ESCUELA DE INGENIERÍAS INDUSTRIALES

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

TRABAJO FIN DE MÁSTER

IMPLEMENTACIÓN DE UN INTERFAZ MULTIMODAL PARA EL MANEJO DE UN MANIPULADOR KUKA COMO ROBOT QUIRÚRGICO

MÁSTER EN INGENIERÍA INDUSTRIAL

MÁLAGA, NOVIEMBRE DE 2023

AUTOR
Javier Lara Juárez

TUTORA
Irene Rivas Blanco

COTUTORA
Mª Carmen López Casado

Resumen

Este Trabajo de Fin de Máster está enmarcado dentro del campo de la robótica quirúrgica, y su objetivo principal es el de implementar una interfaz multimodal en el entorno del sistema operativo ROS para el control de un robot LBR iiwa de la marca KUKA con fines quirúrgicos. Para lograr dicho objetivo, se abordan dos funciones esenciales en el control de un robot quirúrgico: la capacidad de realizar movimientos de pivote alrededor de un punto de fulcro y la capacidad de configuración flexible de todos los parámetros relevantes del robot.

La interfaz multimodal desarrollada debe estar plenamente integrada en el entorno de ROS y ser capaz de cambiar entre diferentes modos de funcionamiento del robot. Además, es esencial que esta interfaz permita ajustar todos los parámetros de configuración del robot quirúrgico, como la ubicación del punto de fulcro y la longitud de la herramienta. Un aspecto fundamental de este trabajo es proporcionar una documentación exhaustiva de todo el proceso llevado a cabo. De esta manera, el trabajo se convierte en una base sólida para futuras investigaciones y desarrollos relacionados con el manipulador KUKA en cuestión.

Palabras clave: Robótica quirúrgica, ROS, interfaz multimodal.

Abstract

This Master's Thesis is framed within the field of surgical robotics, and its main objective is to implement a multimodal interface in the ROS (Robot Operating System) environment for controlling a KUKA brand LBR iiwa robot for surgical purposes. To achieve this goal, two essential functions in surgical robot control are addressed: the ability to perform pivot movements around a fulcrum point and the flexible configuration of all relevant robot parameters.

The developed multimodal interface must be fully integrated into the ROS environment and be able to switch between different modes of operation of the robot. In addition, it is essential that this interface allows adjustment of all configuration parameters of the surgical robot, such as the location of the fulcrum point and the length of the tool. A fundamental aspect of this work is to provide a thorough documentation of the entire process carried out. In this way, the work becomes a solid basis for future research and development related to the KUKA manipulator in question.

Keywords: Surgical robotics, ROS, multimodal interface.

Índice

Índice de figuras

Índice de cuadros

Índice de algoritmos

1. Introducción

La robótica quirúrgica se ha consolidado como un campo de investigación y aplicación en constante evolución. La importancia de este tipo de investigación radica en su contribución a la mejora de la precisión y eficacia en las cirugías, lo que puede tener un impacto significativo en el bienestar de los pacientes y en la práctica médica en general. Este Trabajo de Fin de Máster se sitúa en el contexto de la robótica quirúrgica y se enfoca en un objetivo principal: implementar una interfaz multimodal en el entorno del sistema operativo ROS (Robot Operating System) para el control de un robot de la marca KUKA con fines quirúrgicos.

El proyecto aborda dos funciones críticas en el control de robots quirúrgicos: la capacidad de realizar movimientos de pivote alrededor de un punto de fulcro y la capacidad de configuración flexible de todos los parámetros relevantes del robot. Estas funciones son esenciales para permitir realizar procedimientos quirúrgicos complejos con mayor precisión y facilidad. La implementación de una interfaz multimodal integrada en ROS facilitará la transición entre diferentes modos de funcionamiento del robot, brindando una mayor flexibilidad y control del manipulador para su uso en un entorno de investigación (laboratorio).

Un aspecto fundamental de este proyecto es la documentación detallada de todo el proceso. Esto no solo garantiza la reproducibilidad de los resultados, sino que también sienta las bases para futuras investigaciones y trabajos fin de estudios relacionados con el uso y la ampliación de funcionalidades del manipulador KUKA como robot quirúrgico.

1.1. Antecedentes y motivación

La práctica de la cirugía mínimamente invasiva (CMI) o laparoscópica se ha consolidado como un estándar en numerosas intervenciones quirúrgicas, brindando notables ventajas a los pacientes. A pesar de sus beneficios, este enfoque plantea desafíos sustanciales para los cirujanos, como la carencia de visión directa, limitaciones en la percepción táctil y restricciones en la movilidad de los instrumentos quirúrgicos. En respuesta a estas limitaciones y con el propósito de mejorar la eficiencia de los profesionales de la salud, ha surgido la cirugía asistida por robots (RAS, de su acrónimo en inglés Robot Assisted Surgery). La RAS ha demostrado ser efectiva al proporcionar movimientos más exactos y naturales de los instrumentos quirúrgicos, lo que ha generado un marcado interés en la comunidad global y ha impulsado inversiones económicas significativas por parte de las principales economías. Se estima que este mercado experimentará un crecimiento anual compuesto del 10.7 % hasta 2029, alcanzando un valor proyectado de 15.43 mil millones de dólares [1].

En el ámbito académico, la comunidad científica ha demostrado un enérgico interés en el campo de la robótica quirúrgica, reflejado en miles de publicaciones en las últimas décadas y en la concepción de numerosos proyectos destinados a impulsar esta disciplina. Una figura destacada en este ámbito es Intuitive Surgical, una empresa de renombre que está promoviendo la investigación en robótica quirúrgica al respaldar activamente a la comunidad científica. Lo hace mediante la provisión de plataformas de investigación, como el Kit de Investigación Da Vinci (dVRK), una parte integral del sistema quirúrgico Da Vinci. Este respaldo fomenta la colaboración y la sinergia entre diversos grupos de investigación [2].

A pesar de la amplia evidencia que respalda la eficacia de los robots quirúrgicos como herramientas para mejorar las habilidades de los cirujanos, la función de estos sistemas normalmente es limitada. Actualmente, algunos sistemas, como el robot Da Vinci, ofrecen asistencia real durante las intervenciones, aunque solo replicando con rigurosidad los movimientos realizados por los cirujanos desde una consola principal hacia una plataforma esclava. En consecuencia, el interés de los investigadores e investigadoras en el desarrollo de otros enfoques sigue creciendo con el objetivo de optimizar y facilitar aún más las tareas quirúrgicas y reducir la carga de trabajo de los cirujanos.

Este Trabajo de Fin de Máster se desarrolla en el Grupo de Investigación de Robótica Médica del Departamento de Ingeniería de Sistemas y Automática de la Universidad de Málaga, y tiene como objetivo equipar a un robot KUKA LBR iiwa 7 R800 con los mecanismos necesarios para ser empleado como robot quirúrgico. Para ello, se centra en el desarrollo de una interfaz multimodal para controlar el robot tanto en entornos reales como simulados, cuyos parámetros de configuración sean ajustables, y en la generación de documentación detallada de todo el proceso, permitiendo la reproducibilidad de los resultados y sentar las bases para futuros trabajos fin de estudios relacionados con el manipulador KUKA como robot quirúrgico.

1.2. Objetivos

En este contexto amplio y desafiante de la robótica quirúrgica, el objetivo central de este proyecto reside en la implementación de una interfaz multimodal, integrado de manera fluida en el sistema operativo ROS (Robot Operating System), diseñado para el control de un manipulador KUKA LBR iiwa 7 R800 como robot quirúrgico.

La consecución de este propósito implica:

- **Identificar las funcionalidades del sistema:** Se centra en realizar un análisis detallado de las funcionalidades necesarias para un robot quirúrgico, incluyendo características típicas y movimientos a realizar. Además, se analizan las especificaciones técnicas, capacidades de movimiento y componentes clave del robot KUKA LBR iiwa 7 R800 para su uso quirúrgico y se exploran las vías de comunicación disponibles para establecer una conexión efectiva y segura con el robot.
- **Diseñar una interfaz multimodal para cumplir con las especificaciones:** Se centra en la implementación de las funciones esenciales de movimiento propias de los robots quirúrgicos, que incluyen movimiento articular, movimiento cartesiano libre y movimiento cartesiano de pivoteo alrededor de un punto de fulcro. Estos componentes permiten al robot replicar con fiabilidad los movimientos realizados por los cirujanos. Además, la interfaz a desarrollar debe permitir alternar el modo de operación del robot y ofrecer flexibilidad para configurar los parámetros relevantes del robot, como la posición del punto de fulcro y la longitud de la herramienta empleada.
- **Generación de documentación efectiva:** Se centra en la obtención de documentación efectiva para el proyecto, a través de la utilización de la herramienta de generación de documentación Doxygen. Se busca documentar el código del proyecto de manera que, además de facilitar su comprensión, mantenimiento y colaboración, sirva de base para futuros trabajos que pretendan ampliar las funcionalidades del robot.

