot{\xi}_i = \begin{pmatrix} \dot{v}_i \\ \dot{\omega}_i \end{pmatrix} = \begin{pmatrix} v_{i+1} - v_i \\ rotA2B(\omega_i, \omega_{i+1}) \end{pmatrix} \cdot \frac{1}{t_{sample}} \quad (4.12)$$

donde  $t_{sample}$  es conocido y constante. El problema de la transformación de orientación resumido como  $rotA2B$  se ha resuelto transformando las orientaciones en matrices de rotación homogéneas, calculando la rotación necesaria para transformar una en otra, y transformando finalmente la matriz de rotación resultante de nuevo en su representación en ángulos de Euler. Se detalla la formulación en la sección 2, ecuaciones 2.1 y siguientes.

Las velocidades  $\dot{\xi}_i$  están expresadas en el  $SDC$  de la primera posición de la trayectoria capturada. Para poder utilizar la matriz jacobiana inversa  $J_B^{-1}$  y transformarlas en velocidades articulares las velocidades deben estar expresadas en el sistema de coordenadas de la base  $SDC_B$ .

$$\dot{\xi}_i\{B\} = \begin{pmatrix} \dot{v}_i + v_i \\ \dot{\omega}_i - \omega_i \times \dot{v}_i \end{pmatrix} \quad (4.13)$$

Una vez conocidas las velocidades cartesianas en cada punto expresadas en el  $SDC_B$ , se debe calcular la matriz jacobiana geométrica para cada punto  $J_B(\theta_i)$ , y después, su inversa. Sin embargo, se necesita la configuración en el inicio para poder calcular la primera matriz jacobiana. Para obtener la configuración inicial  $\theta_0$  se utiliza de nuevo CDI aprovechando que se conocen las configuraciones articulares y las posiciones cartesianas de la trayectoria capturada. Utilizando la primera pose cartesiana de la trayectoria capturada  $\xi_{cap(0)}$  y la primera pose cartesiana de la trayectoria ajustada  $\xi_0$ , y la misma fórmula de la Ec. 4.12 se calcula primero la velocidad cartesiana  $\dot{\xi}_{cap2circ}$  que sería necesaria para ir desde el punto  $\xi_{cap(0)}$  al punto  $\xi_0$  en un tiempo inventado  $t = 1$ , expresada en el  $SDC_B$ .

$$\xi_{cap(0)} = \begin{pmatrix} v_{cap(0)} \\ \omega_{cap(0)} \end{pmatrix}, \quad \xi_0 = \begin{pmatrix} v_0 \\ \omega_0 \end{pmatrix} \quad (4.14)$$

$$\dot{\xi}_{cap2circ} = \begin{pmatrix} \dot{v}_{cap2circ} \\ \dot{\omega}_{cap2circ} \end{pmatrix} = \begin{pmatrix} v_0 - v_{cap} \\ rotA2B(\omega_{cap}, \omega_0) \end{pmatrix} \cdot \frac{1}{t} \quad (4.15)$$

$$\dot{\xi}_{cap2circ}\{B\} = \begin{pmatrix} \dot{v}_{cap2circ} + v_{cap(0)} \\ \dot{\omega}_{cap2circ} - \omega_{cap(0)} \times \dot{v}_{cap2circ} \end{pmatrix} \quad (4.16)$$

Una vez obtenido el valor de la velocidad cartesiana necesaria para ir desde el punto inicial de la trayectoria capturada al punto inicial de la trayectoria ajustada, se halla la configuración inicial de la trayectoria ajustada  $\theta_0$  utilizando la CDI y teniendo en cuenta que se conoce la configuración inicial de la trayectoria capturada  $\theta_{cap(0)}$ .

$$\dot{\theta}_{cap2circ} = (J_B(\theta_{cap(0)})^{-1})^{-1} \cdot \dot{\xi}_{cap2circ}\{B\} \quad (4.17)$$

En la sección 2.2.5 se describe el cálculo de una aproximación de la inversa de la jacobiana geométrica para su aplicación a la CDI, permitiéndonos calcular la velocidad articular necesaria  $\dot{\theta}_{cap2circ}$  para llegar a la primera posición de la trayectoria ajustada desde la primera posición de la trayectoria capturada. Integrando esta velocidad en la configuración articular inicial capturada con el periodo de tiempo  $t$  que habíamos elegido antes, se obtiene la primera configuración articular  $\theta_0$  de la trayectoria ajustada. El valor

elegido para  $t$  realmente no es significativo, ya que primero se divide y luego se multiplica por él, por lo que se ha escogido un valor de  $t = 1$  que simplifica los cálculos.

$$\theta_0 = \theta_{cap(0)} + \dot{\theta}_{cap2circ} \cdot t \quad (4.18)$$

Para obtener resultados más precisos, se dividirá la distancia total entre  $\xi_{cap(0)}$  y  $\xi_0$  en fragmentos de 0.01mm, y se actualizará el jacobiano geométrico en cada iteración. El procedimiento será el mismo que el explicado pero realizado varias veces. Como la distancia entre ambas posiciones no debería ser muy grande, podemos dividir la distancia en períodos cortos sin incrementar significativamente el tiempo de computación. Sin embargo, si estas posiciones estuvieran más separadas por una traslación de la trayectoria completa o aplicando este método en otra aplicación, el tiempo de cómputo será mayor o habrá que dividir la distancia en fragmentos de mayor tamaño. En este caso, el tiempo de cálculo de la configuración articular inicial  $\theta_0$  no supera 1ms por lo que no será un problema.

Una vez se ha obtenido la primera configuración de la trayectoria, obtener las demás será sencillo, utilizando un procedimiento similar al anterior para cada punto. Se toman como entradas el vector de velocidades cartesianas expresadas en el  $SDC_B$  calculado anteriormente (Ec. 4.12 y 4.13), la configuración inicial de la trayectoria  $\theta_0$ , y el tiempo de muestreo  $t_{sample}$ . Para  $i \in (1, n_{points} - 1)$  se obtendrán el resto de configuraciones, limitando la posición o la velocidad cuando sea necesario.

$$\dot{\theta}_i = J_B(\theta_{i-1})^{-1} \cdot \dot{\xi}_i \quad (4.19)$$

$$\theta_i = \theta_{i-1} + \dot{\theta}_i \cdot t_{sample} \quad (4.20)$$

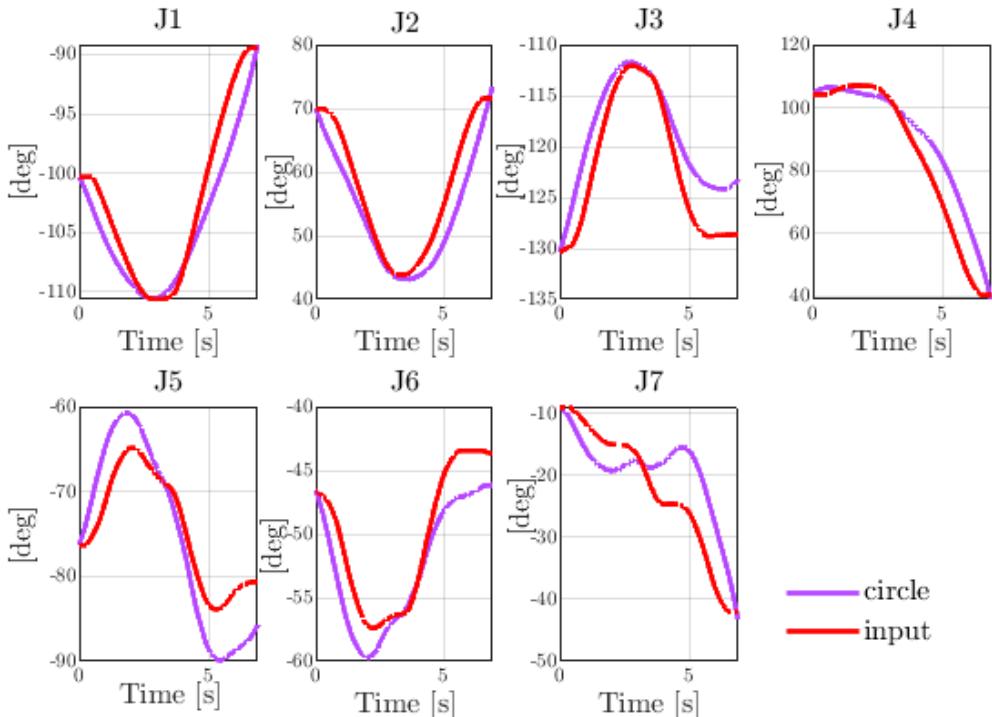


Fig. 4.11. Trayectoria articular capturada sin pausas inicial y final (rojo) y trayectoria articular correspondiente al ajuste a un arco de circunferencia (morado).

En la Figura 4.11 vemos cómo ambas trayectorias articulares tienen características similares pese a no corresponder al mismo vector de poses cartesianas, y a ser un robot redundante con infinitas soluciones para cada pose. Esto se debe al uso de la CDI partiendo de una configuración inicial similar a la configuración inicial de la trayectoria capturada. Si se hubiera utilizado CI, la solución para la primera posición ajustada podría ser completamente diferente a la primera configuración capturada, y al ser necesario que las siguientes configuraciones se parezcan a la anterior, podría llevar en algún punto de la trayectoria a una configuración inalcanzable por parte del robot sin un cambio de codo. La CDI asegura la continuidad de la trayectoria en toda su extensión, al tiempo que tiene una muy baja probabilidad de que no exista una solución para alguno de los puntos que no implique cambio de codo, ya que se utiliza la trayectoria capturada como base, y esta obviamente sí es alcanzable por el robot.

Todos estos cálculos se han implementado en la clase `IiwaTrajectory`, en la función general `CompleteJoint`, que utiliza a su vez multitud de funciones implementadas en la clase `IiwaScrewTheory`.

Para comprobar la precisión del algoritmo se ha transformado de nuevo la trayectoria articular obtenida con CDI en una trayectoria cartesiana utilizando CD. La CD por *screw theory* tiene error nulo, así que la diferencia entre estas dos trayectorias corresponderá con el error que introduce el cálculo de la CDI utilizando la pseudoinversa de la matriz jacobiana.

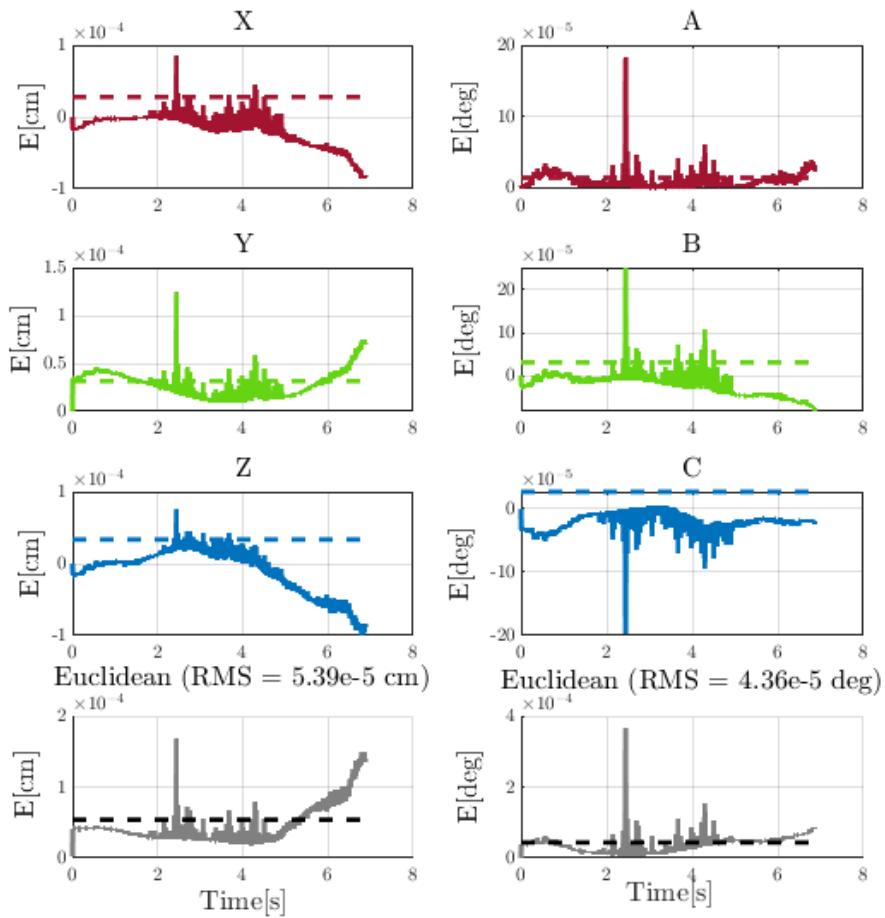


Fig. 4.12. Error entre la trayectoria cartesiana original y la CD de la trayectoria articular calculada por CDI.

En la Figura 4.12 se observa el error entre la trayectoria cartesiana ajustada a un arco de circunferencia y la cinemática directa de la que hemos calculado utilizando cinemática diferencial inversa. Este error se debe principalmente a dos causas:

- El cálculo de la matriz jacobiana se realiza solamente una vez para cada punto, es decir, cada 5 ms. Podríamos reducir el error recalculando la jacobiana geométrica más frecuentemente.
- El uso de la pseudoinversa de la jacobiana. Este problema es inevitable teniendo en cuenta que utilizamos un robot con 7 GDL, cuyo jacobiano mide necesariamente  $6 \times 7$  y cuya inversa no se puede calcular, sólo se puede calcular la pseudoinversa.

Se ha optado por calcular la jacobiana tan sólo una vez para cada posición, y no cada menos tiempo. El error introducido está en el orden de las diezmilésimas de milímetro, un error muy aceptable para esta aplicación.

## Spline

Una vez se ha hallado una lista de configuraciones articulares que colocan al robot en las poses cartesianas requeridas, se debe adaptar a las limitaciones dinámicas del robot. Para asegurar que el robot sea capaz de realizar la trayectoria, no sólo basta con que la trayectoria sea continua, sino que también hay que asegurar la suavidad, es decir, que la velocidad y la aceleración estén acotadas. Para conseguir esto se ha utilizado un spline, una aproximación a la trayectoria original formado por un conjunto de polinomios de orden bajo, cada uno de ellos con ciertas condiciones de contorno que aseguran la adecuación a la señal original, la continuidad entre ellos y la suavidad de la curva completa [68]. Los splines suelen buscar la minimización de diferentes tipos de error entre la salida y la señal original, priorizando la adaptación a la entrada o la suavidad de la salida según preferencia.

El spline utilizado se basa en un interpolador cúbico implementado en MATLAB [69], que se ha incluido con ciertas adaptaciones dentro de la función `BoundedSpline` de la clase `IiwaTrajectory`. La función principal toma como entrada la trayectoria de posiciones articulares con sus respectivas marcas de tiempo, la divide en  $n_{seg}$  segmentos, y trata de encontrar un polinomio de orden 3 para cada segmento, de tal manera que se aproxime lo máximo posible a la trayectoria de entrada. Se le pueden indicar condiciones de posición, velocidad y aceleración tanto inicial como final de la trayectoria total, así como las marcas temporales concretas en que finaliza cada segmento y comienza el siguiente.

El error que se minimiza en este caso es el error entre el spline y los datos de entrada, así como la integral de la sobreaceleración -*jerk*- al cuadrado de la salida. El *jerk* se refiere al cambio de aceleración en el tiempo, es decir, su derivada [70]. Minimizando el *jerk*, se consigue generar movimientos menos bruscos, y aunque se alcancen grandes velocidades no habrá sacudidas en toda la trayectoria. El parámetro *smoothing* de esta función regula qué peso se le da a la minimización del *jerk* al cuadrado y qué peso se le da a la adecuación del spline a la trayectoria. Un parámetro *smoothing* alto, del orden de  $10^{-3}$ , generará splines menos ajustados a la trayectoria original, pero más suaves; mientras que un parámetro *smoothing* bajo, del orden de  $10^{-9}$  tendrá como salida splines mucho más fieles a la entrada, pero también incluirán movimientos bruscos si la entrada los tenía. Reducir el *jerk* puede reducir así mismo la aceleración, ya que al estar la trayectoria acotada en el tiempo, puede ser imposible alcanzar una aceleración demasiado alta. De la misma manera, un parámetro de *smoothing* muy alto puede limitar la velocidad, y en casos extremos, la posición alcanzada. Por este motivo hay que tener cuidado y elegir un *smoothing* lo suficientemente alto como para suavizar la trayectoria pero sin ser demasiado alto, para que el spline siga representando adecuadamente la trayectoria de entrada.

Las condiciones iniciales y finales se han fijado teniendo en cuenta que al inicio y al final de la flexión el robot está parado, por lo que su velocidad es  $\vec{0}$ . Además, se conocen la posición inicial y final de la trayectoria, las cuales se introducen como condiciones de contorno del algoritmo. Es importante señalar que la función no permite indicar condi-

ciones en puntos intermedios, como una posición concreta o una velocidad nula para una marca de tiempo dada de la trayectoria. Este es el motivo principal por el que se adapta a un spline la flexión antes de reflejar la trayectoria para convertirla en una de flexo-extensión. En fases iniciales del trabajo se reflejaba la trayectoria antes de transformarla en un spline, y cuando se utilizaban parámetros de *smoothing* lo suficientemente altos como para garantizar la suavidad, la velocidad durante la parada entre flexión y extensión no era siempre nula, lo cual es incompatible con los objetivos del trabajo. Invirtiendo el orden de los ajustes y transformando primero la trayectoria de flexión en un spline cuyas velocidades inicial y final se fijan a 0, para después reflejarla y obtener la trayectoria de flexo-extensión se asegura que la velocidad en la parada entre flexión y extensión es siempre nula.

El número de segmentos en que se divide la trayectoria global se ha dejado como un parámetro configurable cuyo valor por defecto es de 30 segmentos. En la Figura 4.13 se muestra el spline con sólo 10 segmentos para que se visualice mejor las diferencias entre entrada y salida. El parámetro *smoothing* se ha fijado por defecto a  $10^{-4}$ , aunque se ha incluido la opción de modificarlo desde la interfaz para comparar la suavidad del movimiento en el robot real usando diferentes valores.

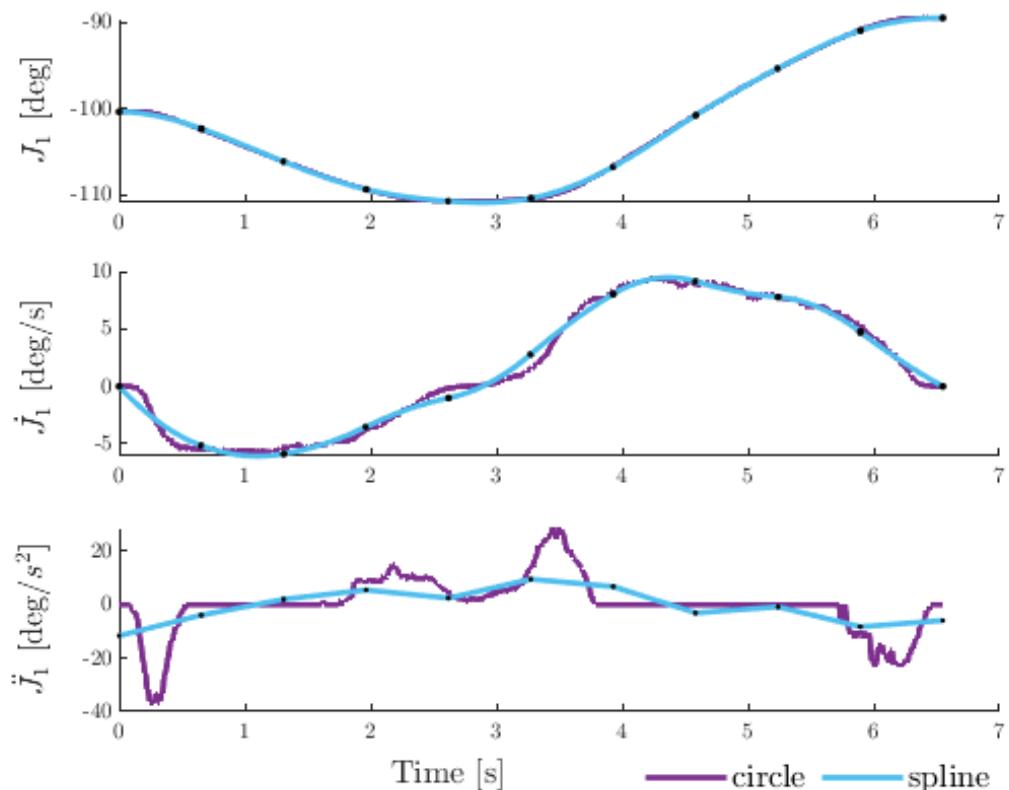


Fig. 4.13. Trayectoria de la articulación J1: posición (arriba), velocidad (medio) y aceleración (abajo), antes y después del spline cúbico.

En la Figura 4.13 se aprecia también claramente el orden de los polinomios. Los poli-

nomios interpoladores en posición son de orden 3; en velocidad, de orden 2; y en aceleración, de orden 1, una recta. El resto de trayectorias articulares interpoladas no se muestran por producir resultados muy similares visualmente.

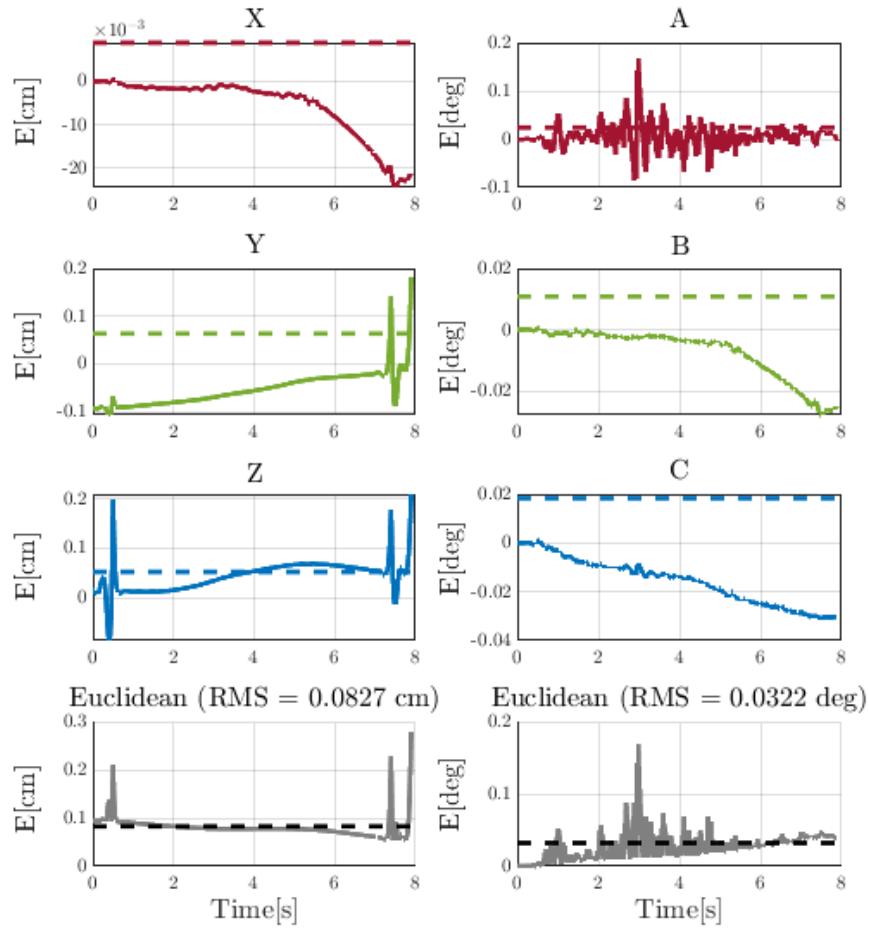


Fig. 4.14. Error en coordenadas cartesianas entre la trayectoria ajustada a un arco de circunferencia y el resultado del spline.