1.3. Metodología

Para llevar a cabo los objetivos descritos anteriormente, se desarrollarán las siguientes fases de trabajo:

1. **Investigación inicial y análisis del contexto:** Se inicia con una fase de investigación inicial para comprender el contexto de la robótica quirúrgica. Esto incluye una revisión exhaustiva del estado del arte en Cirugía Mínimamente Invasiva, durante la cual, se identifican diferentes tipos de cirugía mínimamente invasiva, se presentan ejemplos de robots utilizados en CMI y se exploran sus aplicaciones. Este análisis del estado del arte sienta las bases para la identificación de las funcionalidades necesarias que debe tener un sistema robótico para su empleo en cirugía quirúrgica.
2. **Análisis y preparación del robot KUKA:** Análisis del robot KUKA LBR iiwa 7 R800, de sus capacidades de movimiento y de sus vías de comunicación efectivas. Además, se lleva a cabo la creación de un diseño modular para sus funcionalidades como robot quirúrgico con el fin de permitir su integración en distintos modos de funcionamiento.
3. **Implementación de la interfaz multimodal:** Implementación de todas las funcionalidades definidas en la fase anterior y, posteriormente, se integrarán en el sistema operativo ROS.
4. **Experimentación y recopilación de datos:** Realización de experimentos para poner a prueba la interfaz multimodal en escenarios simulados y reales. Recopilación de los datos y presentación de los resultados obtenidos.
5. **Documentación de la API:** Realización de una documentación exhaustiva de la interfaz desarrollada, de manera que el software desarrollado pueda utilizarse como base para futuras investigaciones que se realicen.

1.4. Estructura de la memoria

El contenido de este Trabajo Fin de Máster se ha estructurado atendiendo a las siguientes secciones:

- **Sección 1 - Introducción:** Esta primera sección proporciona una visión general del trabajo, describe el contexto, los objetivos de la investigación y presenta la estructura general de la memoria.

- **Sección 2 - Estado del arte:** En esta sección, se profundiza en el campo de la Cirugía Mínimamente Invasiva desde varios ángulos. Primero, se explora en detalle el concepto y los tipos de Cirugía Mínimamente Invasiva, incluyendo sus características y ventajas. Luego, se presentan ejemplos de robots utilizados en el ámbito de CMI, detallando su diseño y aplicaciones. Finalmente, se profundiza en las ventajas que conlleva el uso de robots en CMI, considerando tanto los beneficios clínicos como técnicos. Esta sección proporciona un sólido fundamento para comprender el panorama actual de la CMI en relación con la robótica quirúrgica y las interfaces multimodales.
- **Sección 3 - Diseño de la interfaz multimodal:** Esta sección se inicia con un análisis detallado de la arquitectura global que caracteriza a un sistema robótico quirúrgico. Posteriormente, se realiza una descripción de las funcionalidades que van a estar presentes en la interfaz multimodal a desarrollar para después llevar a acabo el diseño de las mismas.
- **Sección 4 - Implementación:** En esta sección se documenta en detalle la ejecución práctica del sistema de interfaz multimodal para el manejo del manipulador KUKA como robot quirúrgico. Se inicia con un análisis exhaustivo de las características del robot KUKA LBR iiwa 7 R800, incluyendo sus especificaciones técnicas, capacidades de movimiento y componentes clave. Además, se profundiza en la exploración de las vías de comunicación disponibles con el robot para establecer una conexión efectiva y segura. Posteriormente, se describen los componentes físicos y software utilizados en la implementación de la interfaz, las decisiones de diseño y los métodos de integración. Aquí se explica cómo se logra la comunicación efectiva entre el usuario y el robot, así como los aspectos técnicos clave relacionados con la seguridad y la precisión en la manipulación quirúrgica. Por último, se destaca la importancia de la documentación en el desarrollo de proyectos de robótica quirúrgica y software. Se introduce Doxygen, una herramienta de generación de documentación de código abierto, resaltando su función principal de facilitar la comprensión del código, mantenerlo y permitir la colaboración efectiva. La sección detalla el proceso de generación de documentación Doxygen para este proyecto de investigación, enfocándose en los archivos centrales de la interfaz multimodal.
- **Sección 5 - Experimentación y resultados:** En esta etapa, se describe los experimentos realizados al sistema implementado. Se detallan los procedimientos de prueba, los escenarios simulados o reales, y cómo se recopilan los datos de manera sistemática. Luego, se presentan los resultados obtenidos de estos experimentos de forma gráfica y numérica, y se realiza un análisis de los datos recopilados.
- **Sección 6 - Conclusiones y futuros trabajos:** En esta último sección, se extraen las conclusiones fundamentales de la investigación realizada. Se destaca cómo la implementación de la interfaz multimodal ha mejorado el manejo del manipulador KUKA como robot quirúrgico y cuáles son las implicaciones clínicas y técnicas de estos hallazgos. Además, se exploran las posibles direcciones para futuras investigaciones, como mejoras adicionales en la interfaz, aplicaciones clínicas específicas y áreas de estudio adicionales en el campo de la cirugía robótica.

2. Estado del arte

2.1. Cirugía Mínimamente Invasiva

La cirugía mínimamente invasiva es un enfoque revolucionario en la realización de procedimientos quirúrgicos, y su importancia ha crecido significativamente desde mediados de la década de 1990. En este tipo de cirugía, los procedimientos se llevan a cabo mediante pequeñas incisiones en el cuerpo del paciente, lo que conlleva numerosas ventajas en comparación con la cirugía convencional abierta. Estas incisiones se aplican principalmente para la cámara, que proporciona al cirujano una visión interna del paciente, y para los instrumentos médicos [3].

Todos los procedimientos de CMI conllevan una disminución de la agresión quirúrgica, lo que resulta en beneficios notables como:

- Reducción de la inflamación relacionada con la cirugía y mejora de la respuesta inmunológica.
- Menor dolor postoperatorio debido a la ausencia de incisiones extensas.
- Menos complicaciones en la herida operatoria debido al tamaño reducido de las incisiones, lo que acelera el proceso de cicatrización.
- Menor tiempo de hospitalización gracias a una recuperación más rápida y sencilla [4].

La CMI ha evolucionado para abordar una amplia variedad de procedimientos quirúrgicos, y ha dado lugar a varios tipos de cirugía mínimamente invasiva, entre los que destacan:

- **Laparoscopía y SPA (Single Port Access):** La laparoscopía es una técnica común en cirugía mínimamente invasiva que implica la inserción de un laparoscopio y otros instrumentos a través de pequeñas incisiones en la cavidad abdominal. La técnica SPA (Single Port Access) es una variante de la laparoscopía en la que se realiza una única incisión, generalmente en la cicatriz umbilical, para insertar todos los instrumentos necesarios. SPA es menos invasiva y reduce el tiempo de recuperación del paciente en comparación con la laparoscopía convencional [5].
- **NOTES (Natural Orifice Translumenal Endoscopic Surgery):** Esta técnica implica la introducción de un endoscopio flexible a través de orificios naturales del cuerpo, como la cavidad oral, la vagina, la uretra o el ano, para acceder a cavidades internas. Es una forma de CMI que no deja cicatrices externas y permite una recuperación más rápida [6].

A pesar de las ventajas notables de la CMI, existen situaciones en las que no se puede garantizar su efectividad debido a la falta de destreza, los grados limitados de libertad y la ausencia de sistemas de retroalimentación de fuerza en los instrumentos. En respuesta a estos desafíos, los robots quirúrgicos han surgido como una solución potencial. Su incorporación en entornos quirúrgicos tiene como objetivo extender las capacidades del cirujano. Estos robots aprovechan las características intrínsecas de los robots, como la alta precisión, la repetitividad y la durabilidad, lo que los convierte en asistentes confiables para los médicos [7].

Las tecnologías de la información han permitido la transición a una nueva generación de CMI al incorporar la visualización en tres dimensiones, nuevas técnicas de fusión de información, retroalimentación de fuerza, percepción de la escala y la reducción de temblores [8]. Además, la realidad virtual se emplea para la planificación preoperatoria y la formación de médicos [9]. Aunque el concepto de usar tecnologías robóticas en la cirugía se propuso en los años 70, se hizo posible en la década de 1990. Hoy en día, existen aplicaciones comerciales de robots en diversas áreas quirúrgicas, incluyendo gastroenterología, neurocirugía y ortopedia, entre otras [10].

2.2. Robots utilizados en CMI

La evolución de la cirugía robótica ha dado lugar a una variedad de sistemas y dispositivos que se han ido utilizando en cirugía mínimamente invasiva. A lo largo de las últimas décadas, varios robots quirúrgicos han sido desarrollados y aplicados en procedimientos clínicos, contribuyendo de gran manera al campo de la medicina. A continuación, se presentan ejemplos de algunos de estos robots, cronológicamente ordenados, con una descripción

detallada de sus características, aplicaciones y avances tecnológicos:

Robot AESOP (Automated Endoscopic System for Optimal Positioning)

El robot AESOP, mostrado en la Figura 1, fue el pionero de la cirugía robótica y marcó un hito fundamental en la evolución de la cirugía mínimamente invasiva y la asistencia quirúrgica. Este robot obtuvo la aprobación de la FDA (Food and Drug Administration) en 1994, convirtiéndose en el primer robot quirúrgico clínicamente aprobado para procedimientos abdominales, un logro histórico.

La génesis de AESOP se remonta a la concesión de investigación de la NASA a Computer Motion, destinada a desarrollar un brazo robótico para el programa espacial de los Estados Unidos. Posteriormente, este brazo se adaptó de manera innovadora para sostener un laparoscopio y asumir el rol del camarógrafo laparoscópico, desencadenando una revolución en la visualización quirúrgica [11].