El error introducido por el spline es algo superior al error introducido por la CDI, siendo sobre todo más alto en el inicio y el final de la flexión, debido probablemente a la limitación de la aceleración inicial y final que introduce el parámetro *smoothing*. Aun así, se encuentra en el orden de las centésimas de centímetro, que sigue siendo suficientemente bajo como para no deformar la forma de la trayectoria de movilización del paciente.

## Obtención de la trayectoria de flexo-extensión

Ahora que ya se tiene una trayectoria de flexión adaptada al paciente, sin pausas, ajustada a las restricciones de forma y plano, a los límites articulares y a los límites dinámicos del robot y con la velocidad que se haya indicado en la interfaz, sólo falta convertirla en una trayectoria de flexo-extensión. Para ello, sencillamente se ha optado por reflejar la trayectoria articular calculada, añadiéndole las marcas de tiempo necesarias. Además, para diferenciar sus partes en el análisis posterior y para la comodidad del paciente, se añade una pausa de duración variable entre la trayectoria de flexión y la de extensión.

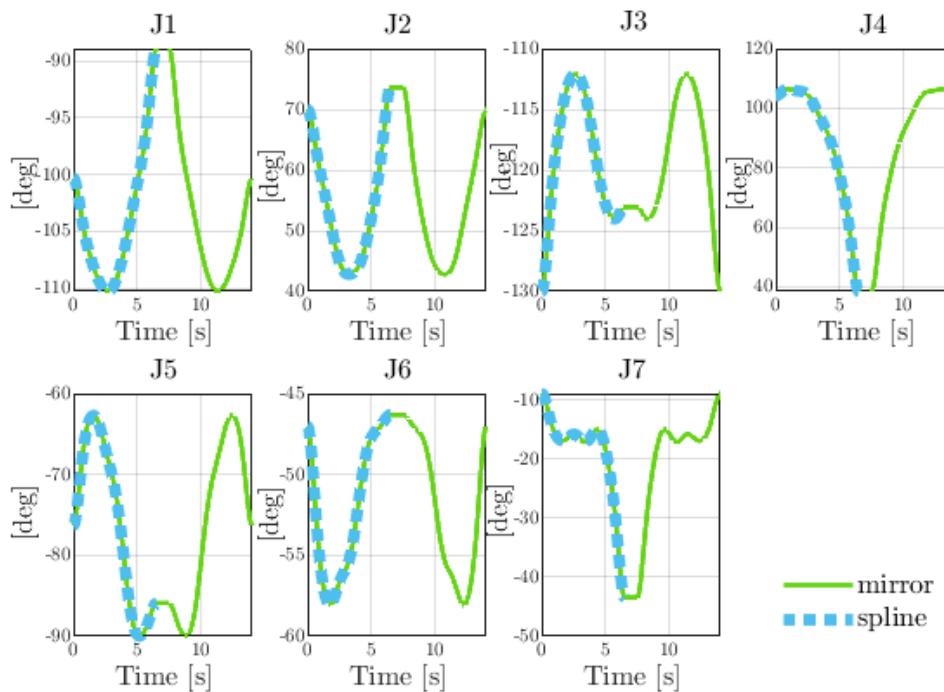


Fig. 4.15. Trayectoria articular tras añadir una pausa de 2 segundos y reflejar la entrada.

Finalmente, se calculan también las posiciones cartesianas por CD y las velocidades articulares por diferenciación como ya se ha explicado en el subapartado 4.1.2 de esta sección, y las correspondientes velocidades cartesianas para la trayectoria final de flexo-extensión. Estos valores nos permiten calcular el error en coordenadas cartesianas entre la trayectoria ajustada a un arco de circunferencia y la salida del spline (Figura 4.14). También serán útiles para representar la trayectoria y comprobar que el ajuste es correcto, y para llevar a cabo un control cinemático cartesiano como se explica en la sección 4.2. En la Figura 4.16 se observan las trayectorias de entrada, el ajuste a un arco de circunferencia y la salida del spline. La diferencia entre la entrada y el arco de circunferencia dependerá de cómo capture el paciente la trayectoria, mientras que la diferencia entre estas dos últimas se percibía mejor en la Figura 4.14.

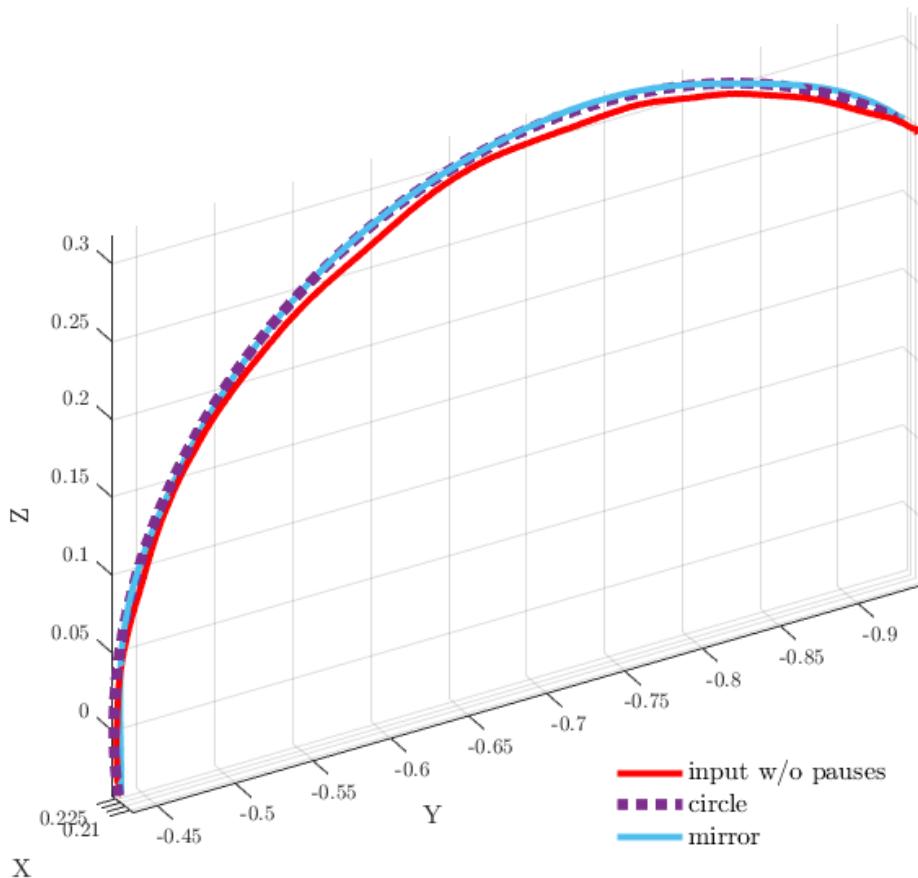


Fig. 4.16. Trayectorias cartesianas de entrada, tras el ajuste al circunferencia y tras el ajuste a spline.

## 4.2. Reproducción de trayectorias

Teniendo una trayectoria de flexo-extensión ya ajustada correctamente, y conociendo sus coordenadas articulares y cartesianas de posición y velocidad, nos centramos ahora en reproducir dicha trayectoria lo más rigurosamente posible, para movilizar de forma pasiva el brazo del paciente. La implementación de la reproducción de trayectorias se ha hecho siguiendo dos estrategias diferentes, que se han analizado y comparado entre sí.

Como se explicaba en la Sección 3.4, el armario controlador del IIWA viene con un sistema operativo de KUKA llamado Sunrise OS, que permite ejecutar aplicaciones programadas en Java, previamente cargadas en la controladora. Resultaría extremadamente complicado programar la totalidad del proyecto sobre este sistema operativo como aplicaciones de Java, y se ha descartado desde el inicio. Sin embargo, no deja de ser necesaria la integración del brazo robótico en la aplicación. Para ello se ha optado por utilizar una arquitectura de control basada en la relación cliente-servidor, en la que el servidor será un programa o interfaz de control ejecutándose sobre el Sunrise OS, y el cliente permitirá conectarse a éste desde un ordenador externo y controlar el robot. De esta manera, se sim-

plifica la integración del IIWA en un sistema global con características como presentación de gráficas de resultados, cálculos complejos en pocas líneas de código o interacción con otros sensores. Se han analizado dos alternativas basadas en el control del IIWA utilizando la arquitectura cliente-servidor:

- **iiwa\_stack**: Es un paquete desarrollado por Salvo Virga [26], que incluye dos partes principales:

1. Un complejo sistema de programas en Java, con un archivo principal que se ejecuta en la controladora del robot (servidor), que publica a través de la red *topics* de ROS con información del estado del robot, ofrece servicios para cambiar configuraciones y se suscribe a otros *topics* en los que el cliente puede publicar mensajes. Cuando recibe un mensaje, planifica una trayectoria y después la manda ejecutar.
2. Ejemplos de uso y otros nodos de ROS/C++ para interactuar con el programa de Java utilizado y otras funcionalidades.

- **FRI**: Interfaz de comunicación con la controladora ofrecida por KUKA, basada en el protocolo UDP, que consta también de dos partes:

1. Un sencillo programa de Java que publica por UDP el estado del robot y se queda a la escucha de nuevas posiciones recibidas como mensajes UDP. Cuando recibe una, sustituye la posición comandada por el planificador interno del robot directamente por la posición recibida.
2. Un sistema de clases de C++ que permite desde el cliente leer los mensajes publicados por la controladora sin mayor complicación, y enviar mensajes de control.

La principal diferencia entre los programas de Java de ambos clientes es que el *iiwa\_stack* puede recibir posiciones alejadas de la posición en la que está el robot, y es el propio programa el que planifica la trayectoria hasta allí; mientras que el segundo requiere necesariamente recibir posiciones alcanzables, ya que estas se van a comandar directamente al brazo robótico, sin planificación de trayectorias intermedia.

Además, el paquete *iiwa\_stack* puede considerarse un sistema de más alto nivel que la interfaz FRI, ya que además de proporcionar planificadores de trayectorias, permite utilizar el modo gravedad compensada mencionado anteriormente y ofrece servicios para cambiar configuraciones básicas del robot. Por su parte, FRI sólo ofrece lecturas de las posiciones y esfuerzos de las articulaciones, y permite comandar los mismos valores. No cuenta con planificador de trayectorias, ni modo gravedad compensada, y la única configuración que se puede cambiar es la frecuencia de comunicación con la controladora.

Ambas tienen sus ventajas e inconvenientes, por eso se ha analizado el desempeño del robot en la reproducción de trayectorias utilizando ambos sistemas. Primero se optó

por el paquete `iiwa_stack`, por ser más sencillo de utilizar en principio, y luego se pasó a utilizar FRI, que permite un control más preciso de las posiciones articulares del robot.

En ambos casos, previa a la movilización del paciente se debe comandar al robot a la posición inicial de la trayectoria, y ya desde ahí comenzar a enviar comandos al robot. El ajuste de la velocidad de giro se debe realizar también previo a la movilización. Esto significa que cuando se capture la trayectoria inicial bastará con realizar los primeros cuatro pasos del ajuste, y posteriormente cuando se elija la velocidad angular desde la interfaz, se terminará el ajuste realizando los últimos cuatro pasos de la sección anterior. En esta sección se parte de los datos ya ajustados por completo a la velocidad escogida.

Para estudiar la ejecución de cada modo de control lo más verazmente posible no nos hemos limitado a ejecutar una trayectoria y mostrar el error de seguimiento, ya que este podría verse influenciado por las condiciones concretas del experimento. En todos los análisis expuestos a continuación se han ejecutado 10 trayectorias en las condiciones que se determinen y se han comparado todas ellas con la trayectoria comandada y entre sí. Para cada conjunto de 10 experimentos se calculan dos tipos de error:

- Error de precisión: Muestra la diferencia entre cada trayectoria ejecutada y la comandada. Se calculan los errores mínimo, medio y máximo para cada punto y, para poder comparar conjuntos de experimentos entre sí, se utiliza el valor eficaz del error máximo como indicador.
- Error de reproducibilidad: Muestra la diferencia entre cada trayectoria ejecutada y la media de las trayectorias ejecutadas. De nuevo, el valor utilizado para las comparaciones es el valor eficaz del mayor error posible.

A continuación, se analizan ambas interfaces de control utilizando esta metodología.

#### 4.2.1. Opción 1: `iiwa_stack`

El `iiwa_stack` ofrece principalmente dos modos de comandar al robot: en posición articular o en velocidad articular. Tiene también un tercer tipo de comando posible, que permitiría establecer una posición y una velocidad deseadas, pero se ha comprobado que sólo sigue la referencia de posición cuando se le comandan puntos cercanos, por lo que no ofrece nada que los comandos en posición no puedan ofrecer para esta aplicación.

##### Comandar posiciones articulares

El primer modo de control con el que se ha experimentado se basa en comandar las posiciones articulares de la trayectoria ajustada en cada marca de tiempo establecida, mediante la publicación de mensajes en el topic `/iiwa/command/JointPosition` ofrecido por `iiwa_stack`. Utilizando este topic se le puede comandar al robot una posición articular cualquiera y el robot se moverá hasta ella, encargándose la aplicación de Java

de `iiwa_stack` de generar la trayectoria interpolada que lleva al robot hasta la posición deseada y de realizar el control en posición articular.

Se ha implementado un *action server* de ROS en la clase `StackMoveJTrajAction`, instanciada desde el nodo `iiwa_command_stack`, que recibe una trayectoria articular completa y comanda cada configuración articular a través del topic mencionado anteriormente en los instantes correctos, y devuelve como *feedback* del *action server* la posición articular en cada instante, y como resultado final tanto la trayectoria articular comandada como la trayectoria articular leída del robot durante la ejecución.

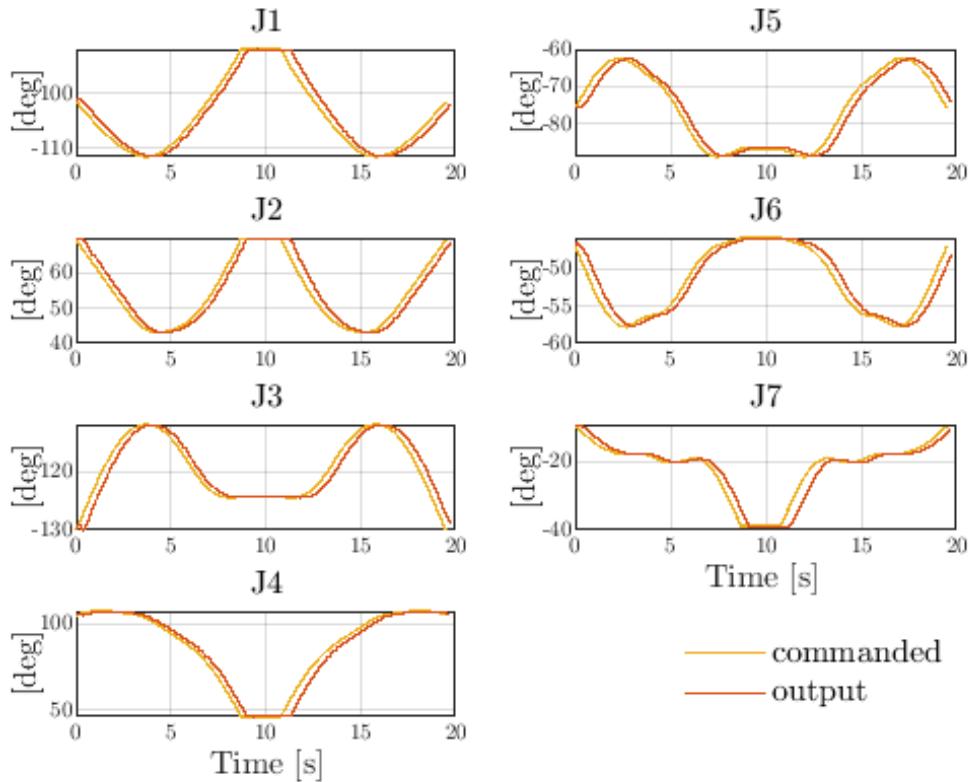


Fig. 4.17. Comparativa entre la trayectoria comandada y la ejecutada utilizando `iiwa_stack` y comandando en posición articular a velocidad baja ( $15^\circ/\text{s}$ ).

En la Figura 4.17, se muestran los resultados de posición articular al ejecutar la trayectoria ajustada a una velocidad angular de  $15^\circ/\text{s}$ . Se observa que las posiciones comandadas se alcanzan con cierta precisión, pero con un retraso perceptible a simple vista. Sin embargo, no es el retraso lo que más preocupa, sino el traqueteo que se observa en el robot real cuando se ejecuta la trayectoria a velocidad baja.

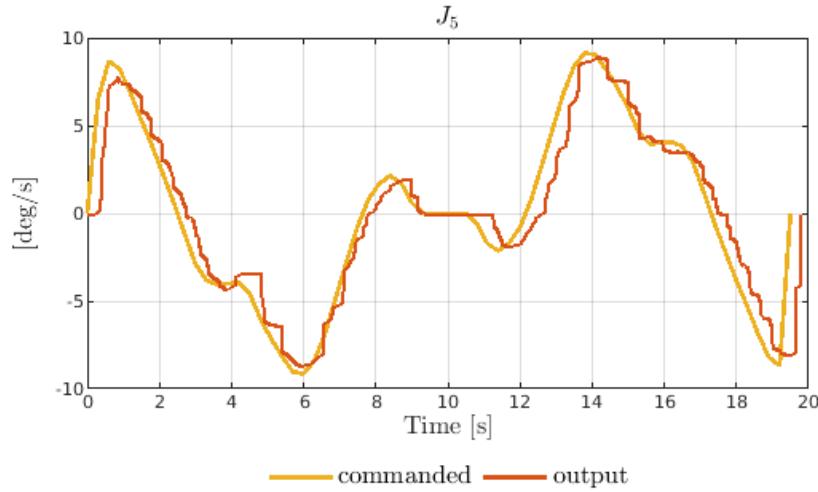


Fig. 4.18. Velocidad de la articulación 5 comandada y ejecutada utilizando iiwa\_stack y comandando en posición articular a velocidad baja ( $15^{\circ}/s$ ).

Si observamos la velocidad de la articulación 5 (Fig. 4.18) se distinguen ciertas aceleraciones y desaceleraciones periódicas, que se reflejan en el movimiento del robot real como movimientos oscilantes en velocidad. Estas oscilaciones son mayores cuanto más baja es la velocidad angular escogida, y se van reduciendo a medida que se aumenta la velocidad de la trayectoria. Intuitivamente, parece que el robot está intentando llegar lo antes posible a cada posición articular comandada, a continuación se para, y luego continúa con la siguiente posición que se le comande. Estudiándolo un poco más, se ha observado que al comandar el robot una única posición, y no una trayectoria completa, y analizar los resultados en velocidad articular, se genera un perfil de velocidades trapezoidal, como se ve en la Fig. 4.19.

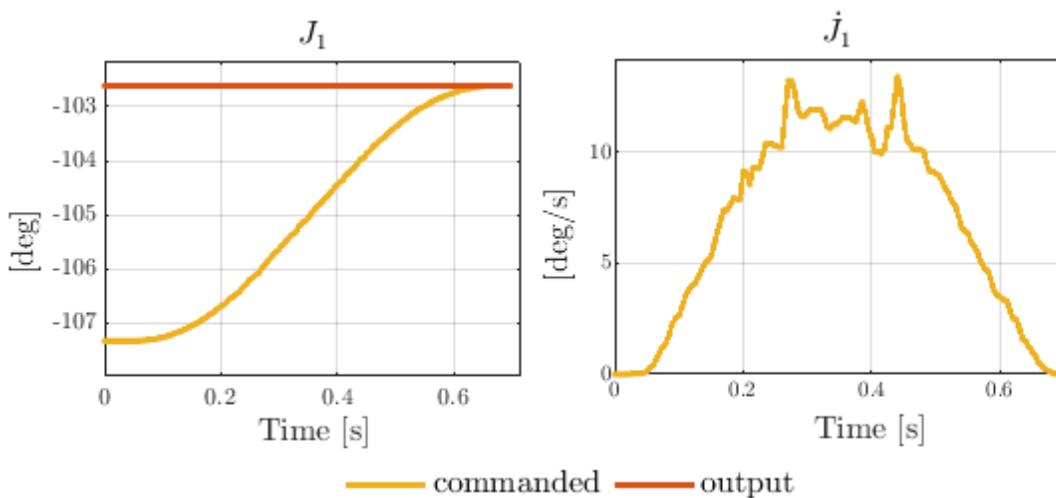


Fig. 4.19. Trayectoria de posición y velocidad de la articulación 1 ejecutada cuando se le comanda una única configuración articular.

Esto significa que aunque se consiguieran ajustar los tiempos de envío de posiciones para que el robot no esté en ningún momento parado, la velocidad que seguiría el robot sería igualmente oscilante. Para cada punto, se generaría un perfil trapezoidal con velocidad inicial y final nula, lo cual haría que, en teoría durante la totalidad de la trayectoria el robot acelerase y frenase pasando por momentos de velocidad nula. En la práctica, el retraso en la ejecución de la trayectoria mencionada anteriormente nos beneficia de alguna manera, ya que a partir de cierta velocidad empiezan a comandarse nuevos mensajes antes de que el robot haya alcanzado la posición anterior, y el perfil de velocidad trapezoidal se corta antes de terminar, reduciendo el traqueteo visible a bajas velocidades.

Este modo de comandar el robot se ha descartado para velocidades bajas porque en los objetivos del trabajo se incluía la necesidad de realizar una movilización suave, sin perturbaciones en velocidad, y la sensación de vibración es completamente incompatible con ese requisito. A velocidades altas no se presenta este problema, pero encontramos otro problema, y es que hay un retraso enorme entre señales, y el error de precisión es también muy alto. En la Figura 4.20 observamos el retraso entre señales a esta velocidad, que llega al orden de los segundos completos.

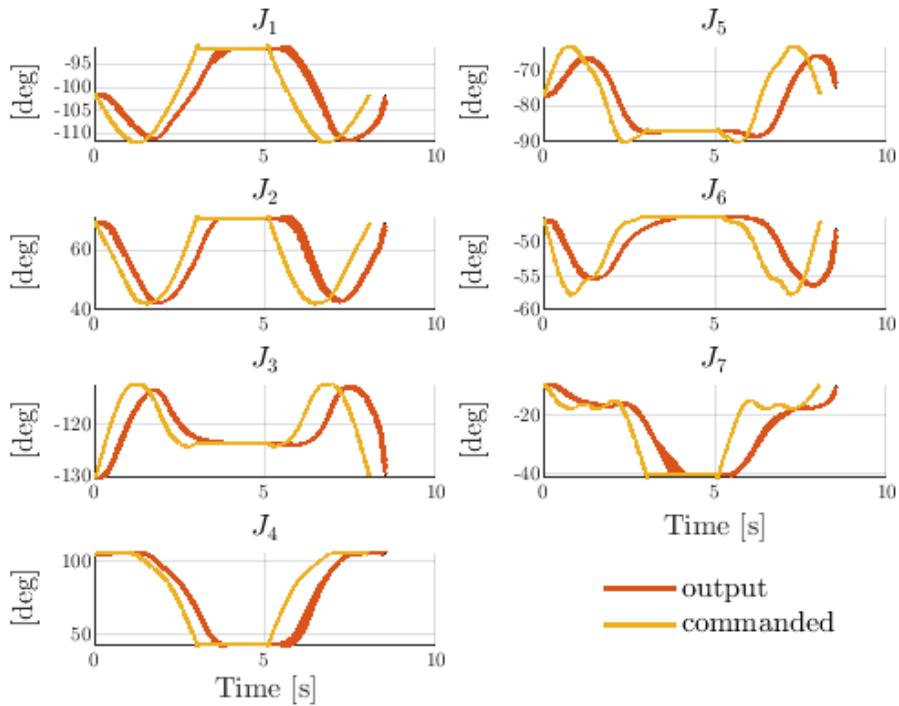


Fig. 4.20. Rango de trayectorias ejecutadas para una trayectoria comandada a velocidad 45 °/s.

Además, aunque coordinemos estas señales y calculemos el error de precisión entre las señales ignorando el retraso, dicho error asciende a aproximadamente 4 centímetros, como se ve en la Figura 4.21.

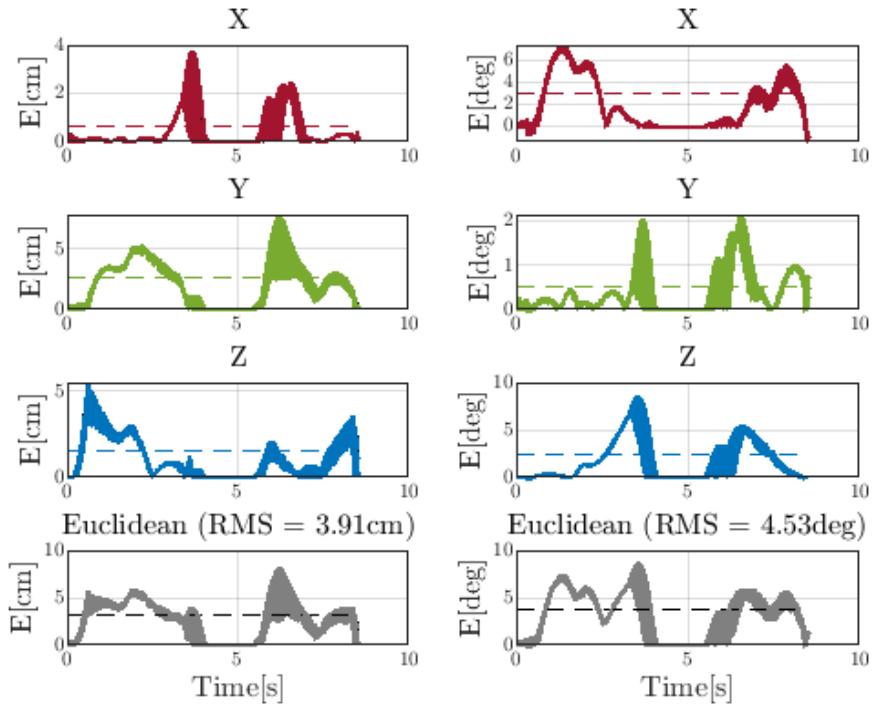


Fig. 4.21. Error de ejecución para una trayectoria comandada a velocidad 45 °/s.

Este modo de comandar no ha dado resultados positivos, descartando su uso para movilizaciones a velocidad igual o inferior a 15 °/s, y aunque se puede utilizar para velocidades más altas, se requiere una alternativa con menor error.

### Comandar velocidades articulares

Se ha probado también a comandar velocidades articulares, mediante la publicación de mensajes en el topic ofrecido por `iiwa_stack /iiwa/command/JointVelocity`, siendo el programa de Java que se ejecuta en el IIWA el que se encarga de generar una trayectoria interpolada y realizar un control que asegure que se alcanza esa velocidad de referencia.

La principal complicación que se encontró al utilizar este modo de control fue que por alguna característica del funcionamiento interno de la aplicación ejecutada en la controladora del `iiwa_stack`, no se le pueden comandar velocidades articulares cuando el robot está parado sin que éste genere un pico de aceleración que mueve al robot muy lejos de dónde se requiere. La solución a este problema ha resultado ser comandar, justo antes de comenzar a reproducir la trayectoria, una velocidad del orden de  $10^{-5}$  rad/s durante unas décimas de segundo. Como se desconoce el funcionamiento interno del planificador y del interpolador, no se ha resuelto por qué es necesario este proceso. Intuitivamente, probablemente esto ocurra porque el interpolador trata de establecer la velocidad comandada en el mínimo tiempo posible, y comanda una aceleración excesiva que aleja al robot de la velocidad requerida y, por ende, de la posición en la que debería estar. La solución propuesta,

aunque improvisada, funciona perfectamente y elimina ese problema por completo.

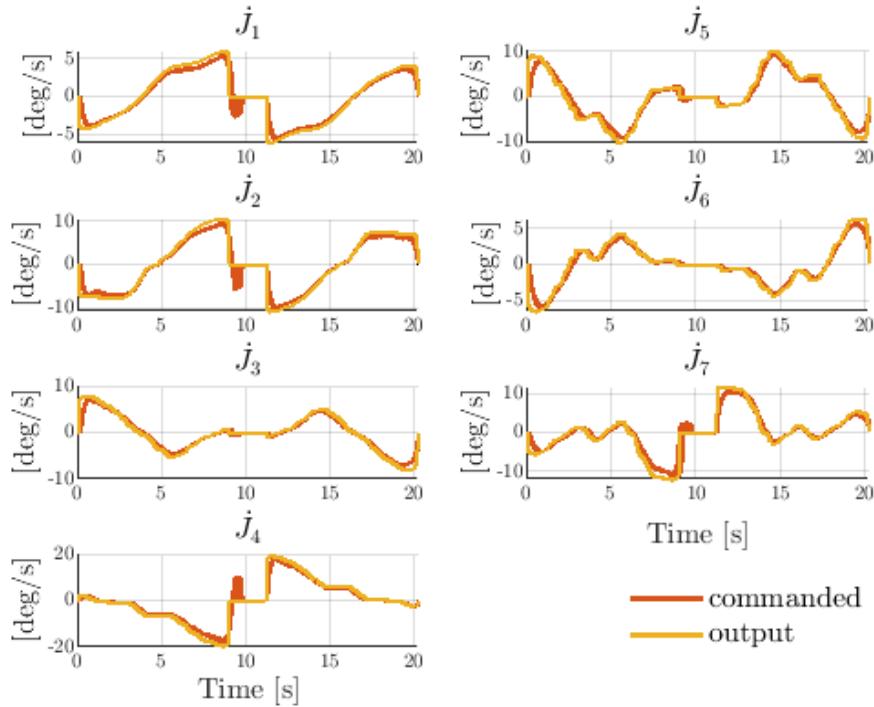


Fig. 4.22. Trayectoria de velocidad articular para una trayectoria ajustada a 15 °/s.

En la Figura 4.22 se presentan los resultados obtenidos al comandar una trayectoria de velocidades articulares correspondiente a una trayectoria ajustada a 15 °/s. Como se observa, sí se genera un movimiento suave, incluso a velocidades bajas, y no muestra vibraciones como el modo de control anterior. Sin embargo, el seguimiento de la velocidad no siempre es bueno, llegando a introducir errores de hasta 1.5 °/s, tal y como se ve en la Figura 4.23. Este error en velocidad articular se traduce en un error en posición articular que además será acumulativo, afectando a su vez al error de precisión cartesiana, como se muestra en la Figura 4.24. El valor eficaz del error euclídeo medio es de 4 cm, muy parecido al error que se obtenía comandando posiciones articulares en vez de velocidades (Figura 4.21), y más alto de lo que sería recomendable. Además, presenta un inconveniente respecto al modo anterior, y es el incremento en el error de reproducibilidad, que cuantifica si las trayectorias ejecutadas se parecen entre sí, se parezcan estas o no a la comandada. La posición en X, por ejemplo, hay repeticiones en las que tiene menos de medio centímetro de error, y otras en las que se aleja 5 cm de donde debería estar. Este problema es casi más grave que un alto error de precisión, ya que no permite comparar diferentes capturas de electromiografía o fuerza-par entre sí, puesto que no se está ejecutando la misma movilización del brazo del paciente.

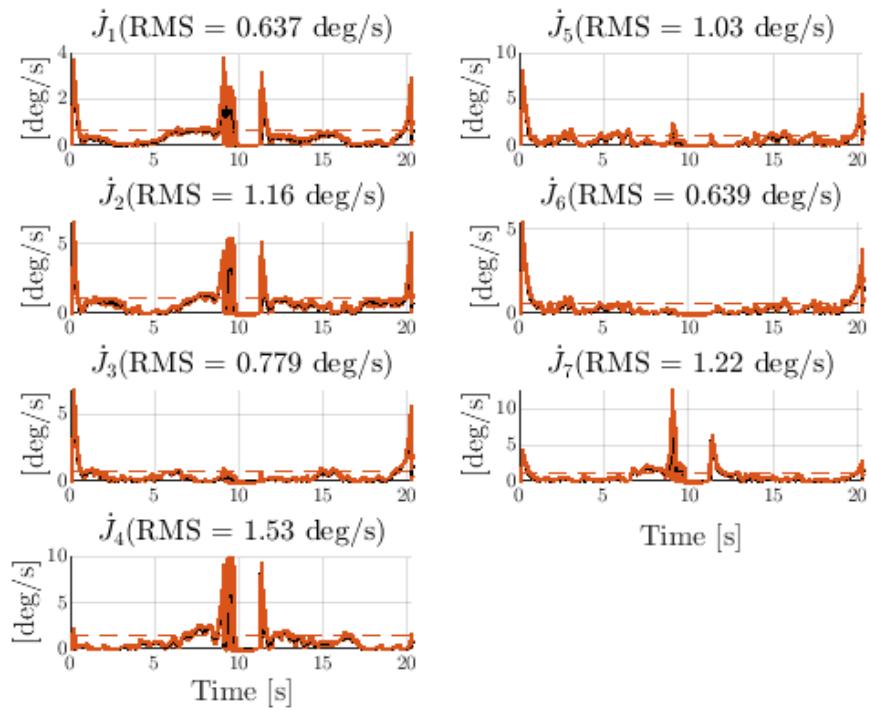


Fig. 4.23. Error de precisión de la velocidad articular comandada para una trayectoria a 15 °/s.

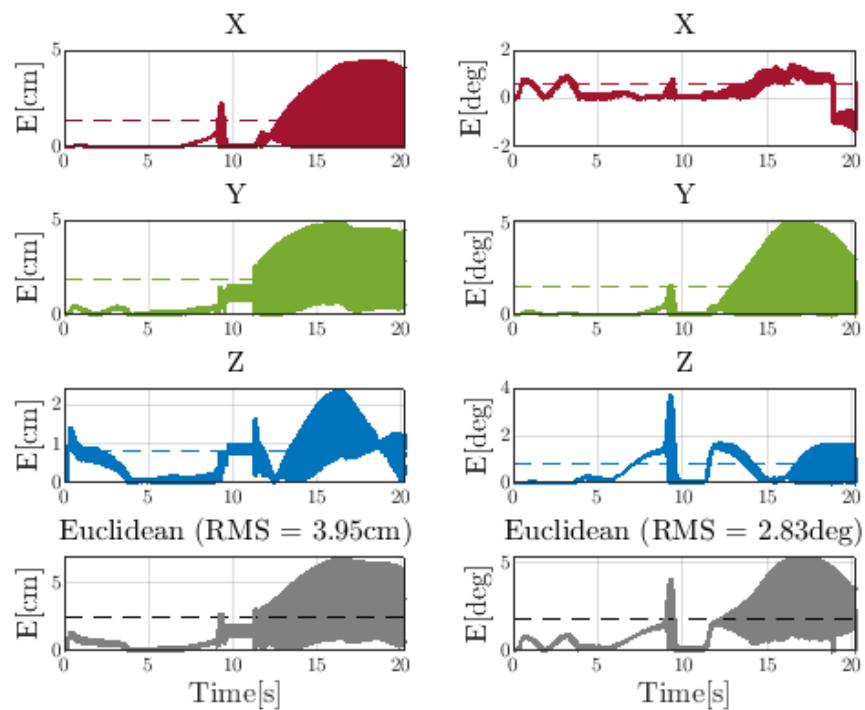


Fig. 4.24. Rango de error entre la trayectoria cartesiana correspondiente a la trayectoria de velocidades articulares comandadas y cada una de las 10 trayectorias ejecutadas a 15 °/s.

Presenta sin embargo la ventaja de que el error en posición no ha aumentado con la velocidad, aunque el retraso de la señal sí lo hace, además esta ventaja no compensa el alto error de precisión y reproducibilidad encontrados. Pese a que teóricamente el iiwa\_stack incluye un control en velocidad interno, debido a la mala calidad de los datos obtenidos se ha intentado implementar un control externo que lea las posiciones y velocidades actuales y las deseadas, y modifique la señal comandada teniendo en cuenta el error calculado. Sin embargo, aunque algunos intentos han resultado en la ligera reducción del error de precisión, la mayoría de ellos han resultado completamente fallidos, llegando a situaciones de desestabilización del robot completamente inaceptables para la aplicación.

Esto se debe principalmente a que el retraso entre señales es muy alto, de aproximadamente entre 100 y 500 ms según la velocidad que se le exija al robot. A 45 °/s, un retraso de 500 ms puede provocar un error en posición de varios centímetros, que al tratar de corregirlo en el instante siguiente, genere una velocidad tan alta que aleje al robot en sentido contrario, incrementando cada vez más la distancia a la trayectoria deseada y desestabilizando el sistema. En el capítulo 4 se debate sobre los posibles orígenes de este retardo.

Priorizando la seguridad sobre la precisión, se ha descartado el uso de cualquier tipo de control externo sobre el iiwa\_stack, para evitar situaciones en las que el robot se desestabilice. Llegamos por tanto a una situación compleja, en la que el retardo presente en todos los modos de control probados hasta ahora, y el mal desempeño de los algoritmos de control internos del iiwa\_stack a altas velocidades nos deja con errores euclídeos de aproximadamente 4 cm para trayectorias a 45 °/s. Aun así se debe mencionar que, a velocidades bajas, el modo de comandado por velocidades articulares sí genera trayectorias visualmente aceptables, que podrían servir para movilizar al paciente si no se contara con una alternativa.

Afortunadamente, sí existe otro método para comandar trayectorias al IIWA que, en principio, debería introducir mucho menos retraso y permitir un control en tiempo real del robot.

#### 4.2.2. Opción 2: Fast Robot Interface

*Fast Robot Interface* (FRI) es una interfaz de comunicación proporcionada por KUKA que permite comandar posiciones o esfuerzos articulares al robot por medio de comunicación UDP. En el caso de este trabajo sólo se ha tratado con comandos de posición.

El funcionamiento interno de FRI es mucho más sencillo de entender que el del iiwa\_stack: los comandos enviados a la controladora sustituyen a los de la salida del interpolador interno de la misma, por lo que la generación de trayectorias deberá hacerse exteriormente. Sin embargo, sí parece que se lleva a cabo un control de posición interno porque siempre que se le han comandado posiciones y se le ha dejado suficiente tiempo sin volver a enviar otra consigna, ha llegado a la referencia con total precisión. Se intuye

por lo tanto que la posición que se envíe como referencia es la posición que se comanda directamente a los motores del robot, pero que además, si no se vuelve a recibir ninguna otra referencia posteriormente, se lleva a cabo un control en posición articular que asegura que las articulaciones alcanzan la referencia establecida.

La sencillez de la interfaz de control no le quita validez, ya que al contar con un rápido sistema de comunicación por UDP, que permite intervalos de lectura y control de posiciones de hasta 1ms, se convierte en una herramienta válida para llevar a cabo un control en tiempo real. KUKA proporciona además una librería de C++ para simplificar dicha comunicación por UDP, con funciones para leer y enviar mensajes utilizando la interfaz FRI.

Se ha implementado el nodo de ROS `iiwa_command_fri` que funciona como *wrapper* de esta librería. En un primer enfoque, se intentó dividir en dos nodos diferentes: uno encargado de la comunicación, y otro encargado de recibir una trayectoria y generar comandos individuales. Esto se ideó con el objetivo de que el nodo que recibiera trayectorias articulares pudiera emplearse tanto para utilizar el `iiwa_stack` como para utilizar FRI, y solo hubiera dos nodos separados encargados estrictamente de la comunicación con uno u otro. Sin embargo, esta propuesta ha resultado no dar buenos resultados, debido probablemente a que al requerir, para cada punto de la trayectoria, la generación de un mensaje de ROS, su envío a través del *master*, y su recepción por el nodo encargado de la comunicación, se introduce un retraso que afecta a la calidad de la ejecución. La alternativa implementada consiste en juntar esas dos funciones en un sólo nodo, que tiene como entrada una trayectoria completa, y se encarga tanto de dividirla en posiciones articulares individuales como de comandarla por UDP, eliminando los posibles retrasos introducidos por la comunicación entre nodos de ROS. En la figura 4.25 se observa la diferencia del desempeño del robot para una misma trayectoria generada con perfil de velocidades trapezoidal utilizando dos nodos separados o un sólo nodo. Se ve claramente que utilizando dos nodos separados se está introduciendo un retraso en la comunicación que perjudica gravemente el seguimiento de las referencias, mientras que con la alternativa implementada el desempeño del robot es muy bueno.

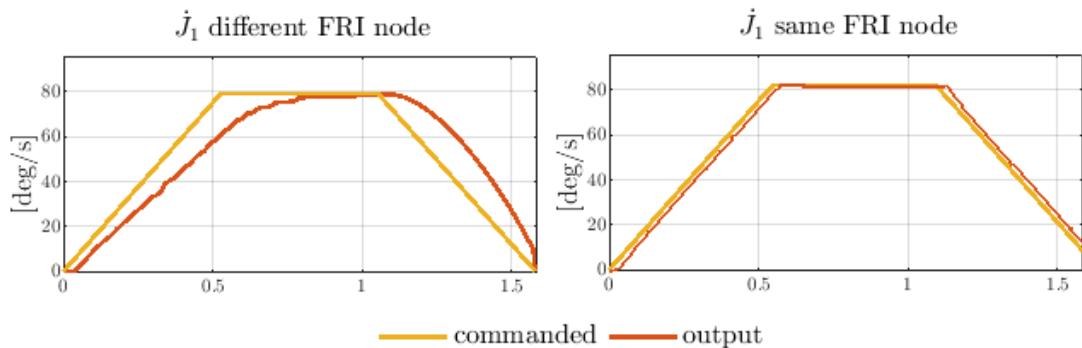


Fig. 4.25. Comparativa del desempeño en velocidad articular de la misma trayectoria comandada desde otro nodo a través de mensajes de ROS (izquierda) y desde el mismo nodo de comunicación con FRI (derecha).

FRI sólo permite comandos de posición, por lo que será el único modo de comandar utilizado. Sin embargo, como en este caso contamos con una comunicación en tiempo real sin los grandes retrasos que habíamos encontrado anteriormente, sí será posible la implementación de bucles de control. Se analizan por lo tanto la reproducción de trayectorias con comandos de posición articular sin control externo, y el desempeño del robot con un control en velocidad cartesiana, realimentación en posición cartesiana y comandos en posición articular.

## Comandar posiciones articulares

La primera estrategia implementada para el seguimiento de trayectorias con FRI ha consistido en el envío de posiciones articulares de referencia consecutivas a través de comunicación UDP. Se ha implementado un *action server* de ROS en la clase `FRIMoveJTrajAction`, instanciada desde el nodo `iiwa_command_stack`, que recibe una trayectoria articular completa y comanda en cada marca de tiempo la configuración articular de referencia a través de comunicación UDP. Devuelve como *feedback* la posición articular en cada instante, y como resultado final, tanto la trayectoria articular comandada como la trayectoria articular ejecutada por el robot.

Es importante dejar claro que aunque se trate del envío de referencias de configuraciones articulares a la controladora, este modo de control no tiene nada que ver con el primero que se analizaba en `iiwa_stack`. En el caso del `iiwa_stack`, se enviaba una posición articular a un programa de la controladora que generaba un polinomio interpolado desde la posición inicial hasta la deseada, y a continuación se iban enviando una a una las posiciones resultado de la generación de la trayectoria al robot. Por otro lado, en el caso de FRI, la posición comandada es directamente la posición que se le envía a las articulaciones del robot.

Los resultados con esta estrategia son también radicalmente diferentes. En primer lugar, la vibración descrita anteriormente desaparece por completo debido a la inexistencia de ese perfil de velocidad trapezoidal no deseado entre puntos que se generaba antes inevitablemente. Además, el error de seguimiento de posición articular baja un orden completo de magnitud, lo que limita también el error de posición cartesiana, que baja desde 1.12 cm que se obtenía con comandos en posición con el `iiwa_stack` a 15 °/s, hasta 0.10 cm con comandos de posición articulares con FRI. El retraso entre trayectorias también es menor, bajando de unos 300ms de media a 30ms; e incluso el error de reproducibilidad baja de 0.3 cm a 0.04 cm.

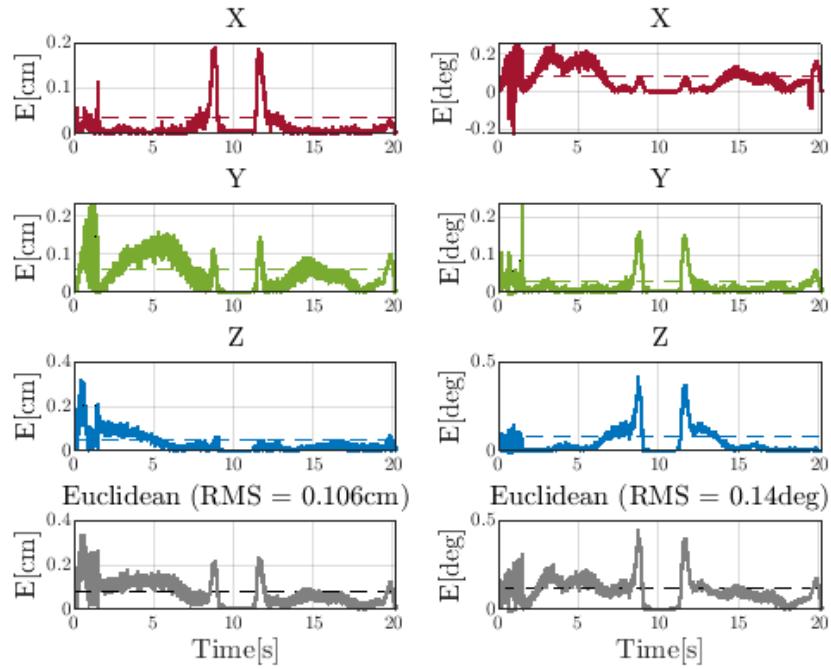


Fig. 4.26. Error entre la trayectoria cartesiana correspondiente a la trayectoria de velocidades articulares comandadas y cada una de las 10 trayectorias ejecutadas a  $15^{\circ}/s$ .

Sin embargo, los resultados obtenidos a velocidades más altas no son tan buenos, aunque no lleguen a ser preocupantemente malos. Con un error de precisión euclídeo de 0.5 cm para  $45^{\circ}/s$ , este modo de control resulta ser una estrategia viable para la reproducción de trayectorias de hasta  $45^{\circ}/s$ . Cuando subimos por encima de este valor hasta  $60^{\circ}/s$ , el error asciende a 1.78 cm, una cifra que ya empieza a ser inaceptable. No obstante, la movilización mediante un brazo robótico a esas velocidades genera otros problemas relacionados con el bienestar del paciente, que podría mostrarse reticente a tener un robot moviéndose a una velocidad tan alta tan cerca de él. Y es que aunque el robot sea colaborativo y no debería haber peligro real para el paciente, se debe buscar no solo su seguridad, sino también su comodidad durante la evaluación. Por este motivo y porque el desempeño del robot empieza a decaer por encima de cierto umbral de velocidad, se ha establecido la velocidad máxima de movilización utilizando el robot colaborativo IIWA a  $45^{\circ}/s$ .

### Comandar posiciones articulares con control cartesiano

Con el propósito de intentar reducir el error de precisión, y ya que FRI permite el control en tiempo real, se ha tratado de implementar un bucle de control externo. Se ha denominado *externo* porque se debe recordar que pese a estar recibiendo referencias por FRI, la controladora del IIWA está realizando un control interno en posición que asegura que se llega a la posición de referencia que se le envíe si no se le vuelve a enviar otra consigna. No se ha encontrado documentación al respecto, pero cuando se comanda a

la articulación 1, por ejemplo, que vaya a la posición 90.01234 °, el robot llega a esa posición, exacta, con todos los decimales, aunque puede tardar unas décimas de segundo en alcanzarla con toda precisión. Esta es una de las razones de que esté funcionando la estrategia de simplemente enviar posiciones una detrás de otra al IIWA, ya que se está relegando el trabajo de control a, precisamente, la controladora del robot.