El robot AESOP, destacó realmente por su capacidad de obedecer comandos de voz, permitiendo a los cirujanos controlar el laparoscopio de manera precisa y eficiente mediante instrucciones específicas. Además, el sistema se destacaba por su enfoque en la seguridad del paciente, al implementar medidas de seguridad que evitaban daños accidentales, incluyendo un mecanismo de desacoplamiento magnético que garantizaba la protección del paciente en situaciones críticas.

Este robot comercial ofrecía una plataforma estable para la cámara, lo cual mejoraba significativamente la calidad de las imágenes y reducía la interferencia de movimientos no deseados en el equipo operatorio. Su capacidad de permitir a los cirujanos trabajar solos en procedimientos laparoscópicos representó un avance clave en la eficiencia quirúrgica al minimizar las oscilaciones indeseadas en las imágenes durante la operación [12].

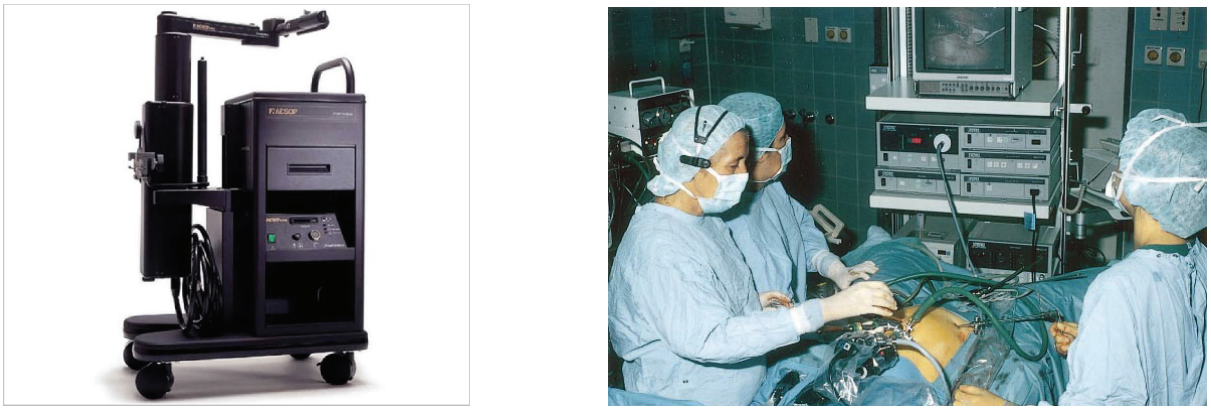


Figura 1: Sistema AESOP [12].

ENDOASSIST

El Endoassist, mostrado en la Figura 2, es un camarógrafo robótico desarrollado por la compañía británica Armstrong Healthcare Ltd, que fue el siguiente sistema de uso comercial aprobado por la FDA para su uso en los Estados Unidos. Aunque la información disponible sobre el Endoassist es limitada, este sistema ofrecía un enfoque único en el control de la cámara durante procedimientos quirúrgicos [13].

Lo que distinguía al Endoassist era la forma en que el cirujano controlaba la vista de la cámara. Utilizando un dispositivo en su frente que emitía rayos infrarrojos, el cirujano apuntaba el rayo al punto en el monitor de video que deseaba observar, y el robot movía la vista de la cámara a dicha posición. Esta metodología permitía a los cirujanos un control riguroso sobre la visualización sin necesidad de comandos de voz u otros dispositivos de control más tradicionales, lo que llegaba a ser beneficioso para aquellos que encontraban distracciones en la charla requerida para controlar sistemas como el AESOP [14].

El brazo del Endoassist se encontraba sujeto a una base separada que podía ser rodada al costado del paciente, lo que facilitaba su ubicación y ajuste durante la cirugía. Este enfoque novedoso en el control de la cámara y la movilidad del sistema representaba una perspectiva interesante en el campo de la cirugía robótica, ofreciendo a los cirujanos una herramienta potencialmente valiosa para mejorar la exactitud y la eficiencia en el quirófano [15].

El Endoassist se sumaba así a la evolución de la cirugía robótica y la telepresencia quirúrgica, ofreciendo un enfoque diferente en la interacción del cirujano con el equipo quirúrgico. Su aprobación por parte de la FDA en los Estados Unidos indicó un interés creciente en la exploración de nuevas tecnologías para mejorar la cirugía mínimamente invasiva.

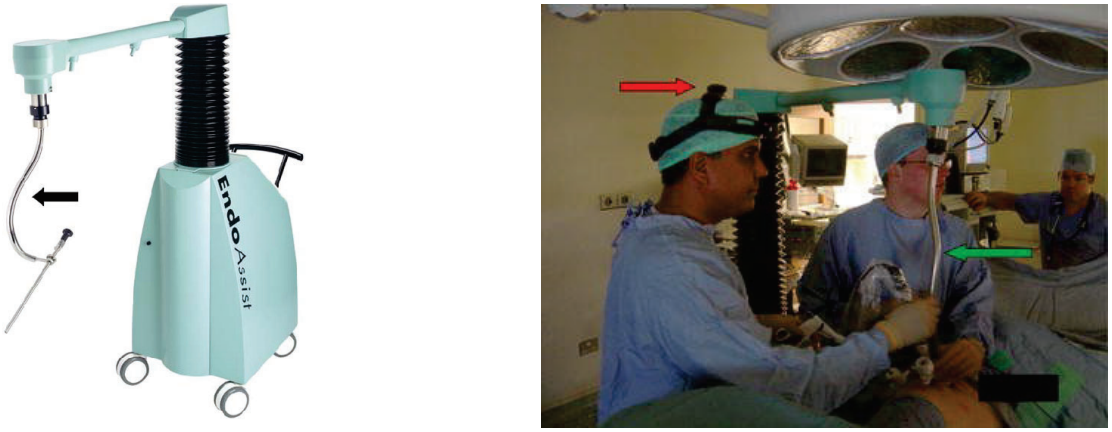


Figura 2: Sistema Endoassist [15].

ZEUS

El sistema robótico Zeus, desarrollado por Computer Motion, fue un avance notable en la cirugía telerobótica y representó una clara evolución de la tecnología utilizada en AESOP. Zeus, como se muestra en la Figura 3, consta de tres brazos independientes, incluyendo un endoscopio robótico controlado por voz para la visualización y dos unidades para sostener instrumentos quirúrgicos. La interfaz mejorada de Zeus permitió un control más ergonómico de los instrumentos quirúrgicos, lo que resultó en una mayor precisión en maniobras laparoscópicas complicadas como suturas. Además, Zeus ofreció una visualización en tres dimensiones del campo operatorio, mejorando la percepción espacial del cirujano y su inmersión en el procedimiento quirúrgicos [16].

Figura 3: Sistema Zeus [17].

Una característica distintiva de Zeus fue la disposición de sus brazos robóticos, ya que estos se encontraban sujetos directamente a la mesa operatoria. Esto permitía una mayor flexibilidad en la reubicación del paciente, especialmente en cirugías abdominales complejas como colectomías. A pesar de que Zeus tenía una aprobación limitada de la FDA en los Estados Unidos, experiencias clínicas en otros lugares, como el Instituto Europeo de Telecirugía, demostró su aplicabilidad en colecistectomías laparoscópicas asistidas, con tiempos operatorios competitivos y pocas complicaciones [11].

El sistema Zeus permitía al cirujano controlar los brazos robóticos de manera remota desde la consola de control, lo que lo convertía en un pionero en la cirugía telerobótica. Además, Zeus demostró ser capaz de realizar cirugía de telepresencia, con cirujanos operando a miles de millas de distancia de los pacientes. Esto abrió nuevas posibilidades en la cirugía telerobótica, con la posibilidad de realizar procedimientos quirúrgicos a larga distancia, pudiendo marcar el inicio de una nueva era en la cirugía [17].

DA VINCI

Da Vinci, la última gran innovación en la cirugía robótica, fue aprobado por la FDA en el año 2000 para realizar cirugías abdominales. Su diseño modular se compone de tres partes principales. El cirujano opera desde una consola que se asemeja a una estación de trabajo ergonómicamente cómoda, donde sus manos se acoplan a los haptics, que actúan como la interfaz principal. En esta consola, se encuentra el sistema de imagen 3-D, que se asemeja a mirar a través de binoculares de campo, permitiendo una percepción tridimensional del campo quirúrgico [11].

En el campo quirúrgico, el sistema Da Vinci, el cual se muestra en la Figura 4, se compone de cuatro brazos robóticos, cada uno con una función específica. Uno de los brazos sostiene la cámara, mientras que los demás brazos laterales están destinados a llevar los instrumentos quirúrgicos. Estos brazos robóticos poseen 6 grados de libertad y se controla mediante 2 mandos manuales y 2 pedales para los pies, siendo altamente móviles, facilitando así la realización de maniobras laparoscópicas [18].

El sistema Da Vinci también permitió la realización de cirugía telerobótica, con experiencias clínicas exitosas en diversos procedimientos gastrointestinales, como colecistectomías y fundoplicaturas. Aunque proporciona una telepresencia en el campo operatorio virtual, facilitando la percepción tridimensional y la orientación del cirujano, también puede llevar al aislamiento del cirujano del entorno de la sala de operaciones. A pesar de estas limitaciones, Da Vinci ha demostrado ser una herramienta valiosa para cirujanos en una variedad de procedimientos actuales, aunque su peso y estructura pueden requerir tiempo adicional en algunas situaciones para reubicar y adaptar los brazos robóticos [19].

Figura 4: Sistema Da Vinci [17].

2.3. Ventajas de la utilización de robots en CMI

La cirugía robótica ha experimentado un crecimiento considerable en las últimas décadas con la aprobación de sistemas como el da Vinci por parte de la Administración de Alimentos y Medicamentos de los Estados Unidos (FDA) en el año 2000. A pesar de los costos asociados, se ha convertido en una parte fundamental del sistema de atención médica en Estados Unidos, con más de 2,800 hospitales invirtiendo en esta tecnología en el año 2017. Esta adopción se ha basado en múltiples ventajas que la cirugía robótica ofrece a pacientes y cirujanos [20]:

1. **Mayor precisión y destreza:** Uno de los aspectos más destacados de la cirugía robótica es su capacidad para proporcionar una precisión y destreza excepcionales. Los sistemas robóticos, como el da Vinci, pueden realizar movimientos microscópicos y eliminar los temblores de las manos humanas. Esto es particularmente valioso en cirugías que requieren una exactitud extrema, como las neurocirugías. Los cirujanos pueden acceder a áreas anatómicas delicadas con un rigor asombroso, mejorando significativamente la seguridad y los resultados de los pacientes.

2. **Recuperación más rápida:** Este tipo de cirujía se asocia con tiempos de recuperación más cortos en comparación con la cirugía abierta. Los pacientes sometidos a cirugías robóticas experimentan menos dolor, menos pérdida de sangre y una estancia hospitalaria más breve. Esto tiene un impacto directo en la calidad de vida postoperatoria de los pacientes, permitiéndoles volver a sus actividades normales en menos tiempo. Además, gracias a esta rápida recuperación, se reduce la carga en los sistemas de atención médica al liberar camas hospitalarias más rápidamente.

3. **Reducción de cicatrices y menos dolor:** La cirugía robótica minimiza el tamaño de las incisiones, lo que se traduce en cicatrices más pequeñas y menos dolor postoperatorio. La estética y la comodidad para los pacientes son aspectos fundamentales, especialmente en procedimientos donde la apariencia física es relevante. Estas incisiones pequeñas también pueden reducir el riesgo de infecciones y complicaciones.

4. **Ampliación de la accesibilidad:** La cirugía robótica ha ampliado la accesibilidad a procedimientos quirúrgicos complejos. Esto es especialmente importante en áreas geográficas donde no se dispone de cirujanos altamente especializados. La telecirugía, una extensión prometedora de la cirugía robótica, permite que cirujanos expertos realicen procedimientos en lugares remotos a través de la tecnología robótica.

5. **Mayor volumen de cirugías:** Un enfoque clave para hacer que la cirugía robótica sea más rentable es aumentar el volumen de cirugías. A medida que más hospitales adoptan esta tecnología, se crea un ciclo de retroalimentación positiva, lo que lleva a un mayor número de cirugías robóticas. Aunque los costos iniciales de adquisición son significativos, este enfoque puede ayudar a amortizarlos y hacer que la tecnología sea más asequible para un mayor número de pacientes.

6. **Futuro prometedor:** El campo de la cirugía robótica sigue evolucionando con nuevos competidores y avances tecnológicos. A medida que la tecnología madura, es probable que sus ventajas se vuelvan aún más evidentes. Aunque los costos iniciales siguen siendo un desafío, se espera que la competencia y la innovación en el campo de la cirugía robótica continúen. Esto podría llevar a una mayor asequibilidad y a una mayor difusión de esta tecnología.

   Es importante señalar que la inversión en cirugía robótica no solo se justifica por sus ventajas actuales, sino también por su potencial a largo plazo para mejorar la calidad de la atención médica y los resultados para los pacientes. La competencia en el mercado de la cirugía robótica también puede llevar a reducciones en los costos, lo que beneficiaría a los sistemas de salud y a los pacientes por igual [20].

2.4. Características de los robots quirúrgicos

En el ámbito de la cirugía robótica, las características de un robot quirúrgico son la piedra angular para garantizar el éxito de las intervenciones, mantener la seguridad del paciente en primer plano y aumentar la eficacia del equipo médico. La robótica ha revolucionado la cirugía al permitir una determinación sin precedentes y una amplia gama de capacidades, lo que a su vez ha creado una demanda significativa para que los robots quirúrgicos cumplan con estándares rigurosos. A continuación, se analizarán en detalle las características principales que un robot genérico debe incorporar para cumplir con estos estándares y satisfacer las necesidades en el ámbito de la cirugía robótica:

- **Precisión y estabilidad:** La precisión y estabilidad son elementos básicos en la cirugía robótica. La habilidad del robot para llevar a cabo movimientos altamente exactos y mantener una posición estable tiene un impacto significativo en la seguridad del paciente y la efectividad de las intervenciones quirúrgicas. La exactitud en los movimientos del robot es esencial para procedimientos delicados, como la extirpación de tumores, donde se deben evitar daños en tejido sano circundante. Además, una posición estable es vital, especialmente a la hora de utilizar herramientas quirúrgicas afiladas y delicadas. Esto es crítico para la seguridad del paciente y la efectividad del procedimiento, ya que cualquier error o inestabilidad podría resultar en lesiones graves o complicaciones [21].
- **Seguridad del paciente**: La precisión y la estabilidad son directamente proporcionales a la protección del paciente. Cualquier error o inestabilidad podría resultar en lesiones graves o complicaciones durante la cirugía. La incorporación de mecanismos de seguridad avanzados, como la detección de colisiones y la prevención de errores, es también importante para garantizar la seguridad del paciente.
- **Sensores y retroalimentación en tiempo real:** Un robot quirúrgico también debe integrar sensores que ofrezcan retroalimentación en tiempo real, incluyendo retroalimentación de posiciones, fuerzas y/o datos visuales. Esta retroalimentación proporciona información crítica al cirujano sobre el estado del paciente y el progreso de la cirugía. Además, mejora la percepción y la capacidad de toma de decisiones durante el procedimiento [22].
- **Control multimodal:** Una característica distintiva que debe presentar un robot quirúrgico es la capacidad de control multimodal, permitiendo a los cirujanos y operadores interactuar con el robot de diversas maneras. Esto se traduce en la posibilidad de realizar movimientos articulares, controlar el robot en coordenadas cartesianas y permitir movimientos alrededor de un punto de fulcro específico. Esto es crucial para adaptarse a las necesidades de procedimientos quirúrgicos variados.
- **Colaboración humano-robot:** En la cirugía robótica, la colaboración efectiva entre humanos y robots es un requisito fundamental. El robot quirúrgico debe ser lo suficientemente adaptable para seguir las instrucciones del cirujano en tiempo real, lo que incluye ajustar los parámetros de sus movimientos según las necesidades del procedimiento. Esto permite una colaboración cercana y fluida con el cirujano, mejorando la eficiencia y la fiabilidad de la cirugía, especialmente en procedimientos delicados y minuciosos como la sutura de vasos sanguíneos o nervios [23].

3. Diseño de la interfaz multimodal

3.1. Arquitectura global de un sistema robótico quirúrgico

Los sistemas robóticos quirúrgicos son sistemas complejos que integran una amplia gama de componentes y tecnologías. La arquitectura global de un sistema robótico quirúrgico se puede dividir en las siguientes capas:

1. **Capa de hardware:** La capa de hardware constituye la base sólida sobre la cual se construye un sistema robótico quirúrgico. En esta capa, se encuentran elementos fundamentales que permiten la ejecución rigurosa y segura de procedimientos quirúrgicos. Los componentes clave de la capa de hardware incluyen:
   - **Robot quirúrgico:** El robot quirúrgico es el protagonista principal de esta capa. Se trata de un brazo mecánico altamente sofisticado diseñado para realizar maniobras quirúrgicas con la máxima exactitud y estabilidad. Su capacidad para movimientos delicados y coordinados es esencial para la cirugía robótica.
   - **Sensores:** Los sensores desempeñan un papel crucial al proporcionar información en tiempo real. Estos dispositivos miden la posición y orientación del robot quirúrgico, lo que garantiza que los movimientos sean concretos y ajustados al entorno quirúrgico. Además, los sensores pueden captar datos relacionados con el paciente y la anatomía.
   - **Herramientas:** La selección precisa y el manejo cuidadoso de herramientas quirúrgicas es fundamentales. Estas herramientas especializadas permiten al robot realizar maniobras quirúrgicas delicadas, garantizando la eficacia y la minuciosidad del procedimiento. La variedad de herramientas permite desempeñar roles específicos según la naturaleza de la intervención.
   - **Dispositivos de visualización:** Los dispositivos de visualización desempeñan un papel crítico al proporcionar al cirujano una visión en tiempo real del campo quirúrgico. Esto incluye la visualización de imágenes médicas, la imagen del sitio quirúrgico y datos relevantes para la toma de decisiones.
2. **Capa de software:** La capa de software es el corazón intelectual del sistema robótico quirúrgico. Aquí es donde se procesan las instrucciones, se planifican los movimientos y se asegura la coordinación sin problemas del sistema. Los elementos esenciales en la capa de software incluyen:
   - **Software de control del robot:** Este componente es el encargado de traducir las instrucciones del cirujano en movimientos específicos del robot. Utiliza algoritmos avanzados para garantizar que el robot realice las maniobras requeridas de manera segura y efectiva.
   - **Software de planificación de movimientos:** El software de planificación de movimientos utiliza la información de los sensores para diseñar y ejecutar movimientos óptimos. Esto implica garantizar que el robot siga una trayectoria segura y eficiente en el área quirúrgica.
   - **Software de visualización:** El software de visualización es esencial para la presentación de datos e imágenes al cirujano. Proporciona una interfaz gráfica intuitiva para que el cirujano interactúe con el sistema y tome decisiones informadas durante la cirugía.
3. **Capa de interfaz humano-máquina:** La capa de interfaz humano-máquina es el punto de contacto entre el cirujano y el sistema robótico quirúrgico. Esta interfaz es necesaria para garantizar una comunicación efectiva y un control riguroso. Sus componentes incluyen:
   - **Consola del cirujano:** La consola del cirujano es la interfaz principal utilizada por el cirujano para controlar el robot quirúrgico y recibir retroalimentación visual del procedimiento en tiempo real. Ésta se divide en dispositivos hápticos para la teleoperación y pantalla para la retroalimentación visual.