Sin embargo, a velocidades altas a la controladora no le da tiempo a actuar y se generan errores de seguimiento, como se ha mostrado en las gráficas de la sección anterior, que aunque no sean muy graves, se quieren corregir. Por este motivo, se ha intentado añadir un control externo ejecutado desde el cliente de FRI, el ordenador donde se ejecuta el nodo de ROS `iiwa_command_fri`. Se ha creado un *action server* dentro de la clase `FRIMoveJTrajAction`, similar a los anteriores, pero que en este caso incluye un bucle de control similar al de la Figura 4.27.

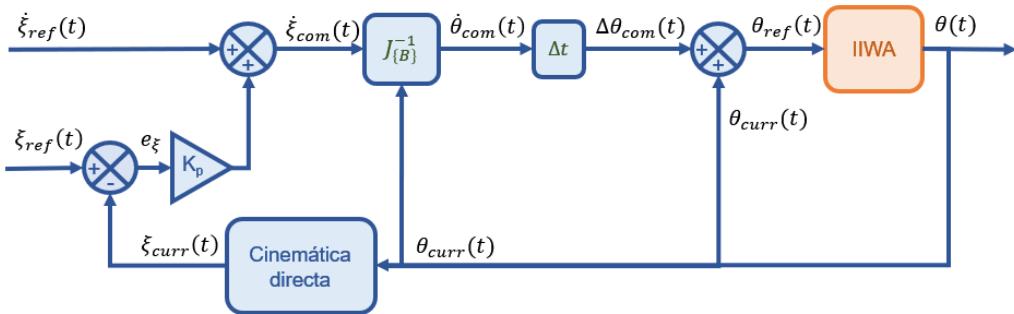


Fig. 4.27. Bucle de control implementado.

Se ha decidido tomar como referencia valores de velocidad cartesiana porque al fin y al cabo, lo que más nos interesa es que la trayectoria de movilización siga un camino lineal en el espacio. De nada nos serviría una trayectoria que pase por todos los puntos cartesianos de referencia pero cuyo recorrido sea zigzagueante. Aun así, no basta con que se sigan las consignas de velocidad cartesiana, ya que el robot no debería alejarse de las posiciones de referencia. Por este motivo, se realimenta la posición cartesiana actual, leída como una posición articular y transformada en una posición cartesiana a través de la CD, y se calcula la diferencia con la posición cartesiana de referencia tomada de la trayectoria deseada en el instante actual. El error calculado se multiplica por una ganancia proporcional  $K_p$ , permitiendo modificar la importancia que se le da al seguimiento de la posición cartesiana frente a la importancia del seguimiento de velocidad cartesiana.

La suma de ambos términos será una velocidad cartesiana, que se debe expresar en el sistema de coordenadas de la base en primer lugar, y después utilizar CDI para transformarlo en una velocidad articular. Para la CDI se ha utilizado la formulación descrita en los apartados 2.2.4 y 2.2.5. Como FRI no permite comandos en velocidad, transformamos esta velocidad  $\dot{\theta}_{com}$  en la próxima posición que se debe comandar multiplicando por el intervalo de control  $\Delta t$ , y sumándole la posición articular actual del robot  $\theta$ . La posición de referencia generada  $\theta_{ref}$  se enviará al IIWA por UDP, y este llevará a cabo un control

interno para acercarse a dicha posición hasta que reciba la siguiente referencia.

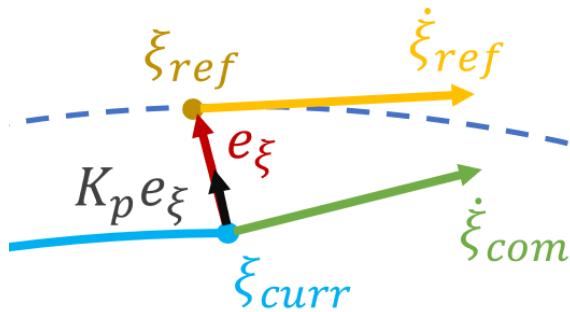


Fig. 4.28. Esquema geométrico de los términos implicados en el bucle de control implementado.

En la Figura 4.28 se muestra un esquema geométrico con todos los términos mencionados e incluidos en el bucle de control de la Figura 4.27. La trayectoria seguida por el robot hasta el instante analizado se ha dibujado en azul claro, mientras que la trayectoria de referencia está dibujada en azul oscuro pespunteado.  $\xi_{curr}$ , en azul claro, es el screw que describe la posición actual de la herramienta, mientras que  $\xi_{ref}$ , en ocre, se refiere a la posición de referencia de la herramienta, es decir, dónde debería estar. El error entre estas dos posiciones se ha denominado  $e_\xi$ , en rojo, y el producto de este vector por la constante  $K_p$  aparece dibujado como el vector  $K_p e_\xi$ , en negro. La referencia de velocidad cartesiana,  $\dot{\xi}_{ref}$ , aparece en amarillo, y la suma de  $K_p e_\xi$  aparece en verde, que será la velocidad cartesiana comandada  $\dot{\xi}_{com}$ .

La implementación de este control en un *action server* de ROS ha requerido además la programación en C++ de una librería con las fórmulas de *Screw Theory* necesitadas, que se ha denominado `IiwaScrewTheory.cpp`, y que contiene una serie de funciones estáticas que utilizan a su vez la librería Eigen3 [71] para cálculos con matrices y vectores.

Los resultados obtenidos con este control a bajas velocidades son los esperados, reduciéndose el error de precisión en posición y velocidad cartesianas a medida que se aumenta el valor de  $K_p$ , tal y como se ve en la Figura 4.29. El error en posición se ha reducido desde 0.877 cm a 0.138 cm para una trayectoria ajustada a 15 °/s, a medida que aumenta el valor de  $K_p$ . El error en velocidad cartesiana también se ha visto reducido con el aumento del valor de  $K_p$ .

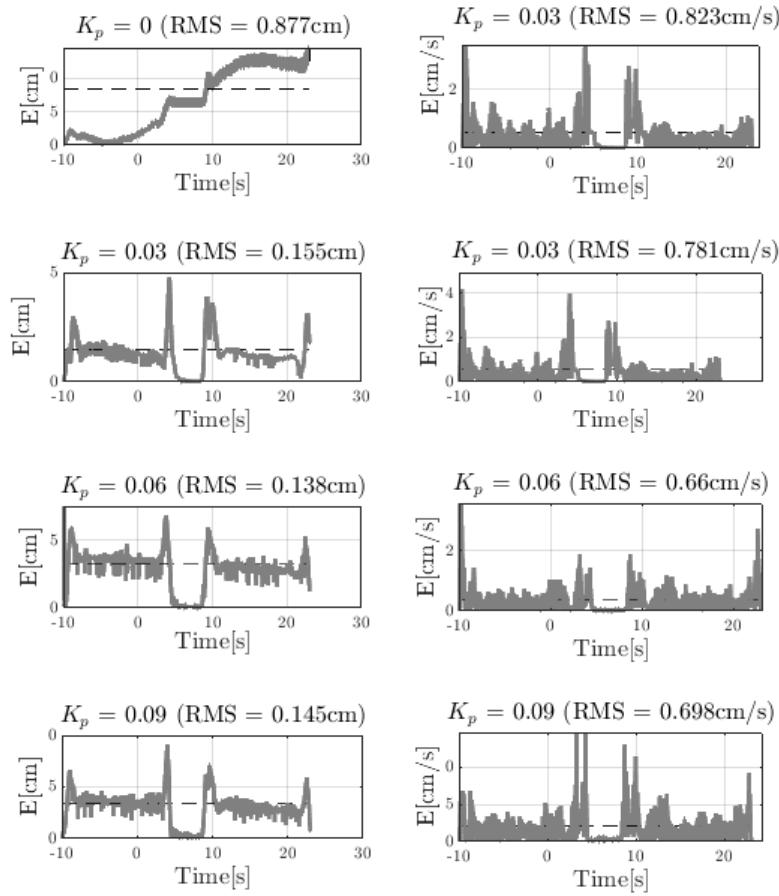


Fig. 4.29. Comparación del error de precisión obtenidos con diferentes valores de  $K_p$  en posición (izquierda) y velocidad (derecha).

Sin embargo, si comparamos estos valores con el error que obteníamos utilizando un control en posición sin realimentación (Figura 4.26), vemos que ha aumentado. Esta diferencia se debe a que en el método anterior la variable de referencia era la posición articular, directamente obtenida desde la posición cartesiana. En este caso la variable de referencia es la velocidad cartesiana, y se obtendrá la posición comandada a partir de la misma. Un pequeño error de seguimiento de la referencia se traduce en un gran error de posición, y será además un error acumulativo. Por este motivo, la gráfica de error para una  $K_p$  de 0 (Figura 4.29, arriba derecha) produce valores incrementales. Partiendo de un valor de error mayor que en el modo anterior, aunque la realimentación en posición cartesiana lo vaya reduciendo significativamente, no se llega a obtener resultados mejores.

A pesar de que a velocidades bajas sí se producen buenos resultados, aunque no mejores que los del modo anterior, a velocidades más altas este control produce resultados peores de los esperados, introduciendo inestabilidades en el sistema que, si no se tiene cuidado, pueden llevar a situaciones peligrosas. En el capítulo 5 se ha realizado una comparación más exhaustiva entre los diferentes modos de control testeados y la influencia de la realimentación en la ejecución.

### 4.3. Captura de datos generados

Durante la movilización del brazo del paciente en el movimiento pasivo de flexo-extensión, se han capturado diferentes datos de los diversos dispositivos que se están utilizando. Se van a dividir estos datos según su fuente en tres grupos: los que se leen del robot, los que proporcionan los sensores Delsys, y los enviados por el sensor fuerzapar. Para cada experimento, se genera una estructura de MATLAB que incluye todos los datos capturados, así como otros datos tomados manualmente, y todas estas estructuras se guardan a su vez en una base de datos que contiene todas las movilizaciones realizadas sobre cada paciente.

#### 4.3.1. Datos proporcionados por el robot

Actualmente, se están capturando dos tipos de trayectorias del robot durante la movilización del paciente.

- Posiciones articulares: Trayectoria formada por  $n_q$  puntos, cada uno de ellos formado por una configuración articular descrita por 7 números reales que muestran valores en radianes, y una marca temporal. Cada trayectoria capturada tiene una longitud diferente, y las marcas temporales capturadas no están siempre equidistantes.
- Pares articulares: Trayectoria formada por  $n_e$  puntos, cada uno de ellos compuesto también por un vector siete-dimensional y una marca temporal, pero esta vez los valores están expresados en N·m por ser pares articulares.

Los datos de la trayectoria articular se están utilizando transformados en poses cartesianas tanto para analizar el desempeño del robot como se ha descrito en la sección anterior, como para introducirlos al modelo biomecánico de OpenSim.

Los datos de pares articulares se utilizan para calcular las fuerzas en el extremo del robot. Para ello, se debe ejecutar una primera trayectoria del robot sin el paciente sujeto a la maneta, en que se midan los pares articulares en vacío del robot exactamente para la trayectoria de movilización que se vaya a realizar a continuación. Se guarda esa trayectoria de pares articulares, y a continuación se repite la trayectoria esta vez ya movilizando al paciente. Las fuerzas en el extremo del robot se calculan restando ambas trayectorias de pares articulares, y multiplicando por el jacobiano de fuerzas utilizando dinámica directa, añadiendo la maneta como un eslabón más de la cadena para hallar las fuerzas en el extremo de la misma.

Todas las trayectorias capturadas se guardan como estructuras de MATLAB de la clase `TiwaTrajectory`, implementada para esta aplicación, que contiene tanto las marcas temporales como los valores de posición, velocidad y aceleración cartesianas o articulares, así como los valores de par capturados y sus respectivas marcas temporales. No todas las trayectorias contienen todos los datos, pero se han implementado funciones que permiten inferir unos datos de la trayectoria desde otros que ya se conozcan.

#### 4.3.2. Datos proporcionados por los sensores Trigno

Los sensores Trigno Delsys [55] proporcionan datos electromiográficos e iniciales del paciente durante el movimiento. Para que los datos leídos sean válidos, es muy importante la colocación de los sensores. Además, para obtener una medida de las señales electromiográficas del paciente en reposo, se aprovecha esa primera trayectoria que se ha mencionado anteriormente que realiza el robot antes de la movilización para capturar los pares articulares sentidos por el robot sin contacto con el paciente. Las señales proporcionadas por cada sensor se dividen en tres categorías:

- Datos electromiográficos: Señal compuesta por  $n_e$  valores de la electromiografía sentida en cada músculo controlado del brazo del paciente.
- Datos del acelerómetro: Señal compuesta por  $n_a$  valores tridimensionales, que describen la aceleración de traslación sentida en el centro del sensor en cada instante.
- Datos del giróscopo: Señal compuesta por  $n_g$  valores tridimensionales, que describen la aceleración de rotación sentida en el centro del sensor en cada instante.

Ninguno de los valores recibidos lleva marcas temporales, pero se conocen de antemano, porque en este caso sí son valores equidistantes en el tiempo. No se recibe el mismo número de valores de electromiografía, acelerómetro y giróscopo necesariamente, ya que van por canales separados. Se envían a través de comunicación TCP/IP. Cómo se produce la comunicación con los sensores, cómo colocarlos sobre el paciente, cómo leer los datos y cómo se han integrado en el sistema Roboespas se describe en detalle en [59].

#### 4.3.3. Datos proporcionados por el sensor fuerza-par

El sensor de fuerza-par ATI SI-660-60 se ha conectado utilizando la tarjeta de lectura DAQ Advantech 9711, cuyos datos se reciben a través de un bloque de Simulink, y posteriormente se pasan al espacio de trabajo de MATLAB como una matriz.

Los datos recibidos son medidas del voltaje sentido en seis resistencias o galgas que tiene el sensor en su interior. Para transformar dichos voltajes en valores de fuerza-par, se debe restar el valor que sientan cuando la fuerza externa es nula, y posteriormente multiplicar por una matriz de calibración  $6 \times 6$  proporcionada por el fabricante, y dará como resultado tres valores de fuerza y tres valores de par en los ejes del sensor, dibujados sobre el mismo. Las medidas de voltaje cuando la fuerza externa es nula cambian según la orientación del robot, por lo que será necesario de nuevo aprovechar esa primera ejecución de la trayectoria en vacío para conservar los valores del voltaje cuando la fuerza externa es nula para cada instante de la trayectoria. En la web del fabricante [56] se pueden encontrar los valores y procedimiento de calibración del sensor.

#### 4.3.4. Otros datos incluidos en la base de datos

Con el fin de etiquetar correctamente la base de datos que se genere con propiedades de los pacientes y los experimentos, además de las capturas realizadas también se guardan otros datos demográficos y biométricos de los pacientes, recogidos manualmente, que se resumen en la Tabla 4.1.

	Altura
<b>Datos generales</b>	Peso
	Sexo
	Mano diestra
	Esternón-hombro
<b>Datos biométricos</b>	Hombro-codo
	Codo-muñeca
	Muñeca-Nudillo puño
	Sensor 1 a la línea del codo interna (brazo estirado)
	Sensor 2 a la línea del codo interna (brazo estirado)
<b>Datos del experimento</b>	Sensor 3 al codo (brazo flexionado)
	Sensor 4 al codo (brazo flexionado)
	Distancia entre sensores 1 y 2
	Distancia entre sensores 3 y 4

Tabla 4.1: Datos tomados manualmente.

Algunos de estos datos se están utilizando en la fase actual del proyecto, en el escalado del modelo, o en la detección de posición de los sensores inerciales; mientras que otros sólo se están guardando por si pudieran ser útiles en el futuro para el entrenamiento del sistema basado en *machine learning* que ayuda a estimar la espasticidad o para una mayor personalización del modelo biomecánico.

Se explica mejor cómo tomar las medidas de las proporciones corporales en base a marcadores naturales del cuerpo en [59]. Para la medida de las distancias entre los sensores hay un trabajo en marcha que trata de leer automáticamente estas distancias a partir de un vídeo o imagen del brazo del paciente con los sensores colocados.

## 5. ANÁLISIS DE RESULTADOS

Con el propósito de evaluar objetivamente los métodos de control utilizados, se ha tratado de cuantificar su desempeño estudiando principalmente tres variables: el retardo de las trayectorias ejecutadas, su error de precisión y repetibilidad. En primer lugar, se han llevado a cabo una serie de experimentos que han tratado de testear a diferentes velocidades los cuatro modos de control del IIWA propuestos.

- Comandos de posición a través de la interfaz iiwa\_stack.
- Comandos de velocidad a través de la interfaz iiwa\_stack.
- Comandos de posición a través de la interfaz FRI.
- Comandos de posición con realimentación a través de la interfaz FRI.

Se han escogido cuatro velocidades: 15, 30, 45 y 60 °/s. Estas velocidades se refieren a la velocidad angular seguida por la herramienta del robot entorno al codo del paciente en la trayectoria de movilización comandada.

Se ha tenido en cuenta que, para la misma trayectoria comandada, el robot no recorre siempre el mismo camino, y que se presentan pequeñas variaciones de una ejecución a otra, por lo que estudiar su error de una única ejecución no sería representativo del desempeño de ese modo de control de manera general. Por este motivo, para cada velocidad y cada modo de control, se han ejecutado 10 trayectorias y se han guardado los datos de la ejecución media y su desviación típica. Además, para el cuarto modo implementado, se han probado diferentes valores de  $K_p$ . En total, se ha repetido una trayectoria 10 veces para cada una de las cuatro velocidades para los primeros tres modos: comandos de posición con el iiwa\_stack, comandos de velocidad con el iiwa\_stack y comandos de posición con FRI; y 10 ejecuciones para 4 valores de  $K_p$  diferentes para cada una de las cuatro velocidades analizadas para el modo de control cartesiano con FRI. En total, se está analizando la ejecución de 280 trayectorias, 70 a cada velocidad. Cada experimento analiza por lo tanto un grupo de 10 ejecuciones con el mismo modo de control, misma velocidad y misma  $K_p$ .

Para cada experimento, se ha tratado de obtener un valor representativo del error de precisión y el error de repetibilidad. El error de precisión se describe en este contexto como la diferencia entre la ejecución del robot y lo que se le ha comandado. Este valor puede referirse a posición articular, velocidad articular, posición cartesiana o velocidad cartesiana. Por otro lado, el error de repetibilidad se refiere a la diferencia entre la ejecución y la media de todas las ejecuciones. Además, se ha analizado el retardo existente desde que se comanda cierto punto hasta que se alcanza.

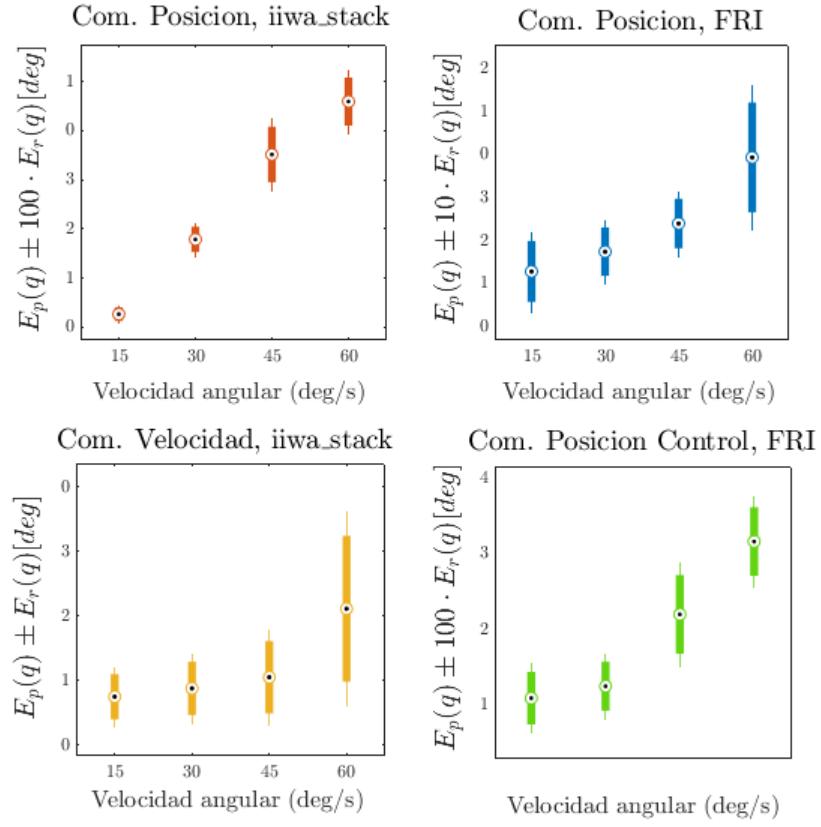


Fig. 5.1. Resumen de errores de precisión y repetibilidad para cada modo de ejecución y cada velocidad.

### 5.1. Retardo entre la trayectoria comandada y la ejecutada

El retardo o retraso entre la trayectoria comandada y la ejecutada se refiere al tiempo que transcurre desde que se comanda una posición o velocidad, y esta es alcanzada por el robot. Para obtener un valor del retardo medio de cada experimento, se ha calculado en primer lugar la trayectoria ejecutada media, que consiste simplemente en hacer un promediado de los valores de posición articular leídos, y a partir de ahí recalcular los valores de velocidad articular, posición cartesiana y velocidad cartesiana.

El retardo se ha calculado sobre las trayectorias de posición articular en los casos de control de posición y en las trayectorias de velocidad articular en las que se comanda en posición, pero se ha comprobado que los valores obtenidos coinciden con un análisis en las trayectorias cartesianas. La estrategia seguida para obtener el valor temporal que sincronizaría las señales ha partido de la suposición de que este no es superior a 1 segundo en ninguna ejecución. Para cada posible valor de retardo de  $t_{retardo} = (0, 0.005, 0.01, \dots, 1)$  se ha calculado el error euclídeo de posición cartesiana entre la trayectoria comandada y la ejecutada si esta se desplazara  $t_{retardo}$  segundos. Con todos los errores obtenidos para cada posible retardo  $t_{retardo}$ , se escoge el retardo cuyo error sea mínimo.

En la Figura 5.2 se muestra una trayectoria articular con un claro retardo presente, a continuación el cálculo del error para cada posible valor de  $t_{retardo}$  y la trayectoria sincronizada una vez se ha obtenido el valor del retardo y suprimido su efecto. Todos los errores calculados posteriormente se calculan sobre las trayectorias sincronizadas.

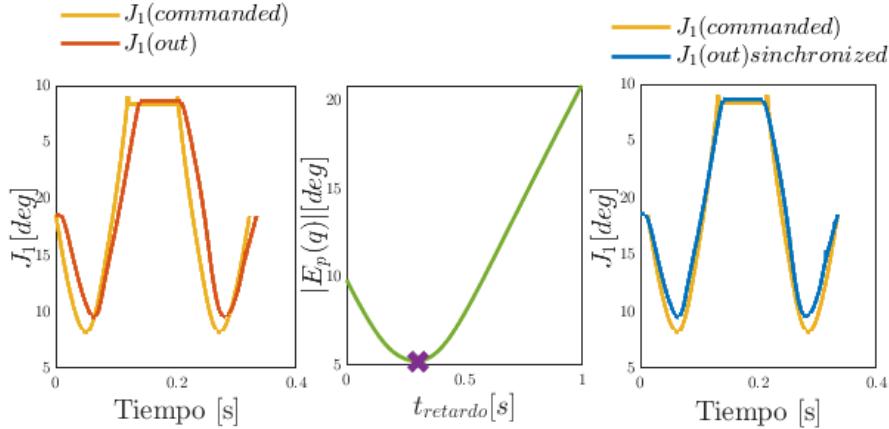


Fig. 5.2. Método de sincronización de la trayectoria comandada y la ejecutada.

Los retardos obtenidos para todos los experimentos por este procedimiento se resumen en la tabla 5.1. En la Figura 5.3 se resumen estos resultados en un diagrama de barras tridimensional según la velocidad de ejecución del experimento y el modo de control utilizado. Para el modo de control por FRI con realimentación de posición cartesiana se han utilizado los valores máximos de retardo de todos los obtenidos para los diferentes valores de  $K_p$ .