Esta arquitectura global, con sus distintas capas pero interconectadas, es la base de la cirugía robótica moderna. El funcionamiento armonioso de estos componentes es esencial para lograr procedimientos quirúrgicos certeros y seguros.

Con el objetivo de brindar una representación más clara de la compleja arquitectura de un sistema robótico quirúrgico, se ha elaborado un esquema que visualiza la interconexión entre sus componentes clave. La Figura 5 presenta esta arquitectura global de manera gráfica, destacando las distintas capas que la componen y su importancia en el funcionamiento conjunto del sistema.

Este esquema visual sirve como un puente fundamental para comprender cómo las capas de hardware, software y la interfaz humano-máquina se interrelacionan en un sistema robótico quirúrgico. A través de esta representación visual, se puede apreciar la interdependencia de los elementos fundamentales que conforman la arquitectura, desde el robot quirúrgico hasta el software de control y la consola del cirujano.

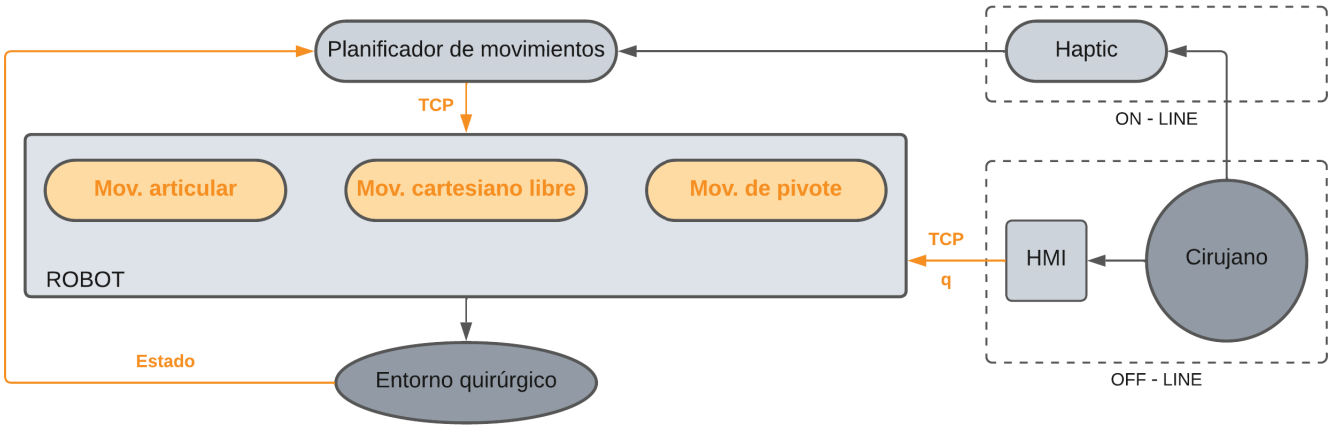


Figura 5: Arquitectura global de un sistema robótico quirúrgico.

La arquitectura global de un sistema robótico quirúrgico se descompone en una serie de etapas altamente coordinadas y esenciales para el éxito de la intervención. En primer lugar, el cirujano establece una comunicación off-line con el sistema robótico a través del Interfaz Humano-Máquina (HMI). Este proceso permite al cirujano modificar los parámetros de configuración del robot y definir su posición inicial antes de la intervención. Es en esta fase donde se configuran las bases que influirán en todo el procedimiento quirúrgico.

Durante la intervención en tiempo real, el cirujano toma el control y comanda los movimientos deseados de la punta de la herramienta, conocida como el Tool Center Point (TCP). Esta comunicación se lleva a cabo a través de una consola de teleoperación o un dispositivo háptico, permitiendo una interacción más exacta y táctil con el robot quirúrgico.

Sin embargo, antes de enviar los comandos de movimiento al robot, entra en juego un componente crítico: el Planificador de Movimientos. Esta entidad se encarga de generar las trayectorias apropiadas para el robot en función de la posición deseada y el estado actual del robot. El Planificador de Movimientos traduce las intenciones del cirujano en movimientos coherentes y seguros del brazo robótico, asegurando la exactitud de cada movimiento durante la cirugía.

Finalmente, dependiendo del tipo de movimiento deseado en un momento dado, el robot debe ser capaz de implementar tres tipos distintos de movimientos. En primer lugar, el robot puede ejecutar un movimiento articular o cartesiano para posicionar el robot en su punto de partida. Este movimiento es esencial para establecer una referencia inicial específica. Esta capacidad cobra una relevancia particular cuando el robot se emplea en un entorno de investigación, donde se llevarán a cabo diversos experimentos que demandarán la habilidad de variar el movimiento del robot de diversas maneras. Además, el robot debe ser capaz de realizar un movimiento de pivote cuando sea necesario mover la herramienta quirúrgica a través de un punto de fulcro específico. Esta versatilidad en los movimientos permite una ejecución detallada y segura de cada paso del procedimiento quirúrgico, adaptándose a las necesidades de la cirugía en tiempo real. La sincronización y coordinación de todas estas etapas son claves para lograr un buen control durante la realización de procedimientos quirúrgicos asistidos por robots.

Los elementos resaltados en naranja en la imagen representan los pilares esenciales de este trabajo, que constituye la base para la construcción de la arquitectura de un sistema robótico quirúrgico. Este estudio se enfocará en el análisis detallado de estos componentes, ya que son el fundamento para poder implementar el resto de bloques de la arquitectura.

3.2. Funcionalidades de la interfaz multimodal

Esta sección se centra en explorar en detalle las funcionalidades esenciales de la interfaz multimodal desarrollada. Cada una de estas funcionalidades son claves en la consecución de procedimientos quirúrgicos certeros y seguros.

A lo largo de este apartado, se va a explorar cada una de estas funcionalidades, destacando su importancia y sus contribuciones a la mejora de la atención médica y la práctica quirúrgica. Cada funcionalidad contribuye al perfeccionamiento de los procedimientos quirúrgicos asistidos por robots.

3.2.1. Movimiento articular y movimiento cartesiano libre

Para lograr una posición inicial concreta y adaptable antes de la cirugía, el robot quirúrgico debe contar con estas dos funcionalidades en su arsenal. Estas capacidades otorgan al cirujano un amplio rango de control para ajustarse a diversas situaciones en el quirófano. Estas funcionalidades son:

- **Movimiento articular:** Esta funcionalidad se refiere a la capacidad del robot quirúrgico para mover cada una de sus articulaciones de forma independiente y determinada.

  El proceso de este movimiento comienza con la entrada de las posiciones articulares deseadas por parte del cirujano. Estas posiciones articulares (en radianes) se transmiten a través de la interfaz en forma de comandos articulares. La interfaz multimodal recibe estos comandos articulares y los procesa para su posterior transmisión al robot.

  Una vez que el robot recibe los comandos articulares, ajusta sus articulaciones de acuerdo con las posiciones deseadas definidas por el cirujano.

- **Movimiento cartesiano libre:** Otra funcionalidad fundamental es la capacidad de controlar la posición y orientación del Tool Center Point (TCP) del robot en un espacio tridimensional. Esta funcionalidad permite realizar movimientos en coordenadas cartesianas, lo que significa que el robot puede moverse de manera flexible en el espacio.

3.2.2. Movimiento cartesiano alrededor de un punto de fulcro

Este tipo de funcionalidad es esencial en el funcionamiento normal de un robot quirúrgico, especialmente en procedimientos de cirugía laparoscópica. Se utiliza para realizar movimientos precisos alrededor de un punto de fulcro específico, previamente definido en la herramienta quirúrgica. Esta característica permite al operador controlar la posición del Tool Center Point (TCP) a través de movimientos de pivote concretos, lo que resulta esencial en cirugías que implican la manipulación cuidadosa de herramientas quirúrgicas alrededor de un punto de referencia crítico.

3.2.3. Recepción del estado del robot

Controlar la posición del robot en todo momento y supervisar su estado son elementos cruciales para el control de trayectorias. Esta información es necesaria para el planificador de movimientos, ya que le permite generar las trayectorias apropiadas para el movimiento del robot. La capacidad de recibir retroalimentación sobre la posición y el estado de las articulaciones del brazo robótico garantiza un control preciso en la ejecución de trayectorias y movimientos del robot.

Además de la información sobre las articulaciones, esta función también permite la transmisión de datos vitales sobre la posición y orientación del Tool Center Point del brazo robótico. Esto es crucial para el control del robot en procedimientos quirúrgicos, ya que el Tool Center Point es la parte del robot que interactúa directamente con el entorno quirúrgico.

La funcionalidad de recepción del estado del robot establece una comunicación efectiva dentro del sistema robótico y garantiza que el robot opere de manera precisa y segura durante los procedimientos quirúrgicos, mejorando

así la calidad de la atención médica y la seguridad del paciente.

3.2.4. Recepción de la posición deseada del TCP

Esta funcionalidad representa un elemento crítico en la interfaz multimodal, ya que se centra en la capacidad por parte de ésta de recibir y trabajar con comandos de posición específicos del Tool Center Point (TCP) del robot quirúrgico. El TCP es el punto de referencia principal en las intervenciones quirúrgicas, ya que marca el lugar exacto donde se realizan las acciones quirúrgicas. Lo que distingue esta funcionalidad es que se centra en traducir el movimiento del cirujano, captado a través del dispositivo háptico, en comandos de posición del TCP.

Esta funcionalidad asegura una comunicación eficaz entre el cirujano y el sistema robótico, permitiendo al cirujano tener un control en tiempo real sobre la ubicación del TCP, lo que es esencial para llevar a cabo operaciones quirúrgicas.

3.2.5. Modo de simulación y modo real

Esta funcionalidad proporciona a los cirujanos y operadores la capacidad de elegir entre dos modos operativos distintos: el «modo de simulación» y el «modo real». Cada uno de estos modos tiene un propósito específico y contribuye significativamente a la seguridad y eficacia de los procedimientos quirúrgicos.

- **Modo de simulación:** En este modo, la interfaz multimodal posibilita a los usuarios practicar y simular procedimientos quirúrgicos sin que el robot real ejecute movimientos físicos. En su lugar, el sistema genera representaciones virtuales de los movimientos planificados y sus consecuencias en un entorno de simulación. Este enfoque resulta de gran importancia para la formación de cirujanos, que acompañado también de una simulación del entorno quirúrgico, les permite adquirir experiencia y destreza en un entorno exento de riesgos para los pacientes. Además, resulta de suma importancia en entornos de investigación, donde el modo de simulación facilita la evaluación de funcionalidades y programas antes de su implementación en el robot real.
- **Modo real:** Cuando se selecciona el este modo, la interfaz multimodal permite al robot quirúrgico realizar movimientos físicos en el entorno real. Esta modalidad se utiliza durante los procedimientos quirúrgicos reales, donde la precisión y la seguridad son de suma importancia. El cirujano puede implementar las maniobras quirúrgicas planificadas en tiempo real, aprovechando la capacidad del robot para realizar movimientos milimétricos y eliminar temblores no deseados. Este modo es esencial para la ejecución efectiva de procedimientos quirúrgicos.

La capacidad de cambiar entre el «modo de simulación» y el «modo real» en la interfaz multimodal ofrece ventajas significativas. Permite una preparación más sólida antes de entrar al quirófano, lo que puede reducir los tiempos de procedimiento y mejorar la seguridad del paciente. Además, durante la cirugía real, el «modo real» garantiza una ejecución precisa y controlada de las maniobras quirúrgicas planificadas.

3.2.6. Configuración de parámetros

Esta funcionalidad es una parte esencial de la interfaz multimodal, desempeñando un papel vital en la personalización y adaptación de las capacidades del robot a las necesidades específicas de cada procedimiento quirúrgico. En este contexto, el cirujano se convierte en el usuario principal de la interfaz, ya que tiene la capacidad de configurar de manera detallada y personalizada una serie de parámetros clave en tiempo real. La personalización de estos parámetros es crucial, ya que estos datos específicos influirán directamente en el comportamiento y los movimientos subsiguientes del robot durante la cirugía.

Es posible definir los siguientes parámetros del modulo:

- **Modo de operación:** Este parámetro permite al cirujano elegir entre el «modo de simulación» y el «modo real». En modo de simulación, el robot opera virtualmente, lo que es útil para pruebas y entrenamiento. En contraste, el modo real implica que el robot realiza las tareas quirúrgicas físicamente. La alternancia entre

estos modos es vital para validar y perfeccionar las maniobras quirúrgicas antes de ejecutarlas en situaciones reales.

- **Longitud de la herramienta:** La longitud de la herramienta, medida desde el efector final (EF) del robot, es un parámetro crucial para definir el alcance y la geometría del robot en un procedimiento quirúrgico. La personalización de este valor es esencial para garantizar que el robot pueda acceder y operar con exactitud en las áreas de interés durante la cirugía. Esto se adapta de manera específica al tipo de herramienta utilizada y a los requisitos del procedimiento en curso.
- **Orientación de la herramienta:** La orientación de la herramienta se refiere a la dirección en la que apunta la herramienta quirúrgica en relación con el EF del robot. Este parámetro permite al cirujano ajustar la posición y la orientación de la herramienta con detalle para llevar a cabo tareas quirúrgicas específicas. Esto garantiza que la herramienta esté alineada de manera óptima para lograr una ejecución minuciosa.
- **Ubicación del punto de pivote:** Este parámetro crucial determina la posición del punto de pivote, punto de la posición de la incisión por la que se introduce la herramienta. Este valor, que debe estar en el rango de 0 a 1 para poder ser usado por la interfaz, define la ubicación precisa entre el efector final (EF) y el Tool Center Point (TCP). La combinación de este parámetro junto con la posición inicial del robot permite configurar con precisión la posición exacta del punto de fulcro, lo que resulta esencial para realizar movimientos quirúrgicos específicos y controlados alrededor de la zona de interés en una amplia gama de procedimientos quirúrgicos.
- **Dirección IP del robot:** La comunicación con los robots se establece normalmente a través de una dirección IP y mediante una interfaz de comunicaciones ethernet, por lo que la capacidad de ajustar este parámetro en tiempo real es fundamental en la interacción de la interfaz multimodal con el robot quirúrgico. Esto asegura que la interfaz esté configurada de manera específica y se comunique de manera efectiva con el robot quirúrgico durante las intervenciones.
- **Modo de trabajo:** Este parámetro define cómo opera el robot. El «modo libre» (free) permite una amplia movilidad del robot en el espacio, lo que resulta esencial para lograr una posición inicial precisa y adaptable antes de la cirugía, brindando al cirujano un mayor control y flexibilidad en el quirófano. Por otro lado, el «modo de pivotaje» (pivot) se utiliza para realizar movimientos concisos alrededor del punto de fulcro, lo que permite una ejecución precisa de maniobras quirúrgicas en torno a una referencia específica.

3.3. Diseño de la interfaz multimodal

Habiendo analizado las funcionalidades de la interfaz multimodal, se va a realizar en este apartado el diseño modular teórico de las funcionalidades del robot utilizadas para su uso quirúrgico.

En el contexto de la robótica quirúrgica, las funcionalidades se suelen integrar en módulos por varias razones fundamentales:

- Diseñar las funcionalidades del robot en forma de módulos permite una mayor modularidad y flexibilidad en el sistema de control. Cada módulo representa una funcionalidad específica y bien definida del robot, lo que facilita la implementación, el mantenimiento y la depuración del sistema en su conjunto. Además, esta modularidad hace posible una mejor reutilización del código, ya que cada módulo puede ser adaptado y utilizado en diferentes contextos y aplicaciones, lo que es esencial para proyectos de investigación en constante evolución.
- Además, la modularización de las funcionalidades del robot aumenta la legibilidad y la comprensión del código. Lo cual, es especialmente importante en proyectos de investigación científica, donde la transparencia y la documentación adecuada son fundamentales. Al dividir las funcionalidades en módulos separados, se establece una estructura más clara y organizada en el código fuente, facilitando tanto la revisión por parte de otros investigadores como la explicación detallada de cómo se desempeña cada funcionalidad.

Para brindar una representación más clara de estos módulos, se ha elaborado un esquema que describe la arquitectura funcional de la interfaz multimodal, la cual se detalla en la Figura 6.