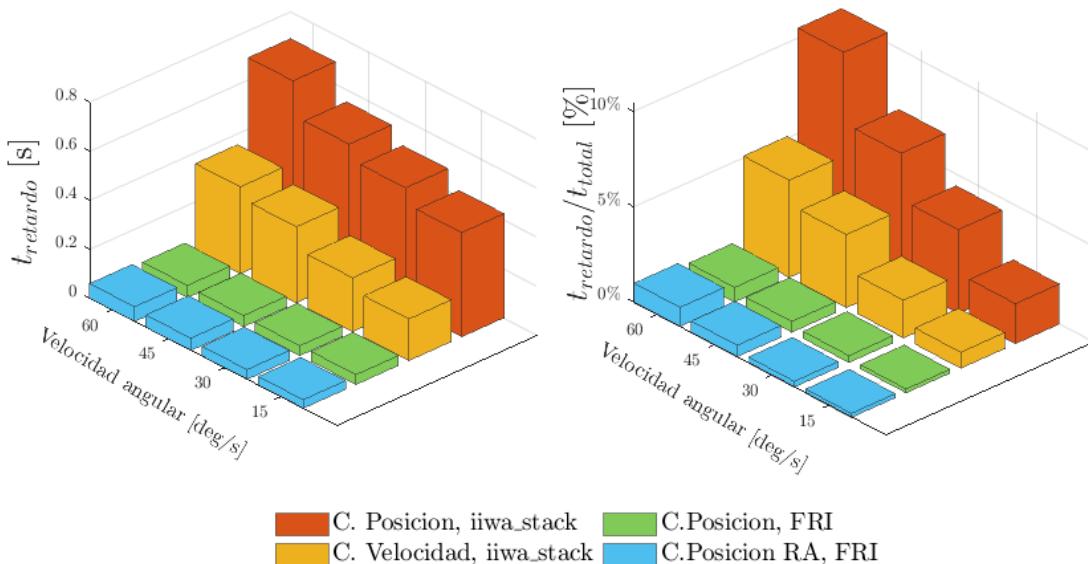


Fig. 5.3. Retardo medio para 10 ejecuciones según el modo de control (izquierda), retardo dividido entre el tiempo total de la trayectoria (derecha).

Comunicación	Control	Velocidad	$K_p$	Retardo [s]	Tiempo total [s]	% Retardo	
iiwa_stack	Posición articular	15		0.4250	20.2051	2.1 %	
		30		0.4900	11.1051	4.4 %	
		45		0.5500	8.0651	6.82 %	
		60		0.6900	6.5551	10.53 %	
	Velocidad articular	15		0.1700	20.2051	0.84 %	
		30		0.2200	11.1051	1.98 %	
		45		0.3100	8.0651	3.84 %	
		60		0.3550	6.5551	5.1 %	
	Posición articular	15		0.0400	20.2051	0.2 %	
		30		0.0400	11.1051	0.36 %	
		45		0.0450	8.0651	0.56 %	
		60		0.05	6.5551	0.76 %	
FRI	Posición articular controlada	0		0.0350	16.4951	0.21 %	
		0.03		0.0350	16.4951	0.21 %	
		15		0.06	0.0350	16.4951	0.21 %
		0.09		0.0350	16.4951	0.21 %	
		0		0.0400	9.2451	0.23 %	
		0.03		0.0300	9.2451	0.25 %	
		30		0.06	0.0400	9.2451	0.3 %
		0.09		0.0450	9.2451	0.42 %	
	Posición articular controlada	0		0.050	6.8251	0.46 %	
		0.003		0.050	6.8251	0.56 %	
		45		0.006	0.050	6.8251	0.64 %
		0.009		0.055	6.8251	0.75 %	
	60	0		-	-	-	
		0.003		0.0600	5.9451	1.01 %	
		0.006		-	-	-	
		0.009		-	-	-	

Tabla 5.1: Resumen de los retardos obtenidos en los experimentos realizados.

Se observa que el retraso es muy superior utilizando la interfaz de comunicación iiwa\_stack a todas las velocidades. Además, aumenta drásticamente con la velocidad de ejecución, algo problemático ya que a mayor velocidad, más importancia cobra el retraso en la trayectoria , porque representa un porcentaje mayor del tiempo total de ejecución.

Al haber más desplazamiento del robot por segundo, un retraso de casi 1 segundo como hay con el control en posición del iiwa\_stack representa un retraso del 10 % del tiempo total de la trayectoria, que inevitablemente, llevará a errores de precisión inadmisibles, como se verá a continuación.

Se ha tratado de descubrir a qué se debe este retraso entre las señales de diferentes maneras. Se ha probado a cambiar a un sistema operativo en tiempo real, y el resultado ha sido similar. También se han eliminado los dispositivos que pudieran estar entorpeciendo la comunicación en la red, como cualquier *router* intermedio, y tampoco ha afectado en absoluto. Se ha probado a incrementar el tiempo de muestreo de la trayectoria para comandar posiciones o velocidades cada más tiempo, por si se estuvieran acumulando mensajes en alguna pila y no estuvieran ejecutándose en las marcas de tiempo establecidas, y tampoco ha habido éxito; de hecho el error incrementa al aumentar el tiempo de muestreo de la trayectoria comandada. También se ha probado a reducir el tiempo de muestreo por debajo de 5 milisegundos, pero la salida no se veía modificada hasta que no se alcanzaban tiempos de muestreo muy bajos, experimentos en los cuales la comunicación directamente saturaba y había que reiniciarla.

En resumen, parece que no hay nada funcionando de manera incorrecta, todos los mensajes que se envían se están leyendo, no se está llenando ninguna pila ni es culpa de un retraso en la comunicación a través de la red. La única explicación posible encontrada es que el interpolador interno del iiwa\_stack tenga una limitación de diseño en la aceleración del robot muy por debajo de su límite real de aceleración. Esto haría que las velocidades comandadas al iiwa\_stack no fueran las velocidades comandadas a los motores realmente, porque el interpolador intermedio las limita antes de enviarlas. Al incrementar la velocidad paulatinamente en lugar de cuando se le indica, tarda más en alcanzar las posiciones y velocidades comandadas, y eso se traduce en un retardo entre la señal comandada y la ejecutada. Sea cual sea el origen del problema, no se ha conseguido evitar esas diferencias temporales tan abismales en el control utilizando el iiwa\_stack.

Respecto a FRI, también tiene un retraso fijo en la señal, pero es mucho más pequeño, del orden de 30 milisegundos. En este caso su dependencia de la velocidad es muy débil, lo cual será positivo a velocidades altas, pero incomprendible para velocidades bajas. ¿Por qué, con un intervalo de control de 5 milisegundos y un seguimiento de referencias excelente, se llega a una posición alcanzable comandada 30 milisegundos después de haber sido comandada? Se está utilizando un sistema operativo en tiempo real, y se está conectando el ordenador que funciona como cliente directamente a la controladora, por lo que no debería haber retrasos en la comunicación. El hecho de que esté presente hasta en ejecuciones con velocidades muy bajas ( $5^{\circ}/s$ ), con el robot casi parado, pero que el retraso apenas incremente cuando se le aumente la velocidad, nos hace pensar que no sea un problema de seguimiento de la posición, sino que simplemente, requiera un tiempo para ponerse en marcha el robot, un tiempo fijo de unos 30 milisegundos, no acumulativo.

Este retardo puede que sea uno de los motivos de que el control en coordenadas cartesianas haya resultado tan insatisfactorio, y en trabajos futuros deberá modelarse una estrategia de control para un sistema discreto con retardo, que parece que es el caso del IIWA utilizando la interfaz de comunicación FRI. Este control podría ser aplicable también al iiwa\_stack, aunque con un retardo tan alto, y dependiente de la velocidad, probablemente los resultados sean muy malos comparado con los obtenibles con FRI, como ha ocurrido hasta ahora.

## 5.2. Error de precisión entre la trayectoria comandada y la ejecutada

Obtener un indicador de la precisión global de un modo de control a partir del gran volumen de datos que se ha obtenido no es algo que tenga solución inmediata. Se cuenta con trayectorias de salida de 10 ejecuciones diferentes, a parte del dato de la trayectoria comandada. De cada trayectoria articular se obtienen en primer lugar sus velocidades articulares, posiciones y velocidades cartesianas correspondientes. A continuación, se calculan los siguientes errores:

- Error de precisión para cada trayectoria: Se calcula la diferencia entre la trayectoria articular ejecutada en cuestión y la trayectoria comandada, por resta en el caso de las posiciones y velocidades articulares, y por resta de *screws* recordando el problema de restar orientaciones (Ecuación 2.1) en el caso de las posiciones y velocidades cartesianas.
- Error de precisión mínimo global: Se toma para cada instante el valor mínimo del error de precisión obtenido de entre todas las trayectorias.
- Error de precisión máximo global: Para cada marca temporal, se toman los valores máximos del error de precisión.
- Error de precisión medio global: Se realiza la media cuadrática de los errores de precisión de cada trayectoria.

Los errores que se expondrán a continuación se referirán al error máximo, ya que nos interesa conocer la peor situación que se puede dar durante la ejecución de trayectoria. De nada serviría un sistema que tuviera un error medio muy bajo, pero una de cada 10 repeticiones pusiera en riesgo la seguridad del paciente. La *trayectoria* del error máximo estará compuesta por tantos puntos como tuviera la trayectoria ejecutada, y cada uno de ellos constará de:

- Un error instantáneo en posición o velocidad articular compuesto por siete variables.
- Un error instantáneo en posición o velocidad cartesiana compuesto por seis variables, tres de posición y tres de orientación.

Como comparar vectores de seis o siete dimensiones resulta muy complejo se ha intentado reducir estos vectores a un único valor representativo. Para el caso de los errores articulares, se ha optado por promediar el vector de error, y utilizarlo como valor de análisis. En el caso de los errores cartesianos, se ha calculado la norma euclídea del error en posición, y se ha dejado sin analizar el error en orientación, ya que en todos los casos era proporcional al de posición. A partir de una señal unidimensional de error, se calculará su valor eficaz y se tomará ese valor como el descriptor del error para ese experimento. En la tabla 5.2 se recogen los errores de precisión calculados por este procedimiento para cada experimento realizado.

Con el objetivo de visualizar qué modo de control utilizado tiene un desempeño correcto a todas las velocidades, se exponen en la Figura 5.4 los errores de precisión obtenidos para todos los experimentos en un histograma acumulativo.

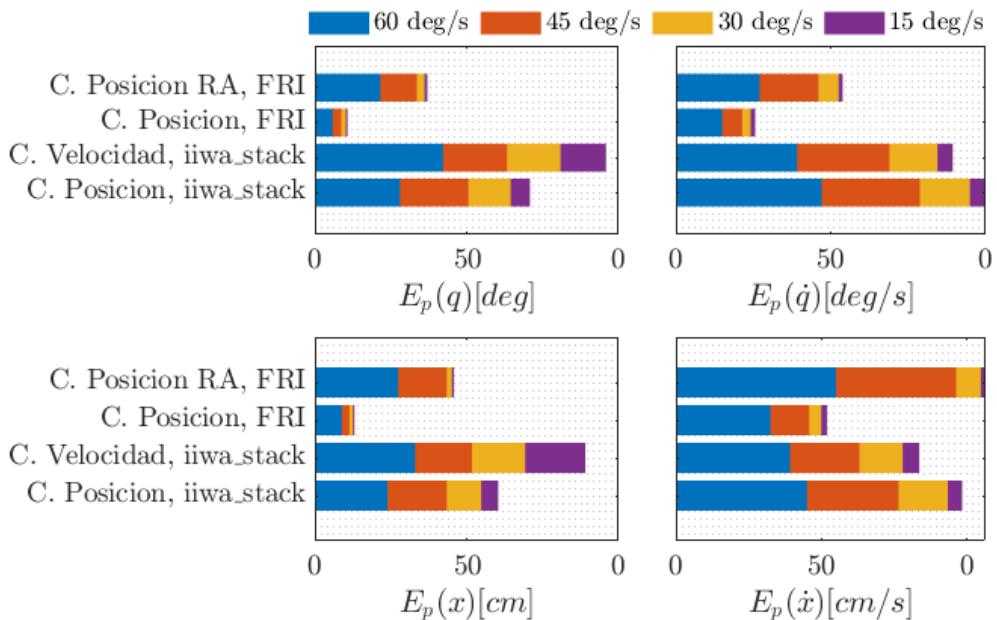


Fig. 5.4. Error de precisión.

En todos los modos de control, el error aumenta con la velocidad del movimiento, y aumenta en la misma proporción. El modo de control ganador para todas las velocidades es claramente el control en posición utilizando FRI sin realimentación, y la diferencia es aún más marcada a velocidades bajas. Para velocidades de menos de 45 °/s, el error es aproximadamente una décima parte que para cualquier otro modo de control. Pero no sólo eso, es que además la suma de los errores de precisión a todas las velocidades representa un tercio de esa misma suma para el control realimentado, y una sexta parte para el control con iiwa\_stack.

Comunicación	Control	Velocidad	$K_p$	$E_p(q)[^{\circ}]$	$E_p(\dot{q})[^{\circ}/s]$	$E_p(x)[cm]$	$E_p(\dot{x})[cm/s]$
iiwa_stack	Posición articular	15	-	0.6330	1.0166	1.120	1.95
		30	-	1.3940	3.2414	2.270	6.78
		45	-	2.2586	6.3514	3.910	12.60
		60	-	2.7986	9.4343	4.780	18
	Velocidad articular	15	-	1.5020	0.993	3.950	2.28
		30	-	1.7593	3.0986	3.510	5.95
		45	-	2.1057	5.9800	3.740	9.53
		60	-	4.2229	7.8429	6.600	15.70
FRI	Posición articular	15	-	0.0575	0.3039	0.106	0.81
		30	-	0.1494	0.5256	0.234	1.66
		45	-	0.2804	1.2951	0.498	5.32
		60	-	0.5876	2.9943	1.780	13.00
	15	0	0.4841	0.2506	0.877	0.71	
		0.03	0.1125	0.2830	0.155	0.78	
		0.06	0.0975	0.2371	0.138	0.66	
		0.09	0.1065	0.2637	0.145	0.60	
	30	0	1.3043	1.1113	2.150	2.63	
		0.03	0.3459	1.3073	0.568	3.97	
		0.06	0.2806	1.1273	0.452	3.40	
		0.09	0.2541	1.0220	0.354	2.50	
	45	0	2.7496	3.4586	4.750	11.00	
		0.003	1.4699	3.5057	3.310	14.30	
		0.006	1.2420	3.8057	3.190	16.50	
		0.009	1.1997	4.0729	3.280	17.80	
	60	0	-	-	-	-	
		0.003	2.1581	5.4186	5.290	22.00	
		0.006	-	-	-	-	
		0.009	-	-	-	-	

Tabla 5.2: Resumen de la precisión obtenida en los experimentos realizados.

El error del iiwa\_stack está muy por encima de los demás, y en concreto, la ejecución utilizando comandos de velocidad. Y es que, aunque esta estrategia produzca trayectorias más suaves, cualquier error de seguimiento de velocidad se refleja en la posición de mane-  
ra incrementada y acumulativa. Si no hubiera ese retardo tan grande entre señales, podría aprovecharse el control en velocidad del iiwa\_stack con una simple realimentación en

posición articular. Sin embargo, este retardo ya tan mencionado imposibilita reducir los errores de precisión del desempeño utilizando iiwa\_stack.

Respecto al control realimentado con FRI, aunque no se esperaba que produjera errores de precisión tan altos, analizándolo un poco sí tiene cierto sentido por varios motivos.

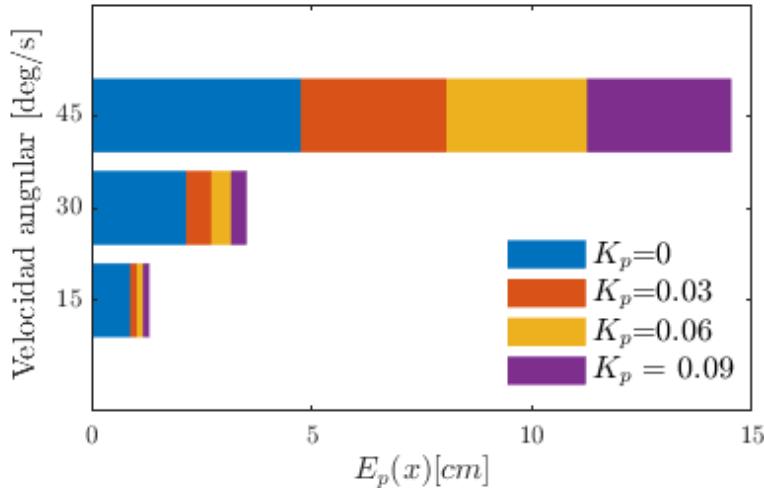


Fig. 5.5. Error de precisión en el modo FRI realimentado.

Lo que más preocupa de estos resultados, visibles numéricamente en la tabla 5.2, es que el error con un valor de  $K_p = 0$  a velocidad baja es aproximadamente 0.7 cm superior al error de FRI sin realimentación(0.877 cm frente a 0.106 cm). En teoría con  $K_p = 0$ , lo que debería estar haciendo el bucle de control únicamente es transformar velocidades cartesianas en posiciones articulares de referencia, utilizando CDI e integrando la velocidad para la configuración actual (ver Figura 4.28). Sin embargo, si comparamos la trayectoria resultante de hacer la CD de la posición comandada con la trayectoria articular correspondiente a la trayectoria de referencia enviada, obtenemos que hay una diferencia de unos 0.7 cm de error de posición euclídeo entre ambas. Esto nos dejaría con un error de unos 0.1 cm entre la consigna que envía el bucle de control al robot y lo que realmente ejecuta para  $K_p = 0$ , el mismo error que cuando no se realiza control, tal y como se esperaba.

El problema radica por tanto en la generación de la referencia de posición articular a partir de una velocidad cartesiana. Parece que se trata de un error de cálculo en la programación del bucle, pero se ha comprobado que la implementación de las fórmulas es correcta. Pero si todas las fórmulas son correctas, ¿por qué el resultado de la posición articular a partir de una velocidad cartesiana calculada dentro del bucle de control produce resultados diferentes que cuando se calcula fuera, sobre la trayectoria teórica ajustada?

Parece que el problema tiene que venir de la utilización del dato de la configuración real del robot en los cálculos, ya que es la única diferencia respecto a los cálculos teóricos. La configuración articular real del robot se está utilizando en dos partes del bucle de control:

- Cálculo del jacobiano geométrico.
- Integración de la velocidad articular deseada en la posición actual para obtener la posición siguiente deseada.

La lectura de la posición articular en cada instante introducirá un error al bucle debido al problema descrito anteriormente respecto a ese pequeño retardo de unos 30-50ms entre que se ejecutan los comandos y el robot alcanza la posición ejecutada. Donde en el cálculo teórico se utilizaba la posición que el robot debería tener en teoría, ahora se utiliza la posición que el robot tiene, que va 50ms retrasada respecto al valor teórico. La posición articular será diferente, el jacobiano será diferente, y la integración de la velocidad articular será diferente. Todas esas modificaciones, alejan la consigna de posición de la teórica anterior, y, sin realimentación, acabarán por alejar la herramienta de la posición deseada. Además, si no se introduce realimentación, se convierte en un error acumulativo. Si se introduce realimentación, se va compensando el error a medida que se sube el valor de  $K_p$ .

No existe una solución clara a este inconveniente sin cambiar el diseño del bucle de control, ya que se haga lo que se haga, ese retardo en la lectura va a estar ahí. La alternativa sería modificar la estrategia de control y añadir algo que lidie con el pequeño retardo en el seguimiento de la posición articular, como por ejemplo, una función que en base a observar las posiciones articulares pasadas, prediga dónde estará el robot dentro del tiempo que dura ese retardo, e introducir esa posición articular al bucle como si fuera la leída actualmente, aunque no sea así realmente. El siguiente paso a seguir para mejorar el seguimiento de trayectorias probablemente sea solucionar el retardo existente en el control de sistemas discretos.

Sin embargo, no todo es negativo en este modo de control, ya que el hecho de que el error vaya bajando a medida que aumenta la  $K_p$  (hasta cierto límite) en todas las velocidades analizadas, significa que realmente la realimentación sí está funcionando, y muy bien. Si consiguiéramos eliminar el *offset* del error en posición articular debido al retardo, una adaptación de este bucle de control podría dar resultados aún mejores que los ya conseguidos sin realimentación. Además, a velocidades altas el sistema se vuelve inestable por no estar realimentando también el error en velocidad. Esto deberá ser otro punto a tener en cuenta en trabajos futuros.

Analizando todos los modos de control en general por su error de precisión, indudablemente nos quedaríamos con el modo de control por envío de posiciones articulares sin realimentación. Aunque no tenga realimentación externa, se recuerda una vez más que la controladora sí está realizando un control en posición articular cuando se le envía una consigna, de ahí los buenos resultados obtenidos sin, aparentemente, cerrar el lazo. En realidad, el lazo ya está cerrado *per se* dentro de la controladora. Con este modo de control, se tiene un error de precisión en posición cartesiana de 0.106 cm a 15 °/s y 0.498 cm a 45 °/s, resultados bastante aceptables para la aplicación.

Aun así, lo que más nos interesa a parte de que el robot siga la trayectoria de movilización correctamente, es que todas las trayectorias ejecutadas se parezcan entre sí. Esto se estudia con el error de repetibilidad, analizado a continuación.

### 5.3. Error de repetibilidad entre las trayectorias ejecutadas y la media de las mismas

Se ha estudiado la diferencia entre las trayectorias ejecutadas en ciertas condiciones, y la media de esas trayectorias. Para ello, se toman los datos de todos los experimentos realizados en diferentes condiciones, consistiendo cada uno de ellos en la reproducción de una misma trayectoria de referencia 10 veces. Para cada experimento, se han calculado los siguientes errores:

- Trayectoria ejecutada media: Se hace la media de las trayectorias en posición articular ejecutadas, y a continuación se completan los valores para la velocidad articular y la posición y velocidad cartesianas.
- Error de repetibilidad para cada trayectoria: Se calcula la diferencia entre la trayectoria articular ejecutada y la trayectoria ejecutada media, por resta en el caso de posiciones, y por resta de *screws* en el caso de las posiciones y velocidades cartesianas, recordando el problema de restar orientaciones 2.1.
- Error de repetibilidad máximo global: Para cada instante, se calcula el máximo de los errores de repetibilidad calculados anteriormente.
- Error de repetibilidad medio global: Se realiza la media de los errores de repetibilidad obtenidos para cada trayectoria independiente.
- Error de repetibilidad mínimo global: Se toma el valor mínimo del error de repetibilidad para cada instante.

De nuevo, para comparar los errores entre sí se toma el valor del error de repetibilidad máximo en cada instante. Al igual que para los errores de posición, se busca transformar los vectores seis y siete-dimensionales de error en un único valor para cada experimento, y se hará calculando la media en los valores articulares, y la norma euclídea en los errores cartesianos. En la Tabla 5.3 se encuentran todos los resultados obtenidos para los experimentos por este procedimiento.

En la Figura 5.6 se han incluido casi todos estos valores divididos según la velocidad a la que se hayan realizado los experimentos y el modo de ejecución. Para el análisis del cuarto modo de control se han utilizado los valores de error para el experimento con la  $K_p$  que produjera menor error de precisión, salvo para el caso a velocidad 60 °/s, donde no se han podido realizar todas las ejecuciones con todos los valores de  $K_p$  propuestos porque a una velocidad tan alta el control realimentado ha producido resultados inestables.

Se ha utilizado una escala logarítmica para los colores de las celdas de la Figura 5.6, ya que la diferencia entre los errores de repetibilidad utilizando FRI y iiwa\_stack y de los mismos entre sí a diferentes velocidades era tan alta que no se visualizaba correctamente con una escala tradicional.

Comunicación	Control	Velocidad	$K_p$	$E_r(q)[^{\circ}]$	$E_r(\dot{q})[^{\circ}/s]$	$E_r(x)[cm]$	$E_r(\dot{x})[cm/s]$
iiwa_stack	Posición articular	15		0.0945	0.1797	0.279	0.63
		30		0.1722	0.4563	0.525	2.27
		45		0.3769	1.0721	1.160	5.55
		60		0.3260	1.0557	1.380	5.61
	Velocidad articular	15		0.9309	0.3277	2.120	1.14
		30		1.1018	0.4630	2.250	1.71
		45		1.4886	0.5887	1.700	2.61
		60		3.007	1.4470	5.610	5.58
FRI	Posición articular	15		0.0189	0.3139	0.046	0.51
		30		0.0450	0.0896	0.013	0.35
		45		0.0154	0.4793	0.050	0.51
		60		0.034	0.1152	0.017	0.60
	Posición articular controlada	15	0	0.0332	0.0796	0.0764	0.25
		0.03	0.0047	0.1619	0.013	0.41	
		0.06	0.0115	0.1091	0.018	0.38	
		0.09	0.0046	0.0849	0.021	0.30	
		30	0	0.0050	0.1051	0.018	0.35
		0.03	0.0214	0.1959	0.049	0.73	
		0.06	0.0174	0.1670	0.0541	0.75	
		0.09	0.0043	0.1253	0.0175	0.48	
	Velocidad articular	15	0	0.0586	0.2636	0.1270	1.05
		0.003	0.0606	0.2363	0.1470	0.93	
		0.006	0.0213	0.6140	0.0480	1.97	
		0.009	0.0069	0.1477	0.0182	0.45	
		30	0	-	-	-	-
		0.003	0.006	0.1116	0.0172	0.35	
		0.006	-	-	-	-	
		0.009	-	-	-	-	

Tabla 5.3: Resumen de la repetibilidad obtenida en los experimentos realizados.

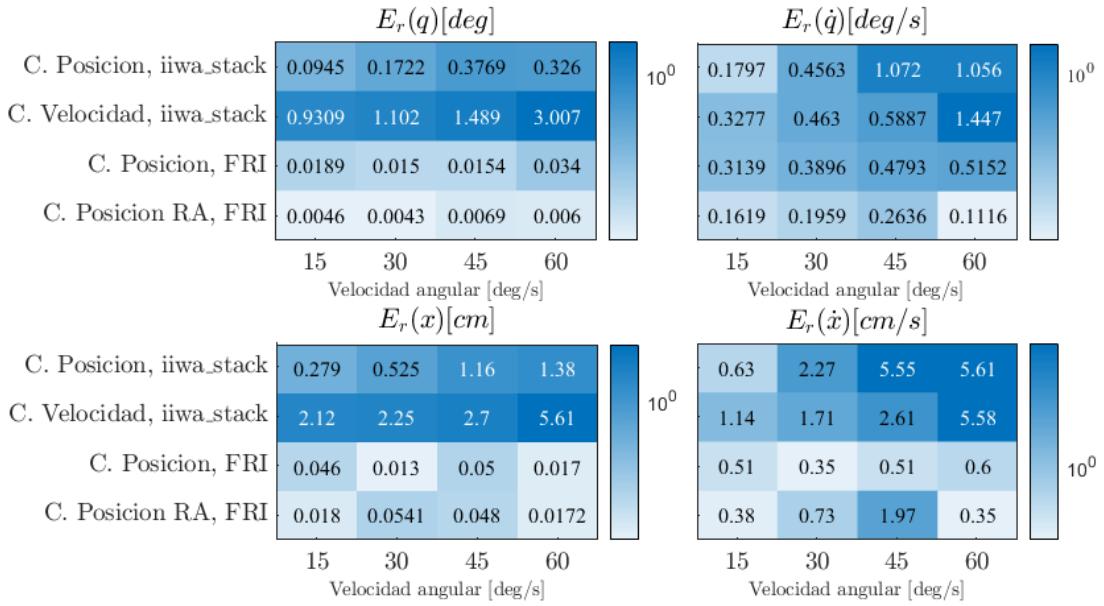


Fig. 5.6. Error de repetibilidad.

Como se observa, la interfaz de comunicación FRI vuelve a dar mejores resultados que los obtenidos con iiwa\_stack. Además, en este caso no aumenta tan drásticamente con la velocidad como aumentaba el error de precisión. Esto es un punto muy positivo ya que aunque el robot a altas velocidades se aleje de la referencia, las trayectorias ejecutadas no difieren mucho entre sí, y se podrán comparar movilizaciones realizadas en diferentes momentos a una misma velocidad.

También se observa que en este caso el modo de control basado en FRI con realimentación en posición no tiene más error de repetibilidad que el modo por comandos de posición sin realimentación generalmente. Esto se debe a que, como se ha explicado antes, el error de precisión del modo realimentado se debe a los cálculos que se realizan para convertir la velocidad cartesiana en posición de referencia, y no realmente a un problema de seguimiento de referencia por el robot.

En general, los valores de repetibilidad son más de 10 veces inferiores a los errores de precisión. A baja velocidad y utilizando FRI con comandos de posición sin realimentación, se obtienen errores de repetibilidad en posición cartesiana de 0.013 cm. Es decir, aunque las trayectorias se alejen 0.1 cm de la consigna (error de precisión), todas ellas van por el mismo camino, aunque ese camino no sea del todo correcto. En concreto cada punto sólo oscila 0.013 cm en el espacio entre una movilización y la siguiente.

Un error de repetibilidad bajo entre movilizaciones era uno de los objetivos del proyecto y se considera que utilizando la interfaz de comunicación FRI sin realimentación se logra con creces.

## **5.4. Comparativa iiwa\_stack y FRI**

Pese a que parece que todos los análisis apuntan a que se debe utilizar FRI como interfaz de comunicación y prescindir por completo de iiwa\_stack, llegar a esa conclusión sería simplificar el problema global, y olvidar que la reproducción de la trayectoria no es el único uso que se le da al robot en este proyecto.

Es cierto que en seguimiento de referencias FRI sale ganando en todos los aspectos: tiene menos retardo, menor error de precisión y menor error de repetibilidad. Entre los dos modos de FRI, el que mejores resultados ha dado ha sido el no realimentado, aunque no por un problema con el uso de la realimentación en sí, por lo que no se debe abandonar esa vía como posible medio para reducir el error de precisión en trabajos futuros.

Sin embargo, no se debe olvidar que hay una gran parte de este trabajo que consiste en la captura y personalización de trayectorias para generar una trayectoria adaptada a las dimensiones y situación del paciente. Este paso se hace actualmente utilizando el modo gravedad compensada proporcionado por iiwa\_stack, que no solo incluye esta característica, sino que también permite la integración con Gazebo y otros paquetes de ROS, proporciona servicios de configuración y está en constante mejora ya que el repositorio sigue activo y sacando nuevas versiones.

Entre estas características la que más útil está resultando es el modo gravedad compensada. Si quisiera prescindir por completo de iiwa\_stack se debería implementar un control similar utilizando FRI. Sería posible hacerlo utilizando el control por pares articulares. Pero hasta que esa funcionalidad no esté implementada, el sistema de captura y personalización de trayectorias seguirá dependiendo irremediablemente del iiwa\_stack, así que no se puede descartar su utilización tan fácilmente.

Respecto al control del robot, el iiwa\_stack también tiene ciertas ventajas. Permite, por ejemplo, comandar posiciones alejadas de la actual sin necesidad de generar una trayectoria interpolada manualmente. Además, aunque el error en precisión es verdad que no sea el mejor para ejecutar trayectorias de puntos, cuando se mandan referencias únicas de posición se alcanzan con total precisión. Para una aplicación en la que no se requiera seguir trayectorias, sino que baste con alcanzar posiciones concretas, iiwa\_stack sería la mejor opción por tener más funcionalidades que FRI y ser fácilmente integrable.

Se resumen en la Tabla 5.4 las ventajas e inconvenientes de ambas interfaces de comunicación con el robot colaborativo IIWA.

	FRI	iiwa_stack
Precisión a velocidades bajas	Excelente	Media
Precisión a velocidades altas	Buena	Mala
Repetibilidad	Excelente	Media
Retardo de seguimiento de referencias	Bajo	Muy alto
Facilidad de integración	Media	Excelente
Herramientas proporcionadas	Ninguna	Modo gravedad compensada, Integración con Gazebo, ...
Control en posición	Sí	Sí
Control en velocidad	No	Sí
Control en par	Sí	No
Documentación encontrada	Poca	Suficiente

Tabla 5.4: Comparativa de características de FRI y iiwa\_stack.

## 6. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo fin de máster se ha presentado una alternativa de evaluación objetiva de espasticidad basada en la movilización asistida por el robot colaborativo IIWA. El sistema se complementa a través de la utilización de sensores electromiográficos y el uso de un modelo biomecánico personalizado para estimar el descriptor espástico. Dentro de este proyecto, este trabajo se ha centrado en el desarrollo del bloque de captura, ajuste y reproducción de trayectorias personalizadas para la movilización del brazo del paciente en movimientos de flexo-extensión.

Se considera que se han alcanzado los objetivos de las características deseadas de la trayectoria ajustada y la reproducción de la misma:

- La trayectoria es personalizable, ya que se captura directamente del movimiento natural del paciente utilizando el modo gravedad compensada restringida en coordenadas cartesianas, incluida en iiwa\_stack. De esta manera, se estiman sus medidas de longitud de antebrazo, la posición de su codo durante el experimento, y su rango de movimiento.
- Mediante el ajuste de la trayectoria, se asegura la continuidad y suavidad del movimiento, así como unas características de simetría de la trayectoria articular de flexo-extensión respecto al punto más alejado de la posición inicial. Esto asegura que el movimiento es realizable, sin interrupciones, y adaptado al paciente.
- Se ha conseguido un método de comandar al robot que asegura unos valores máximos de error de reproducibilidad de  $\sim 0.1\text{cm}$  a velocidades bajas ( $15^\circ/\text{s}$ ) y  $\sim 0.5\text{cm}$  para velocidades más altas ( $45^\circ/\text{s}$ ).
- Las trayectorias reproducidas utilizando este método son también repetibles, con una diferencia máxima de  $0.05\text{cm}$  entre repeticiones consecutivas.
- Además, se han integrado las diferentes partes del proyecto utilizando una arquitectura MATLAB-ROS, y se ha desarrollado una interfaz que permite la interacción con las distintas tecnologías (sensores EMG-IMU, sensor F/T, OpenSim, ROS, ...) utilizadas en el proyecto.

En general, se considera que se han alcanzado todos los objetivos muy satisfactoriamente. El único objetivo que podría haber dado resultados no tan positivos es el de obtener un bajo error de reproducibilidad en la ejecución de las trayectorias, que pese a que los resultados son aceptables, no son perfectos a velocidades altas. La utilización de la interfaz de comunicación FRI ha resultado ser un acierto a la hora de ejecutar trayectorias en el IIWA.

Sin embargo, hay ciertas líneas que siguen abiertas a mejora en este sistema, y que perfeccionarían el desempeño de la ejecución del robot, y el sistema Roboespas en general. Dado que FRI ha demostrado ser mejor que iiwa\_stack en la reproducción de trayectorias, y para no requerir cambiar entre interfaces de comunicación entre la captura y la reproducción de trayectorias, como trabajo futuro se propone implementar un modo de control por compensación de gravedad utilizando FRI, e integrarlo en el proyecto. Se buscará que tenga las mismas características que el disponible en iiwa\_stack, como la posibilidad de restringir el movimiento en ciertos ejes del sistema de coordenadas cartesiano.

Además, si se consiguiera prescindir por completo de iiwa\_stack, no sería necesario la utilización del sistema operativo Ubuntu, y se podría migrar el sistema de comunicación con FRI a Windows-C++. Esto simplificaría la arquitectura del software, y también el hardware necesario.

Respecto a la reproducción de trayectorias, el bucle de control implementado sobre FRI no ha dado los resultados esperados, debido principalmente a un retardo de ~50 ms en el seguimiento de referencias por parte del robot. Aun así, el uso del control realimentado sí parece haber resultado en la disminución del error. Probablemente se conseguirían mejores resultados de reproducibilidad utilizando una estrategia de control realimentado que tuviera en cuenta en su diseño el retardo presente en el seguimiento de referencias.

A través de este trabajo se ha presentado la captura y reproducción de trayectorias para la movilización asistida de extremidad superior, integrada en un sistema de evaluación objetivo de espasticidad, Roboespas. Este proyecto es un claro ejemplo de la utilización de sistemas instrumentados para la mejora de los diagnósticos clínicos, incrementando la precisión y sistematizando el proceso.

## Contribuciones publicadas

- E. D. Oña, A. Casanova, A. Gordillo, C. Balaguer, A. Jardón. *Towards Objective Assessment of Upper Limb Spasticity by Means of Collaborative Robots*, Biosystems & Biorobotics, Springer, 2020.
- A. Casanova, E.D. Oña, A. Gordillo, A. Jardón. *Robot-based Strategy for Objective Evaluation of Upper Limb Spasticity*, Robótica e Inteligencia Artificial: Retos y Nuevas Oportunidades, 2019.

## A. PUESTA EN MARCHA DEL PAQUETE IIWA\_STACK

El IIWA habitualmente se programa utilizando el software KUKA Sunrise Workbench y sus aplicaciones de Java, incluidas siempre dentro de un Proyecto Sunrise. El paquete iiwa\_stack cuenta con una serie de clases programadas en Java que se llaman desde una API de Sunrise llamada ROSSmartServo, y que utilizan el puerto KONI del robot (uno de los dos puertos Ethernet que tiene, con una pegatina que pone KONI justo encima) para enviar y recibir mensajes a través de ROS. El roscore se ejecutará en el cliente, y la API de Java de iiwa\_stack se conectará a ese roscore, tal y como se muestra en la Figura E.2.

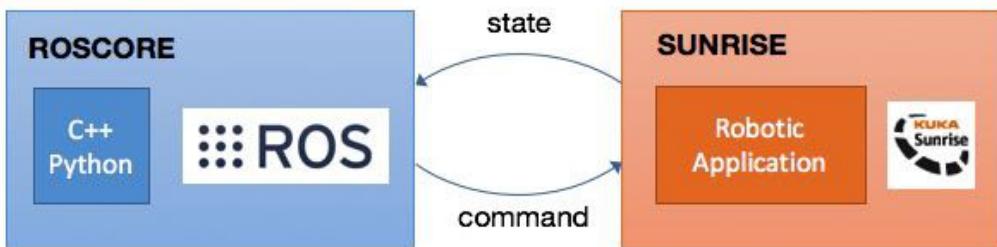


Fig. A.1. Interacción ROS-Sunrise iiwa\_stack iiwa\_stack.

Sin embargo, para que eso funcione, antes habrá que llevar a cabo una serie de tareas de configuración, tanto en la controladora como en el cliente. En primer lugar, se configurará la controladora para permitir enviar y recibir mensajes a través del puerto KONI.

1. Arrancar la controladora, y conectar un ratón, teclado y pantalla. Introducir usuario y contraseña de KUKA y acceder al sistema operativo Windows Server que corre por debajo de Sunrise OS.
2. Apagar KRC. Abajo a la derecha, en la barra de tareas, clic derecho en el icono de KRC y clic en *Shutdown*.
3. Abrir una terminal y escribir

```
C:\KUKA\Hardware\Manager\KUKAHardwareManager.exe -assign  
OptionNIC -os WIN.
```

Esto asignará el puerto KONI a Windows y lo desasignará al sistema en tiempo real de FRI.
4. Reiniciar la controladora.

5. Establecer la IP del robot en su red KONI. Ir al Centro de Redes y Configuración de Windows y cambiar la IP de la red cuyas IPs no estén establecidas (la otra parece que es la red con los motores del robot así que no conviene cambiar la IP) por **160.69.69.69**. Dejar lo demás como estaba.

A continuación, se realizarán unas tareas de configuración en el cliente, el ordenador con Ubuntu que vayamos a utilizar para controlar el robot:

1. En primer lugar, se debe tener instalado ROS Kinetic o ROS Indigo [72].
2. Se debe descargar la versión 1.2.5 del paquete iiwa\_stack, que es la que se está utilizando, en el workspace de ROS que se esté empleando.

```
wget https://github.com/IFL-CAMP/iiwa_stack/archive/1.2.5.tar.gz
```

3. Se deben instalar todas las dependencias del paquete.

```
rosdep install --from-paths-src --ignore-src -r -y
```

4. A continuación, ya podemos construir el paquete.

```
catkin_make
```

Si da cualquier problema, compilar primero **iiwa\_msgs**, ya que a veces requiere que se compile antes que los demás.

```
catkin_make --pkg iiwa_msgs
```

5. No podemos olvidarnos de referenciar los archivos de origen y establecer la IP de la conexión, añadiendo el siguiente comando al `~/.bashrc` o ejecutándolo siempre que vayamos a usar el iiwa\_stack.

```
source devel/setup.bash
```

```
export ROS_IP=160.69.69.100
```

```
ROS_MASTER_URI = http://$ROS_IP:11311
```

6. Establecer una red con la controladora mediante la conexión de un cable Ethernet desde el cliente hasta el puerto KONI.

7. Crear una red cableada nueva manualmente desde el centro de redes y establecer la IP fija a **160.69.69.100**. Conectarse a esa red.

8. Abrir una terminal en el cliente de Ubuntu y hacer ping al puerto KONI del robot.

```
ping 160.69.69.69
```

Si se obtiene respuesta, ya está todo listo para instalar el paquete.

A continuación, se debe configurar el proyecto de Sunrise para ejecutar la aplicación de iiwa\_stack.

1. Abrir Sunrise Workbench en un ordenador conectado con el IIWA a través del puerto X66 (el que no es el puerto KONI).
2. Crear un proyecto de Sunrise nuevo o cargar el que hay de la controladora.
3. Pegar los archivos de la carpeta `iiwa_java` del repositorio que hemos descargado en el proyecto.
4. Añadir las librerías al path del proyecto haciendo clic derecho en el proyecto y ejecutando “Editar vía de construcción” o *Edit Build Path* según se tenga el programa en inglés o en español.
5. Volcar el proyecto sobre el IIWA.
6. Habrán aparecido algunas aplicaciones nuevas en el SmartPad del IIWA, pero la que vamos a utilizar será ROSSmartServo.

Para probar que está todo bien instalado, se va a probar a leer las posiciones del robot.

1. Lanzar la aplicación ROSSmart Servo en la controladora.
2. Ejecutar un `roscore` o uno de los launchers del paquete `iiwa_gazebo` de `iiwa_stack`.
3. Comprobar que suenan los frenos del robot activándose. Si es así, probar a leer la posición del robot mediante: `rostopic echo /iiwa/state/JointPosition`.

Y de esta manera, hemos conseguido conectarnos con el robot y obtener información de su configuración actual utilizando el paquete `iiwa_stack`. Para más ejemplos, sincronizar `iiwa_stack` con Gazebo, añadir una herramienta al modelo de Gazebo o comprender mejor las posibles configuraciones del `iiwa_stack`, visitar la wiki de `iiwa_stack` [73].

## B. PUESTA EN MARCHA DE FAST ROBOT INTERFACE

Fast Robot Interface es una interfaz de comunicación proporcionada por el propio fabricante, KUKA. Los archivos necesarios para su utilización deben ser provistos por el fabricante, o pedidos a él si no se tienen.

Para poder utilizar FRI, en primer lugar debemos asegurarnos de que el puerto KONI está asignado al sistema operativo en tiempo real (RTOS).

1. Arrancar la controladora, y conectar un ratón, teclado y pantalla. Introducir usuario y contraseña de KUKA y acceder al sistema operativo Windows Server que corre por debajo de Sunrise OS.
2. Apagar KRC. Abajo a la derecha, en la barra de tareas, clic derecho en el icono KRC y clic en *Shutdown*.
3. Abrir una terminal y escribir:

```
C:\KUKA\Hardware\Manager\KUKAHardwareManager.exe -assign  
OptionNIC -os RTOS.
```

Esto asignará el puerto KONI a la interfaz FRI y lo desasignará de Windows, por lo que la comunicación con iiwa\_stack dejará de ser posible.

4. Reiniciar la controladora.

Además, se debe configurar el proyecto de Sunrise correctamente para poder utilizar FRI, de la siguiente manera:

1. En un Proyecto de Sunrise, ir a *Ayuda >Install New Software >Añadir >Archive*. Se debe seleccionar la carpeta con los .zips de instalación de FRI, llamada habitualmente *Sunrise.OS.1.16.0\_B7\_C480415* y proporcionada por KUKA. A continuación, seleccionar KUKA Connectivity, e instalarlo. Si ya está instalado anteriormente, dará un error informando de que la instalación será ignorada, pero no debemos preocuparnos porque significa que ya lo tenemos instalado.
2. Personalizar el archivo SafetyConfiguration.sconf de nuestro Proyecto, asegurándonos de que las tres primeras filas del archivo están desmarcadas.
3. En el archivo StationSetup.cat, marcar las dos opciones disponibles para FRI: Fast Robot Interface Extension y Fast Robot Interface Extension Example Application.
4. Al marcar estas opciones, aparecerán dos cosas nuevas en el proyecto: las aplicaciones de ejemplo, que aparecerán en *SunriseProject >examples >com >kuka*

>connectivity >fri >example; y las clases de C++ que proporciona KUKA para interactuar con la interfaz de FRI a través de UDP. Estas aparecerán en un archivo .zip denominado *FRI – Client – SDK\_Cpp.zip*. Se deben copiar y pegar desde el ordenador de Windows donde estamos ejecutando Sunrise Workbench para personalizar el proyecto hasta el ordenador con Ubuntu que vaya a funcionar como cliente.

La IP elegida en este caso será la 192.170.10.2, por ser la que utilizan en los manuales de FRI. Cuando se conecten cliente y controladora con un cable a través del puerto KONI, se debe crear una red cableada nueva manualmente cuya IP fija debe ser 192.170.10.2, y debemos conectarnos a esa red. En Sunrise Workbench, todos los archivos de ejemplo que habían aparecido anteriormente, se deben abrir y cambiar la IP por defecto introducida a la IP escogida, si no tiene ya el mismo valor.

Por último, queda compilar los ejemplos de FRI, que se hará siguiendo el siguiente procedimiento.

```
cd FRI-Client-SDK_Cpp/  
mkdir build && cd build  
cmake ..  
make -j$(nproc)  
make -j$(nproc) examples  
sudo make install
```

Es posible que de un error de configuración porque los archivos de KUKA vienen con un carácter \ sobrante. Si es así, quitar la \ de donde señala el error, y continuar con la compilación.