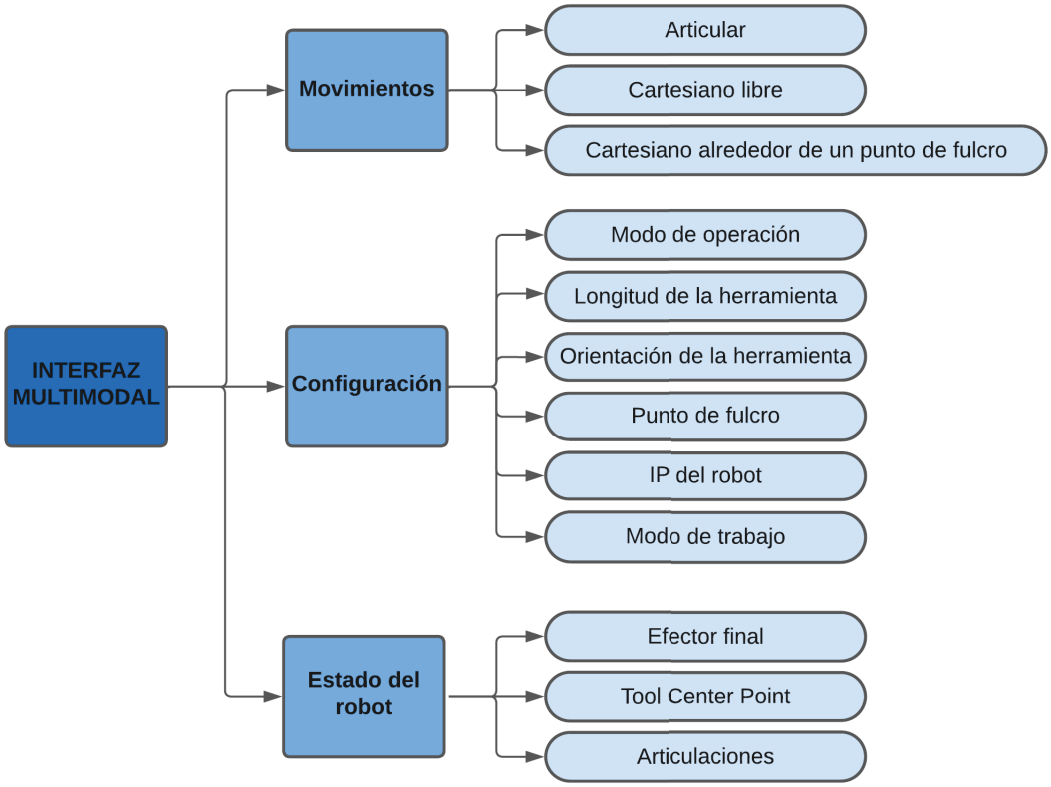


Figura 6: Arquitectura funcional de la interfaz multimodal.

Desde el cuadro principal que representa la interfaz multimodal, se desprenden tres categorías principales de funcionalidades representadas por flechas: «Movimientos», «Configuración» y «Estado del robot».

Bajo la categoría de «Movimientos», se muestran tres módulos importantes: «Movimiento articular», «Movimiento cartesiano libre» y «Movimiento cartesiano alrededor de un punto de fulcro», que conforman los movimientos básicos ya mencionados de un robot quirúrgicos.

En cuanto a la categoría «Configuración», se representan seis módulos con los seis parámetros clave ya comentados con anterioridad que pueden ser personalizados. Estos parámetros son esenciales para adaptar la interfaz a las necesidades específicas de cada procedimiento quirúrgico.

Por último, en la categoría «Estado del robot», se destacan tres módulos con los aspectos críticos que proporcionan información en tiempo real sobre el robot quirúrgico. Estos incluyen el «Efector Final», el «Tool Center Point», y las «Articulaciones» del robot, siendo todos ellos fundamentales para conocer la configuración actual del robot.

En los siguientes puntos, se aborda más en detalle los módulos con las funcionalidades de «Movimiento cartesiano cibre», «Movimiento cartesiano alrededor de un punto de fulcro» y el cálculo del «Tool Center Point (TCP)» en el estado del robot. Esto permitirá una comprensión más profunda de cómo se han implementado estas funcionalidades y cómo contribuyen a las capacidades de control.

3.3.1. Movimiento cartesiano libre

El módulo de movimiento cartesiano libre en la interfaz multimodal, representa un componente básico para lograr movimientos quirúrgicos controlados. Su función central radica en permitir al cirujano definir y transmitir la posición y orientación deseada del Tool Center Point (TCP) al robot. Es importante destacar que en la mayoría de los casos, los robots utilizan cuaternios para representar la orientación, por lo que a continuación se detalla el desarrollo para esta forma de representación. El proceso ilustrado en la Figura 7 consta de los siguientes pasos:

1. El proceso comienza con el cirujano suministrando a la interfaz la posición y la orientación deseadas del TCP. Inicialmente, esta orientación que se proporciona en forma de cuaternio ($q = (w, x, y, z)$) y que es una representación matemática compacta de la orientación espacial, es transformada por la interfaz en una matriz de rotación 3x3 para poder trabajar con ella con mayor facilidad en cálculos subsiguientes. Esta transformación se lleva a cabo mediante la siguiente ecuación:

$$Rm = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2zw & 2xz+2yw \\ 2xy+2zw & 1-2x^2-2z^2 & 2yz-2xw \\ 2xz-2yw & 2yz+2xw & 1-2x^2-2y^2 \end{bmatrix} \quad (1)$$

   Donde:

   - Rm: Matriz de rotación 3x3 obtenida a partir de la conversión de un cuaternio $q = (w, x, y, z)$. Esta matriz define la orientación de la herramienta del robot.

2. Posteriormente, se lleva a cabo el cálculo del vector que determina la dirección relativa desde el TCP hacia el EF. Este vector define cómo se extiende la herramienta quirúrgica desde el TCP hasta el EF. Corresponde al eje Z (negativo) relativo al TCP.

$$\overline{z} = Lh * [0, 0, -1] \quad (2)$$

   Donde:

   - Lh: Longitud de la herramienta empleada, en metros.
   - $\overline{z}$: Vector de dirección relativa desde el TCP hasta el EF. Este vector define cómo se extiende la herramienta quirúrgica desde el TCP hasta el EF. Corresponde al eje Z (negativo) relativo al TCP.

3. La matriz de rotación obtenida previamente en la ecuación 1 se emplea para transformar el vector $\overline{z}$ al sistema de coordenadas global del robot de la siguiente manera:

$$\overline{z_t} = Rm * \overline{z}' \quad (3)$$

   Donde:

   - $\overline{z}$: Vector de dirección relativa desde el TCP hasta el EF.
   - Rm: Matriz de rotación obtenida a partir del cuaternio que define la orientación del TCP del robot.
   - $\overline{z_t}$: Vector $\overline{z}$ transformado al sistema de coordenadas global del robot.

4. Simultáneamente, se extrae la posición del TCP de los datos proporcionados por el cirujano, comprendiendo las coordenadas espaciales del punto de referencia desde donde se extiende la herramienta quirúrgica. Este conocimiento es esencial para calcular la posición final del EF. Con todos estos datos en mano, se lleva cabo el cálculo de la posición exacta del EF. Esto implica agregar el vector de dirección relativa transformado al punto de referencia del TCP. El resultado es la nueva ubicación en la que el EF debe posicionarse para cumplir con las especificaciones del cirujano.

$$P_{ef} = P_{tcp} + \overline{z_t}' \quad (4)$$

   Donde:

- P_{tcp}: Posición deseada de la punta de la herramienta, en coordenadas cartesianas con respecto al sistema de referencia de la base.
- P_{ef}: Posición del efector final del robot, en coordenadas cartesianas con respecto al sistema de referencia de la base.
- $\overline{z_t}$: Vector $\overline{z}$ transformado al sistema de coordenadas global del robot.

5. Finalmente, se actualizan los valores de posición y orientación del EF con los resultados de los cálculos. Estos valores actualizados se transmiten al robot mediante mensajes de posición, lo que permite al robot ajustar su posición y orientación para coincidir con las indicaciones del cirujano.

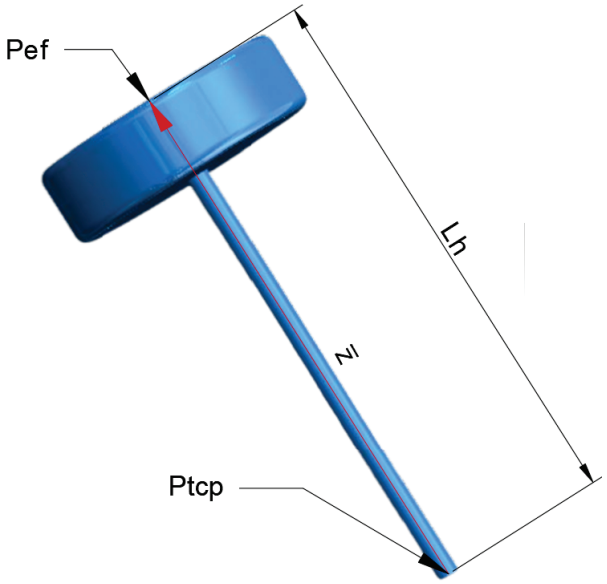


Figura 7: Obtención de P_{ef} a partir de P_{tcp}.

La importancia de estas transformaciones y cálculos matemáticos radica en que garantizan que el robot pueda llevar a cabo movimientos quirúrgicos específicos de acuerdo con las indicaciones del cirujano. La interfaz desempeña un gran papel al llevar a cabo estas operaciones de manera eficiente y concisa, siendo esencial para el éxito de los procedimientos quirúrgicos.

3.3.2. Movimiento cartesiano alrededor de un punto de fulcro

El módulo de movimiento cartesiano alrededor de un punto de fulcro es una parte crítica de la interfaz de control de un robot quirúrgico. A continuación se detallan los pasos seguidos en este modulo:

Estimación del punto de fulcro

El módulo comienza recibiendo comandos de posición y orientación a través de mensajes. Estos comandos son proporcionados por el cirujano y representan la ubicación deseada del Tool Center Point (TCP) del robot quirúrgico en el espacio 3D. Al igual que ya se dijo con anterioridad, es importante destacar que en la mayoría de los casos, los robots utilizan cuaternios para representar la orientación, por lo que a continuación se detalla el desarrollo para esta forma de representación.