Una vez se haya compilado, ya podremos probar los ejemplos de FRI y leer el estado del robot ejecutando, por ejemplo:

```
cd \example\IOAccess  
../IOAccess "192.170.10.2 30200"
```

Si ha funcionado correctamente, debería aparecer en la terminal de Ubuntu del cliente:

```
IOAccess Client initialized: Enter IOAccess Client Application...
```

y en el SmartPad:

```
Creating FRI connection from controller port 30200 to  
192.170.10.200:30200 SendPeriod: 5ms ReceiveMultiplier: ...
```

## C. ESQUEMA DEL PAQUETE DE ROS DESARROLLADO

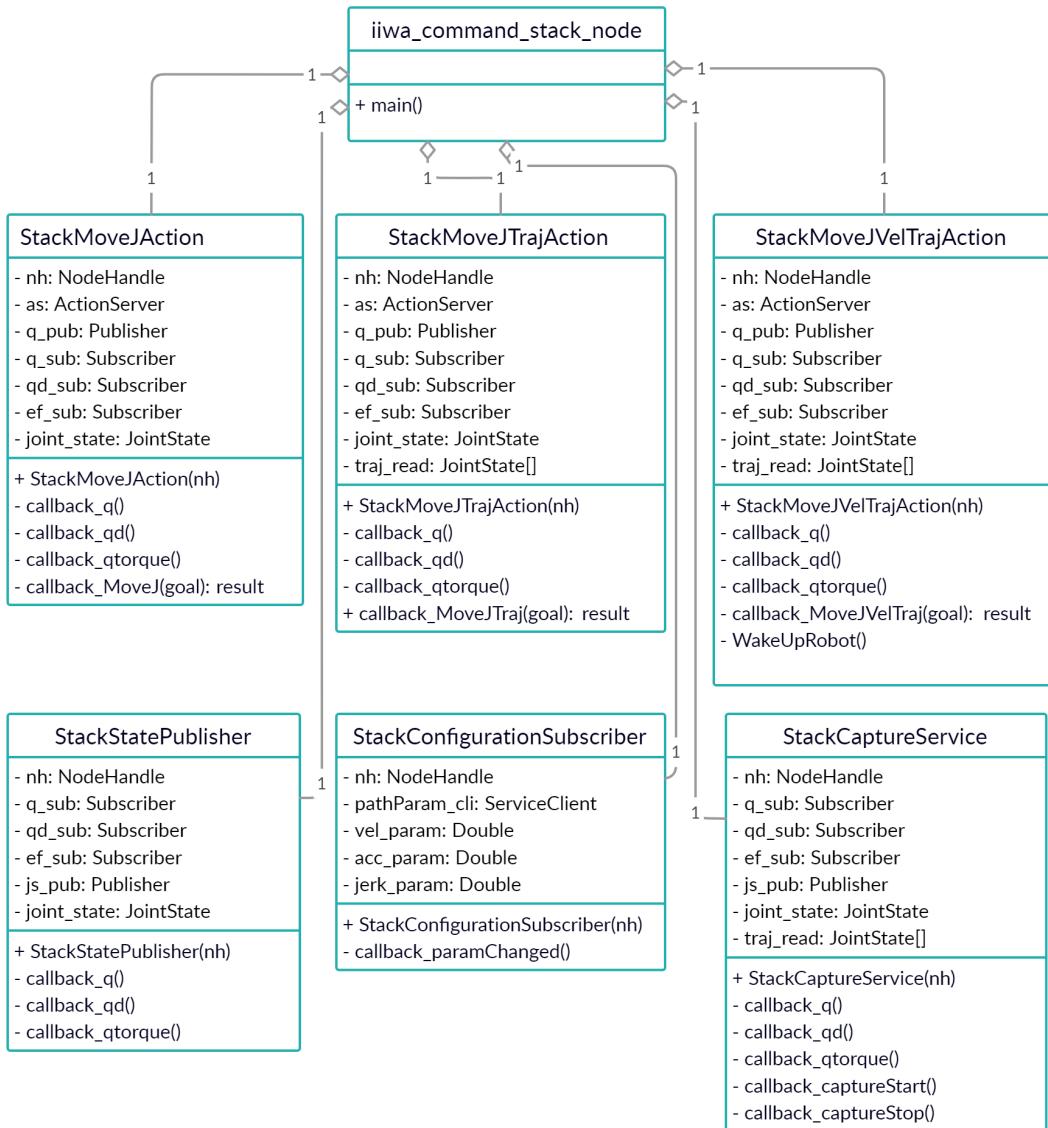


Fig. C.1. Esquema UML de `iiwa_command_stack_node`.

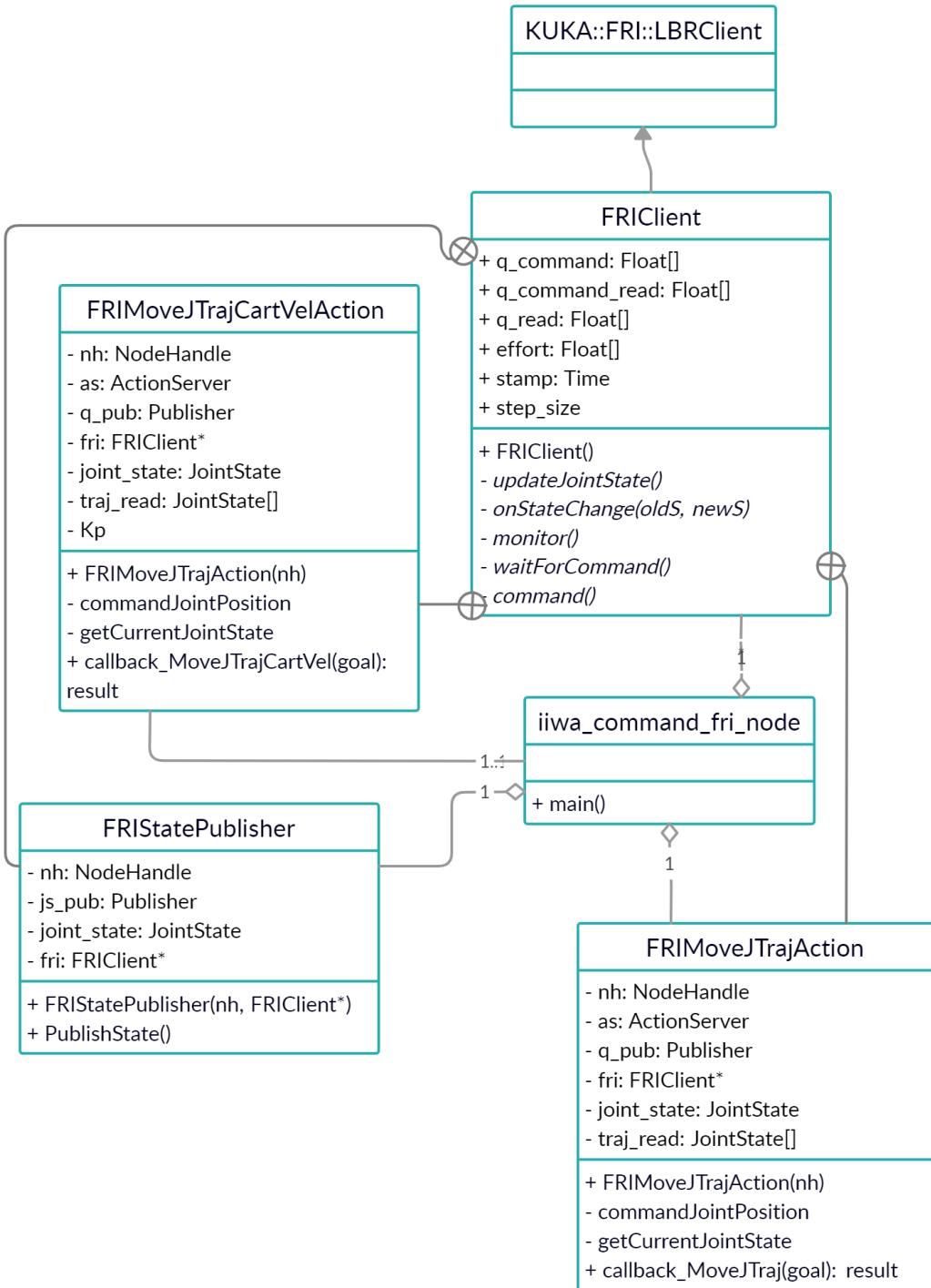


Fig. C.2. Esquema UML de `iiwa_command_fri_node`.

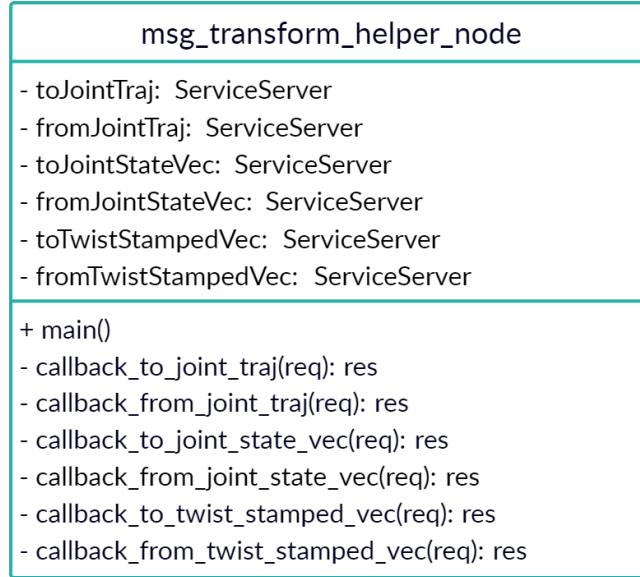


Fig. C.3. Esquema UML de IiwaMsgTransformer.

## D. ESQUEMA UML DEL PAQUETE DE MATLAB DESARROLLADO

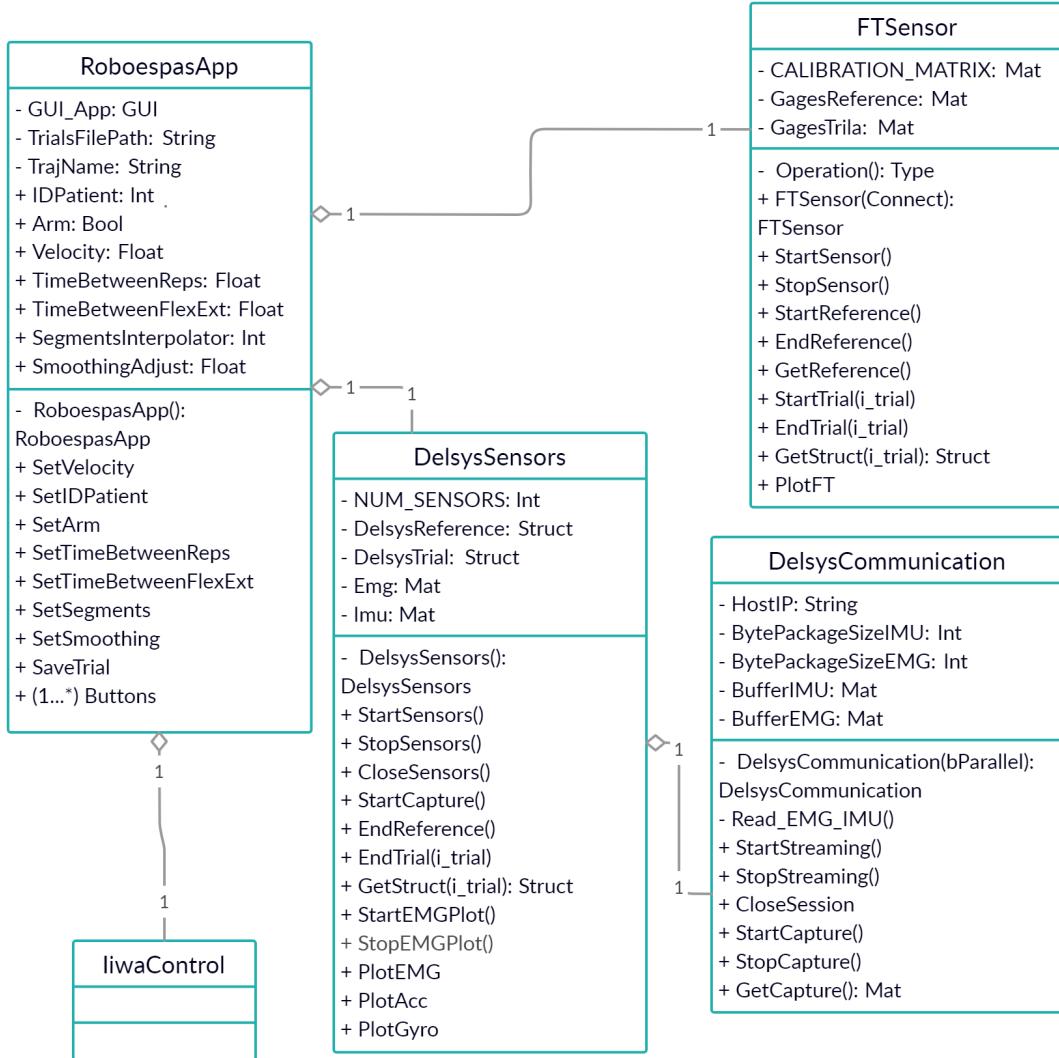


Fig. D.1. Esquema UML de la interfaz gráfica y las tres clases principales: FTsensor, DelsysSensors y IiwaControl.

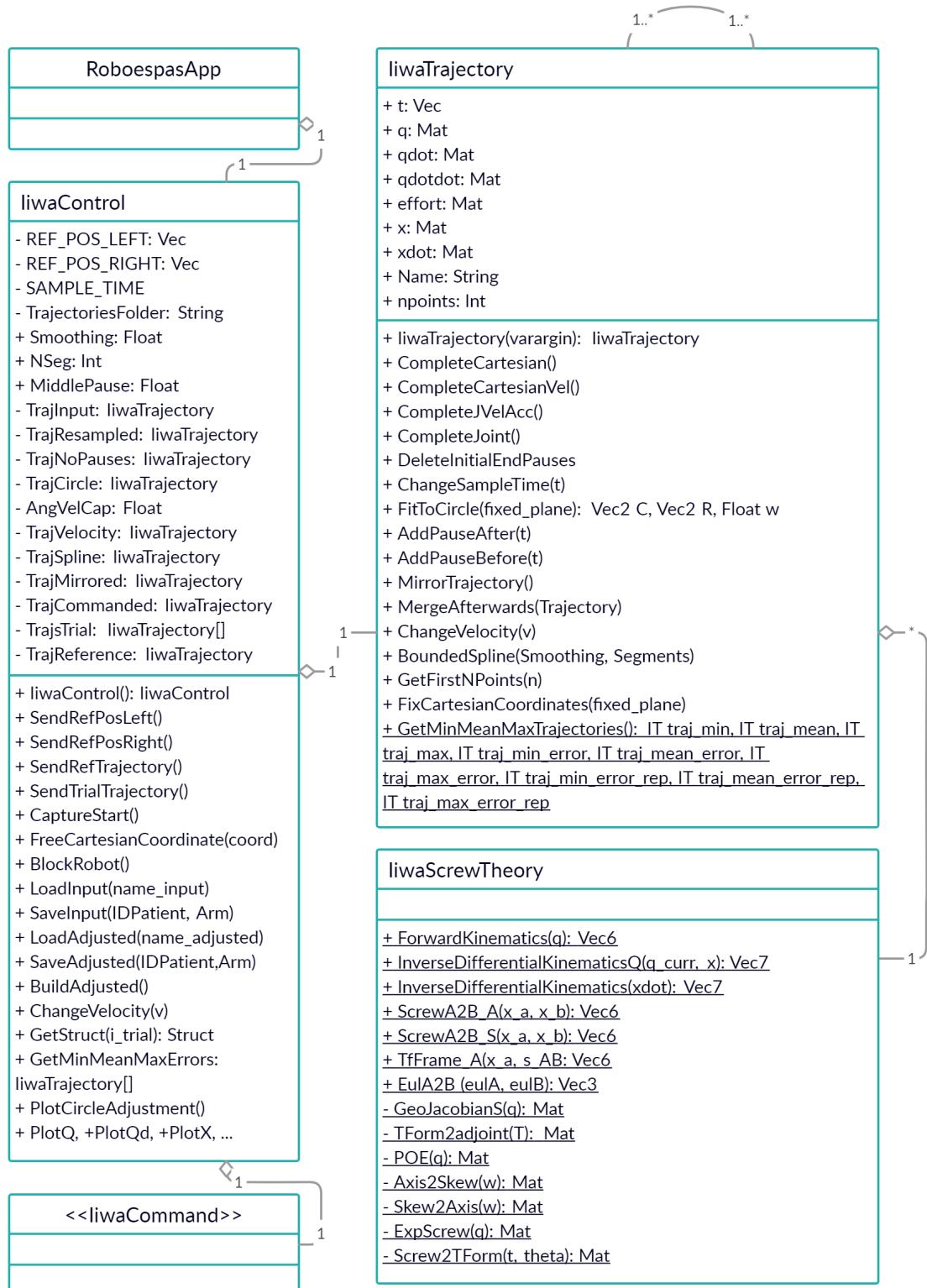


Fig. D.2. Esquema UML de las clase liwaControl, liwaTrajectory y liwaScrewTheory.

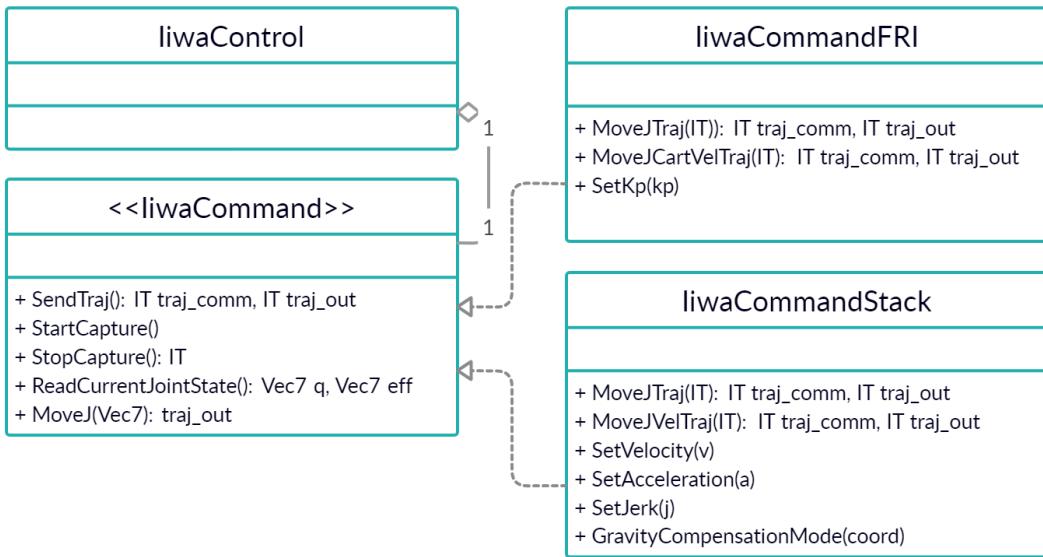


Fig. D.3. Esquema UML de la interfaz LiwaCommand y sus dos implementaciones: LiwaCommandFRI y LiwaCommandStack.

## E. ESQUEMA DE LA INTERACCIÓN ENTRE MATLAB Y ROS

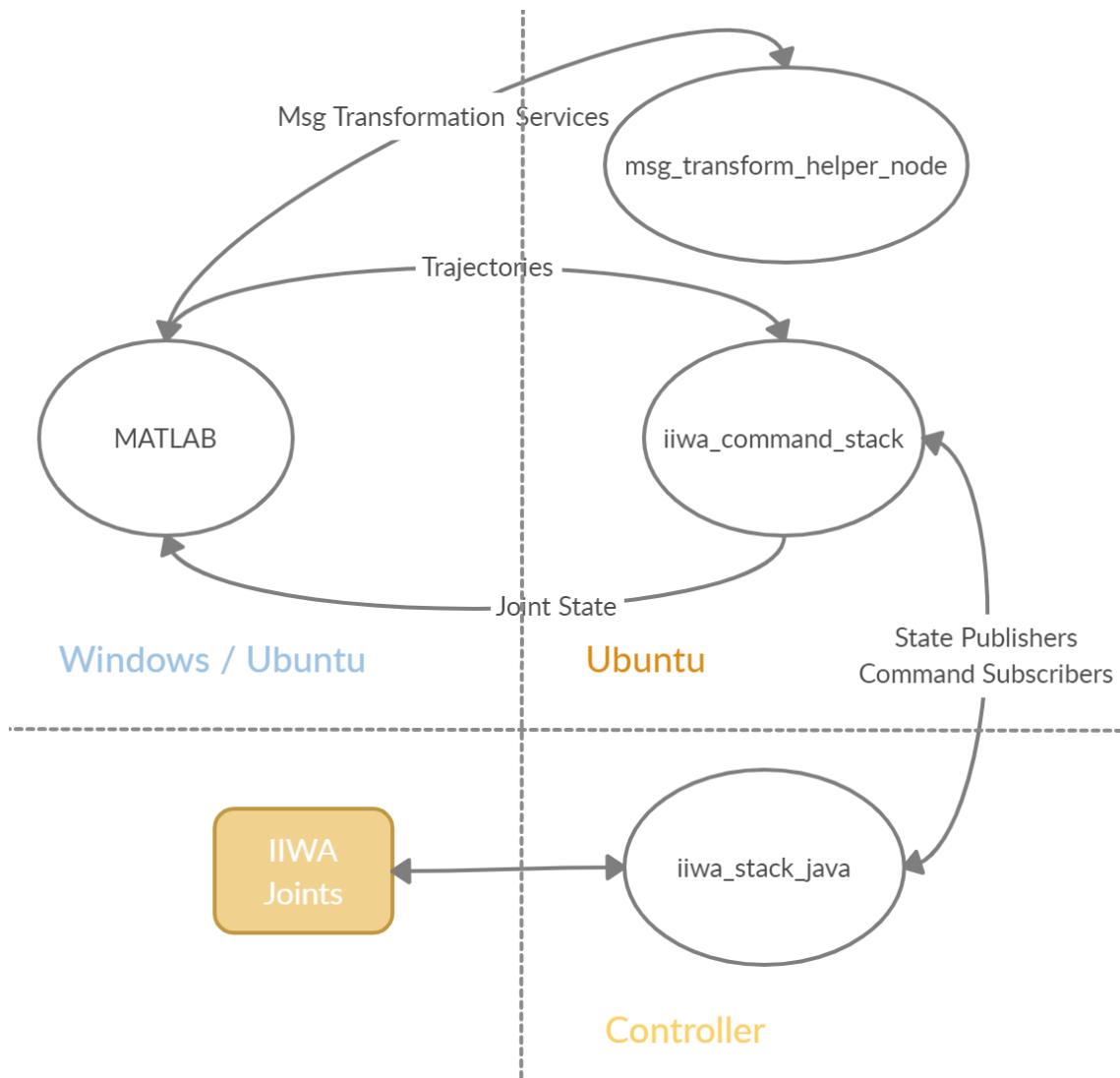


Fig. E.1. Esquema de interacción entre MATLAB y ROS con iiwa\_stack.

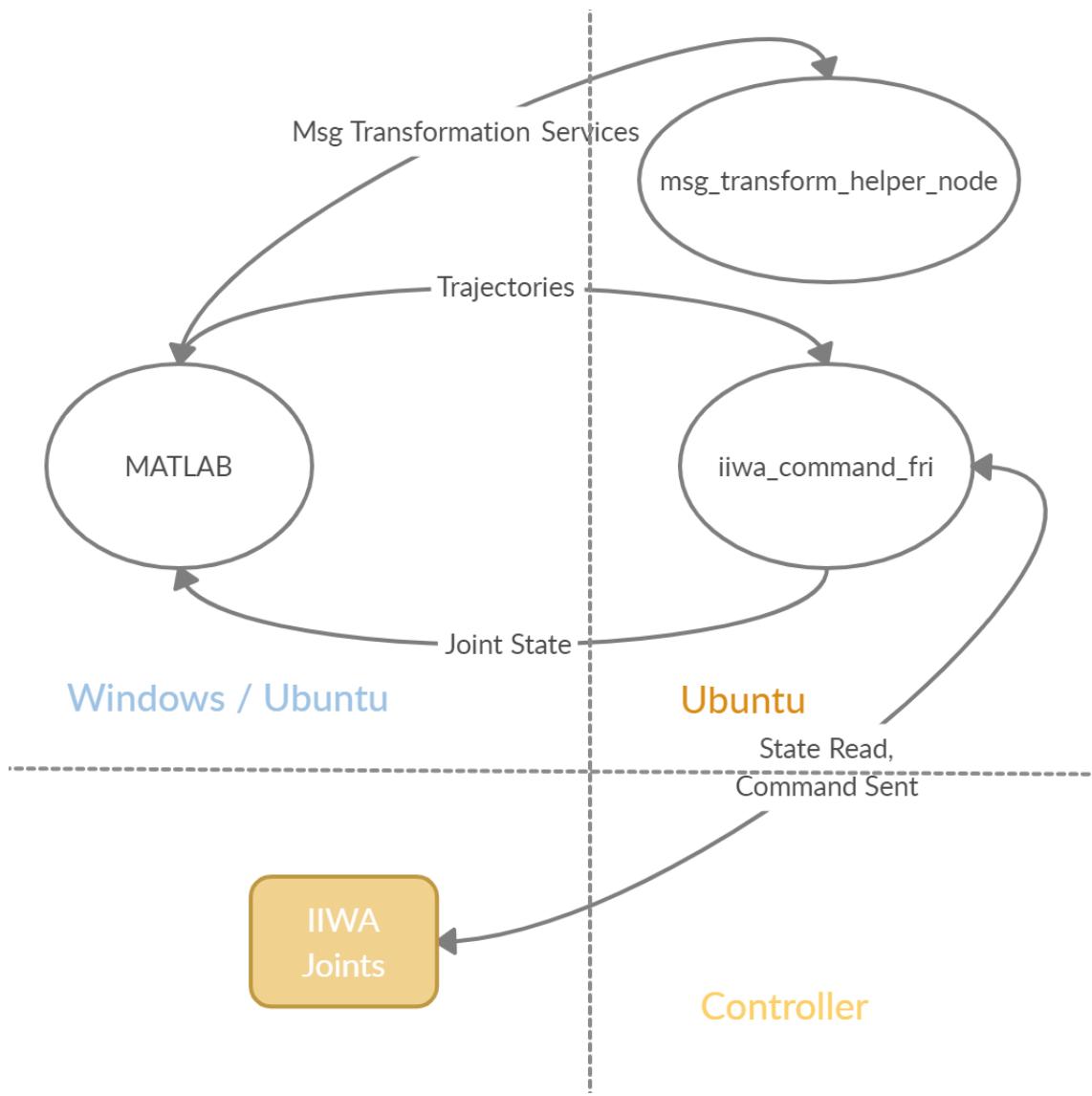


Fig. E.2. Esquema de interacción entre MATLAB y ROS con FRI.

## F. INTERFAZ DE USUARIO

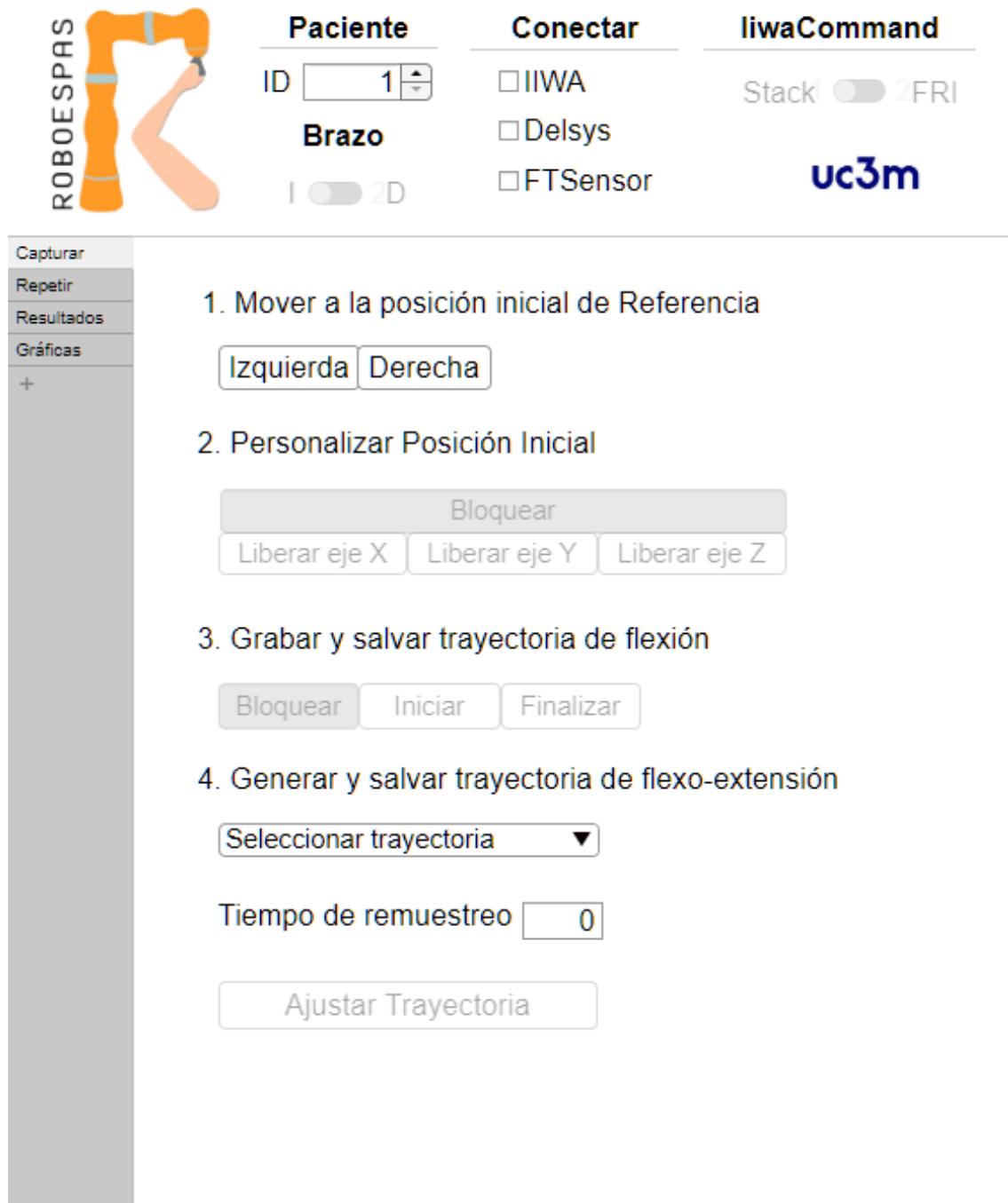


Fig. F.1. Interfaz de captura de datos.

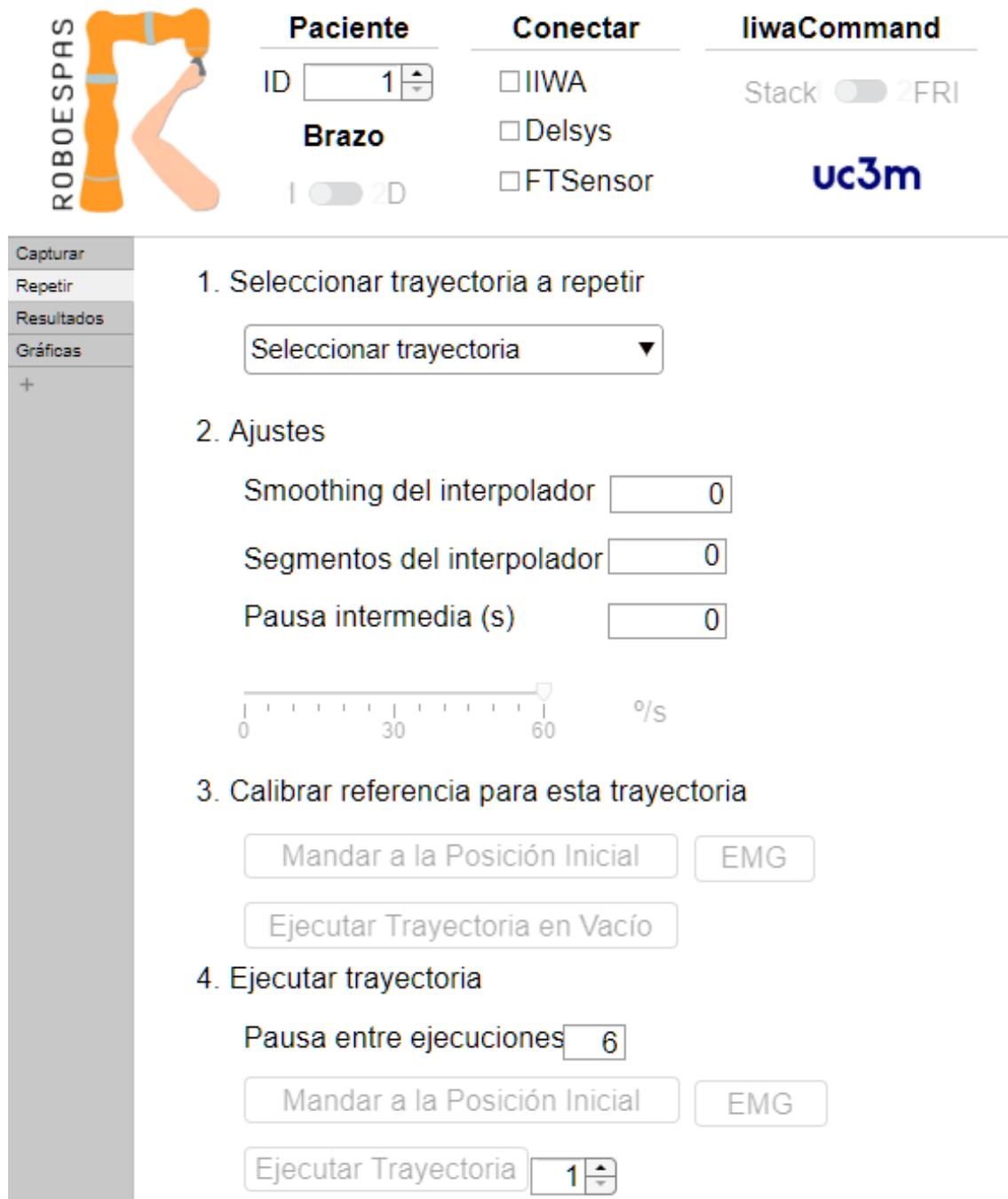


Fig. F.2. Interfaz de reproducción de trayectorias.



Fig. F.3. Interfaz de presentación de gráficas de datos.

## BIBLIOGRAFÍA

- [1] E. D. Oña, P. Sánchez-Herrera, A. Jardón y C. Balaguer, «Review of automated systems for upper limbs functional assessment in neurorehabilitation», *IEEE Access*, vol. 7, pp. 32 352-32 367, 2019.
- [2] E. D. Oña, R. Cano-de La Cuerda, P. Sánchez-Herrera, C. Balaguer y A. Jardón, «A review of robotics in neurorehabilitation: Towards an automated process for upper limb», *Journal of healthcare engineering*, vol. 2018, 2018.
- [3] N. Tejima, «Rehabilitation robotics: a review», *Advanced Robotics*, vol. 14, n.º 7, pp. 551-564, 2001.
- [4] L. Santisteban et al., «Upper limb outcome measures used in stroke rehabilitation studies: a systematic literature review», *PloS one*, vol. 11, n.º 5, e0154792, 2016.
- [5] D. J. Gladstone, C. J. Danells y S. E. Black, «The Fugl-Meyer assessment of motor recovery after stroke: a critical review of its measurement properties», *Neurorehabilitation and neural repair*, vol. 16, n.º 3, pp. 232-240, 2002.
- [6] K. Kontson, I. Marcus, B. Myklebust y E. Civillico, «Targeted box and blocks test: Normative data and comparison to standard tests», *PloS one*, vol. 12, n.º 5, e0177965, 2017.
- [7] E. D. Oña, J. A. García, W. Raffe, A. Jardón y C. Balaguer, «Assessment of manual dexterity in VR: Towards a fully automated version of the box and blocks test», *Studies in Health Technology and Informatics*, vol. 266, pp. 57-62, 2019.
- [8] E. D. Oña, A. Jardón y C. Balaguer, «Automatic Assessment of Arm Motor Function and Postural Stability in Virtual Scenarios: Towards a Virtual Version of the Fugl-Meyer Test», en *2020 IEEE 8th International Conference on Serious Games and Applications for Health (SeGAH)*, IEEE, 2020, pp. 1-6.
- [9] E. Rocon et al., «Design and validation of a rehabilitation robotic exoskeleton for tremor assessment and suppression», *IEEE Transactions on neural systems and rehabilitation engineering*, vol. 15, n.º 3, pp. 367-378, 2007.
- [10] F. Vivancos-Matellano et al., «Guía del tratamiento integral de la espasticidad», *Rev Neurol*, vol. 45, n.º 6, pp. 365-375, 2007.
- [11] F. Biering-Sørensen, J. B. Nielsen y K. Klinge, «Spasticity-assessment: a review», *Spinal cord*, vol. 44, n.º 12, pp. 708-722, 2006.
- [12] J. Mehrholz et al., «Reliability of the Modified Tardieu Scale and the Modified Ashworth Scale in adult patients with severe brain injury: a comparison study», *Clinical rehabilitation*, vol. 19, n.º 7, pp. 751-759, 2005.
- [13] S. Q. Aguilar, P. Claudia, D. César y J. G. F. Javier, «Espasticidad en adultos», *Revista Mexicana de Neurociencia*, vol. 10, n.º 2, pp. 112-121, 2009.

- [14] K. S. Kim, J. H. Seo y C. G. Song, «Portable measurement system for the objective evaluation of the spasticity of hemiplegic patients based on the tonic stretch reflex threshold», *Medical engineering & physics*, vol. 33, n.º 1, pp. 62-69, 2011.
- [15] J.-Y. Kim, G. Park, S.-A. Lee e Y. Nam, «Analysis of Machine Learning-Based Assessment for Elbow Spasticity Using Inertial Sensors», *Sensors*, vol. 20, n.º 6, p. 1622, 2020.
- [16] H. Wang et al., «Spasticity assessment based on the maximum isometrics voluntary contraction of upper limb muscles in post-stroke hemiplegia», *Frontiers in neurology*, vol. 10, p. 465, 2019.
- [17] L. Bar-On et al., «A clinical measurement to quantify spasticity in children with cerebral palsy by integration of multidimensional signals», *Gait & posture*, vol. 38, n.º 1, pp. 141-147, 2013.
- [18] C. A. McGibbon, A. Sexton, M. Jones y C. O'Connell, «Elbow spasticity during passive stretch-reflex: clinical evaluation using a wearable sensor system», *Journal of neuroengineering and rehabilitation*, vol. 10, n.º 1, p. 61, 2013.
- [19] F. Posteraro et al., «Technologically-advanced assessment of upper-limb spasticity: a pilot study», *Eur J Phys Rehabil Med*, vol. 54, pp. 536-44, 2018.
- [20] R. S. Calabro et al., «Is two better than one? Muscle vibration plus robotic rehabilitation to improve upper limb spasticity and function: A pilot randomized controlled trial», *PLoS ONE* 12, 2017.
- [21] J. Gäverth, M. Sandgren, P. G. Lindberg, H. Forssberg y A.-C. Eliasson, «Test-retest and inter-rater reliability of a method to measure wrist and finger spasticity», *Journal of rehabilitation medicine*, vol. 45, n.º 7, pp. 630-636, 2013.
- [22] A. Centen, C. R. Lowrey, S. H. Scott, T.-T. Yeh y G. Mochizuki, «KAPS (kinematic assessment of passive stretch): a tool to assess elbow flexor and extensor spasticity after stroke using a robotic exoskeleton», *Journal of neuroengineering and rehabilitation*, vol. 14, n.º 1, p. 59, 2017.
- [23] G. Fazekas, M. Horvath y A. Toth, «A novel robot training system designed to supplement upper limb physiotherapy of patients with spastic hemiparesis», *International Journal of Rehabilitation Research*, vol. 29, n.º 3, pp. 251-254, 2006.
- [24] M. Sin, W.-S. Kim, K. Cho, S. Cho y N.-J. Paik, «Improving the test-retest and inter-rater reliability for stretch reflex measurements using an isokinetic device in stroke patients with mild to moderate elbow spasticity», *Journal of Electromyography and Kinesiology*, vol. 39, pp. 120-127, 2018.
- [25] R. de-la-Torre, E. D. Oña, C. Balaguer y A. Jardón, «Robot-Aided Systems for Improving the Assessment of Upper Limb Spasticity: A Systematic Review», *Sensors*, vol. 20, n.º 18, p. 5251, 2020.

- [26] C. Hennersperger et al., «Towards MRI-based autonomous robotic US acquisitions: a first feasibility study», *IEEE transactions on medical imaging*, vol. 36, n.º 2, pp. 538-548, 2017.
- [27] O. Bottema y B. Roth, *Theoretical kinematics*. Courier Corporation, 1990, vol. 24.
- [28] L. I. G. Calandín, «Modelado cinemático y control de robots móviles con ruedas», Tesis doct., Universidad Politécnica de Valencia, 2006.
- [29] F. Reuleaux, *The kinematics of machinery: outlines of a theory of machines*. Courier Corporation, 2013.
- [30] A. Barrientos, L. F. Peñin, C. Balaguer y R. Aracil, *Fundamentos de robótica*. McGraw-Hill Madrid, 2007, vol. 2.
- [31] R. S. Hartenberg y J. Denavit, «A kinematic notation for lower pair mechanisms based on matrices», *Journal of applied mechanics*, vol. 77, n.º 2, pp. 215-221, 1955.
- [32] P. I. Corke, «A simple and systematic approach to assigning Denavit–Hartenberg parameters», *IEEE transactions on robotics*, vol. 23, n.º 3, pp. 590-594, 2007.
- [33] L. F. Giraldo, E. Delgado y G. Castellanos, «Cinemática inversa de un brazo robot utilizando algoritmos genéticos», *Revista Avances en Sistemas e Informática*, vol. 3, n.º 1, pp. 29-34, 2006.
- [34] C. Lee y M. Ziegler, «Geometric approach in solving inverse kinematics of PUMA robots», *IEEE Transactions on Aerospace and Electronic Systems*, n.º 6, pp. 695-706, 1984.
- [35] L. Carvalho y E. Gaspar, «The solution of the inverse kinematic problem of robot arm with neural networks», en *IX Brazilian Congress on Mechanical Engineering, Brasil*, 1991.
- [36] EsAcademic, *Ángulos de Euler*, <https://esacademic.com/dic.nsf/eswiki/85423>, [Accessed: 23 de Octubre de 2020], 2010.
- [37] G. G. Slabaugh, «Computing Euler angles from a rotation matrix», *Retrieved on August*, vol. 6, n.º 2000, pp. 39-63, 1999.
- [38] V. Kumar, «The Theorems of Euler and Chasles», *University of Pennsylvania School of Engineering and Applied Science, I-Net, USA*, 2000.
- [39] R. S. Ball, «The theory of screws: A study in the dynamics of a rigid body», *Mathematische Annalen*, vol. 9, n.º 4, pp. 541-553, 1876.
- [40] K. Kuratowski y R. R. Vidal, *Introducción a la teoría de conjuntos ya la topología*. Vicens-Vives, 1973.
- [41] *Lenguaje matemático, conjuntos y números*, [https://www.sanzytorres.es/static/pdf/61021039\\_LRRVldM.pdf](https://www.sanzytorres.es/static/pdf/61021039_LRRVldM.pdf), [Accessed: 19 de Septiembre de 2020].
- [42] C. Chevalley, *Theory of Lie groups*. Courier Dover Publications, 2018.
- [43] N. Bourbaki, *Groupes et algèbres de Lie: Chapitres 2 et 3*. Springer Science & Business Media, 2007.

- [44] B. Hall, *Lie groups, Lie algebras, and representations: an elementary introduction*. Springer, 2015, vol. 222.
- [45] J. Davidson y K. Hunt, «The active interpretation and the active transformation», en *Robots and Screw Theory: Applications of Kinematics and Statics to Robotics*, Oxford University Press, 2004.
- [46] R. M. Murray, Z. Li, S. S. Sastry y S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.
- [47] J. M. Selig, *Geometric fundamentals of robotics*. Springer Science & Business Media, 2004.
- [48] K. M. Lynch y F. C. Park, *Modern Robotics*. Cambridge University Press, 2017.
- [49] B. Siciliano y O. Khatib, *Springer handbook of robotics*. Springer, 2016.
- [50] F. Bullo y A. D. Lewis, «Geometric Control of Mechanical Systems», 2005.
- [51] J. M. Pardos Gotor, «Algoritmos de geometría diferencial para la locomoción y navegación bípedas de robots humanoides: Aplicación al robot RH0», 2005.
- [52] B. E. Paden, «Kinematics and control of robot manipulators», *PhDT*, 1985.
- [53] R. Penrose, «A generalized inverse for matrices», en *Mathematical proceedings of the Cambridge philosophical society*, Cambridge University Press, vol. 51, 1955, pp. 406-413.
- [54] LBR iiwa KUKA, <https://www.kuka.com/es-es/productos-servicios/sistemas-de-robot/robot-industrial/lbr-iiwa>, [Accessed: 15 de Septiembre de 2020].
- [55] Delsys, *Trigno Wireless EMG Sensors*, <https://delsys.com/trigno/sensors/>, [Accessed: 23 de Octubre de 2020].
- [56] A. I. Automation, *F/T Sensor Delta SI-660-60*, [https://www.ati-ia.com/products/ft/ft\\_models.aspx?id=Delta](https://www.ati-ia.com/products/ft/ft_models.aspx?id=Delta), [Accessed: 23 de Octubre de 2020].
- [57] MATLAB, <https://es.mathworks.com/products/matlab.html>, [Accessed: 10 de Octubre de 2020].
- [58] OpenSim, <https://opensim.stanford.edu/>, [Accessed: 20 de Octubre de 2020].
- [59] A. Gordillo Dagallier, *Tesis de maestría: Incorporación de sensores electromiográficos al sistema de evaluación de espasticidad ROBOESPAS*, Universidad Carlos III de Madrid, Madrid, 2020.
- [60] E. Saénz Aldea, *Tesis de maestría: Evaluación de modelos biomecánicos para espasticidad mediante OpenSim*, Universidad Carlos III de Madrid, Madrid, 2019.
- [61] M. Millard, T. Uchida, A. Seth y S. L. Delp, «Flexing computational muscle: modeling and simulation of musculotendon dynamics», *Journal of biomechanical engineering*, vol. 135, n.º 2, 2013.

- [62] D. G. Thelen, «Adjustment of muscle mechanics model parameters to simulate dynamic contractions in older adults», *J. Biomech. Eng.*, vol. 125, n.<sup>o</sup> 1, pp. 70-77, 2003.
- [63] E. M. Arnold, S. R. Ward, R. L. Lieber y S. L. Delp, «A model of the lower limb for analysis of human movement», *Annals of biomedical engineering*, vol. 38, n.<sup>o</sup> 2, pp. 269-279, 2010.
- [64] G. Velásquez, *Tesis de maestría: Machine learning aplicado al aprendizaje del modelo biológico del brazo robótico*, Universidad Carlos III de Madrid, Madrid, 2020.
- [65] C. Hou, Y. Zhao, G. Song y J. Wang, «Gravity Compensation of KUKA LBR IIWA Through Fast Robot Interface», en *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, IEEE, 2018, pp. 164-168.
- [66] *ROS Message Type: trajectorymsgs/JointTrajectory*, [http://docs.ros.org/en/kinetic/api/trajectory\\_msgs/html/msg/JointTrajectory.html](http://docs.ros.org/en/kinetic/api/trajectory_msgs/html/msg/JointTrajectory.html), [Accessed: 12 de Octubre de 2020].
- [67] J. M. Pardos-Gotor, «Screw Theory for Robotics: A practical approach for Modern Robot KINEMATICS», 2018.
- [68] J. Ahlberg, J. Walsh, R. Bellman y E. N. Nilson, *The theory of splines and their applications*. Academic press, 1967.
- [69] M. Kelly, *MATLAB Central File Exchange: fitSplineToData*, <https://es.mathworks.com/matlabcentral/fileexchange/64119-fitsplinetodata>, [Accessed: 20 de Octubre de 2020], 2017.
- [70] K. E. Chlouverakis y J. Sprott, «Chaotic hyperjerk systems», *Chaos, Solitons & Fractals*, vol. 28, n.<sup>o</sup> 3, pp. 739-746, 2006.
- [71] G. Guennebaud, B. Jacob et al., *Eigen v3*, <http://eigen.tuxfamily.org>, 2010.
- [72] Stanford Artificial Intelligence Laboratory et al., *Robotic Operating System*, <https://wiki.ros.org/kinetic>, ROS Kinetic Kame.
- [73] S. Virga, *IiwaStack WIKI*, [https://github.com/IFL-CAMP/iiwa\\_stack/wiki](https://github.com/IFL-CAMP/iiwa_stack/wiki), [Accessed: 7 de Noviembre de 2020].