Del primer mensaje, se obtendrá la posición y orientación del efector final del robot (EF) de la misma manera que ya se explicó para el movimiento cartesiano libre. Con esta información ya se estará en condiciones de determinar el punto de fulcro. La determinación de este punto resultará fundamental para el desarrollo posterior de la tarea, ya que su identificación es un requisito previo para cualquier movimiento del robot. En términos generales, se seguirá las siguientes ecuaciones como método para su cálculo:

$$\overline{z} = Fi * Lh * [0, 0, 1] \tag{5}$$

$$\overline{z_t} = Rm * \overline{z}' \tag{6}$$

$$P_f = P_{ef} + \overline{z_t}' \tag{7}$$

Donde:

- P_f: Posición del punto de fulcro, en coordenadas cartesianas con respecto al sistema de referencia de la base.
- P_{ef}: Posición del efector final del robot, en coordenadas cartesianas con respecto al sistema de referencia de la base.
- Fi: Parámetro que indica en qué punto de la longitud de la herramienta se encuentra el punto de fulcro. Debe estar entre 0 y 1.
- Lh: Longitud de la herramienta empleada, en metros.
- Rm: Matriz de rotación obtenida a partir del cuaternio que define la orientación del efector final del robot.
- $\overline{z}$: Vector de dirección relativa desde el EF hasta el punto de fulcro. Este vector define cómo se extiende la herramienta quirúrgica desde el EF hacia el punto de fulcro. Corresponde al eje Z (positivo) relativo al efector final del robot.
- $\overline{z_t}$: Vector $\overline{z}$ transformado al sistema de coordenadas global del robot.

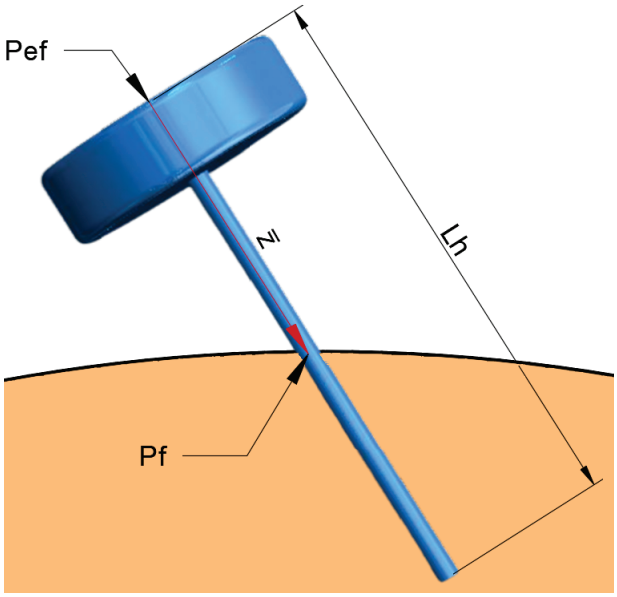


Figura 8: Estimación del punto de fulcro P_f.

En la Figura 8 se ilustra un ejemplo de posición de este movimiento junto con todas las variables relevantes de las ecuaciones anteriores. A partir de la información sobre la posición del efector final, la orientación y la distancia relativa en relación con la longitud de la herramienta, es posible realizar el cálculo que nos proporcionará la ubicación del punto de fulcro. Esta ubicación será registrada en una variable expresada en coordenadas cartesianas.

Movimiento alrededor del punto de fulcro

Habiendo establecido el punto de fulcro, se procede a calcular la posición y orientación requerida para que el robot pueda alcanzar una nueva ubicación específica en la punta de la herramienta. Este procedimiento se repetirá cada vez que se desee realizar un nuevo movimiento de la punta. No obstante, es importante destacar que el cálculo del punto de fulcro solo se llevará a cabo en la fase inicial, ya que se mantendrá constante a lo largo de todo el proceso.

El proceso implementado para alcanzar el propósito representado en la Figura 9 se describe de la siguiente manera:

1. Se recibe el incremento de posición que deseamos aplicar en cada una de las direcciones y se suma a la posición calculada de la punta de la herramienta:

$$P_{tn} = P_{tcp} + \Delta P \tag{8}$$

Donde:

- P_{tn}: Nueva posición de la punta de la herramienta tras incremento, en coordenadas cartesianas.
- P_{tcp}: Posición de la punta de la herramienta, en coordenadas cartesianas.
- ΔP: Incremento de posición.

2. Se determina el vector que conecta la recién calculada nueva posición de la punta (P_{tn}) con el punto de fulcro (P_f). Este vector resultante se convierte en la nueva dirección del eje Z de nuestra herramienta:

$$z_n = P_f - P_{tn} \tag{9}$$

Donde:

- z_n: Vector que indica la nueva dirección de la herramienta.
- P_f: Punto de fulcro, en coordenadas cartesianas.
- P_{tn}: Nueva posición de la punta de la herramienta tras incremento, en coordenadas cartesianas.

3. Se determina la distancia desde el punto de fulcro hasta la que será la nueva posición del efector final (P_n. Teniendo en cuenta la longitud total de la herramienta (Lh) y la distancia existente desde el punto de fulcro hasta su punta, obtenemos lo siguiente:

$$\rho = Lh - ||Pz_n|| \tag{10}$$

Donde:

- ρ: Distancia desde el punto de fulcro hasta la que será la nueva posición del efector final, en metros.

- Lh: Longitud de la herramienta, en metros.
- $||\overline{z_n}||$: Módulo del vector de dirección z_n, en metros.

4. Habiendo determinado esta distancia, se está en condiciones de calcular la nueva posición del efector final. Al sumar la distancia ρ en la dirección correcta desde el punto de fulcro, se obtiene:

$$P_n = P_f + \rho * z_n \tag{11}$$

Donde:

- P_n: Nueva posición del efector final, en coordenadas cartesianas.
- P_f: Punto de fulcro, en coordenadas cartesianas.
- ρ: Distancia desde el punto de fulcro hasta la que será la nueva posición del efector final, en metros.
- z_n: Vector que indica la nueva dirección de la herramienta.

5. Para llevar a cabo el cálculo de la nueva orientación de la herramienta, se determinará primero el vector unitario en la dirección relativa al eje Z (positivo) de la herramienta.

$$\mathbf{z}_{nn} = \frac{-\mathbf{z}_n}{\|\mathbf{z}_n\|} \tag{12}$$

Donde:

- $\mathbf{z}_{nn}$: Vector unitario de dirección relativa desde el EF hasta TCP. Corresponde al eje Z (positivo) relativo al efector final del robot.
- $\mathbf{z}_n$: Vector que indica la nueva dirección de la herramienta.
- $\|\mathbf{z}_n\|$: El módulo del vector $\mathbf{z}_n$.

6. Posteriormente se calculan dos vectores unitarios perpendiculares $\mathbf{x}_{nn}$ y $\mathbf{y}_{nn}$ en un espacio tridimensional. Primero, se obtiene un vector $\mathbf{x}_{nn}$ que es perpendicular al eje Z estándar y al vector $\mathbf{z}_{nn}$, y luego se normaliza. Después, se calcula un segundo vector $\mathbf{y}_{nn}$ que es perpendicular a $\mathbf{z}_{nn}$ y $\mathbf{x}_{nn}$, también normalizado. Estos vectores se utilizan para definir un sistema de coordenadas ortogonales.

$$\mathbf{x}_{nn} = \frac{\mathbf{e}_z \times \mathbf{z}_{nn}}{\|\mathbf{e}_z \times \mathbf{z}_{nn}\|} \tag{13}$$

$$\mathbf{y}_{nn} = \frac{\mathbf{z}_{nn} \times \mathbf{x}_{nn}}{\|\mathbf{z}_{nn} \times \mathbf{x}_{nn}\|} \tag{14}$$

Donde:

- $\mathbf{z}_{nn}$: Vector unitario de dirección relativa desde el EF hasta TCP. Corresponde al eje Z (positivo) relativo al efector final del robot.
- $\mathbf{x}_{nn}$: Vector unitario perpendicular al eje Z estándar y a $\mathbf{z}_{nn}$.
- $\mathbf{y}_{nn}$: Vector unitario perpendicular a los vectores unitarios $\mathbf{z}_{nn}$ y $\mathbf{x}_{nn}$.
- $\mathbf{e}_z$: Vector que representa el eje Z estándar, es decir, [0,0,1].
- $\|\mathbf{e}_z \times \mathbf{z}_{nn}\|$: representa la norma del resultado del producto cruz entre $\mathbf{e}_z$ y $\mathbf{z}_{nn}$.
- $\|\mathbf{z}_{nn} \times \mathbf{x}_{nn}\|$: representa la norma del resultado del producto cruz entre $\mathbf{z}_{nn}$ y $\mathbf{x}_{nn}$.

7. A continuación se crea una matriz de rotación tridimensional al combinar los vectores unitarios $\mathbf{x}_{nn}$, $\mathbf{y}_{nn}$, y $\mathbf{z}_{nn}$ como sus columnas.

$$M = \begin{bmatrix} \mathbf{x}_{nn}[0] & \mathbf{y}_{nn}[0] & \mathbf{z}_{nn}[0] \\ \mathbf{x}_{nn}[1] & \mathbf{y}_{nn}[1] & \mathbf{z}_{nn}[1] \\ \mathbf{x}_{nn}[2] & \mathbf{y}_{nn}[2] & \mathbf{z}_{nn}[2] \end{bmatrix} \tag{15}$$

Donde:

- M: Matriz de rotación 3x3 generada.

8. Por último, para poder garantizar que el robot utilizado interprete correctamente la orientación requerida, se convierte la matriz de rotación M en cuaternio para poder mandarla al robot junto a la nueva posición P_n del efector final.

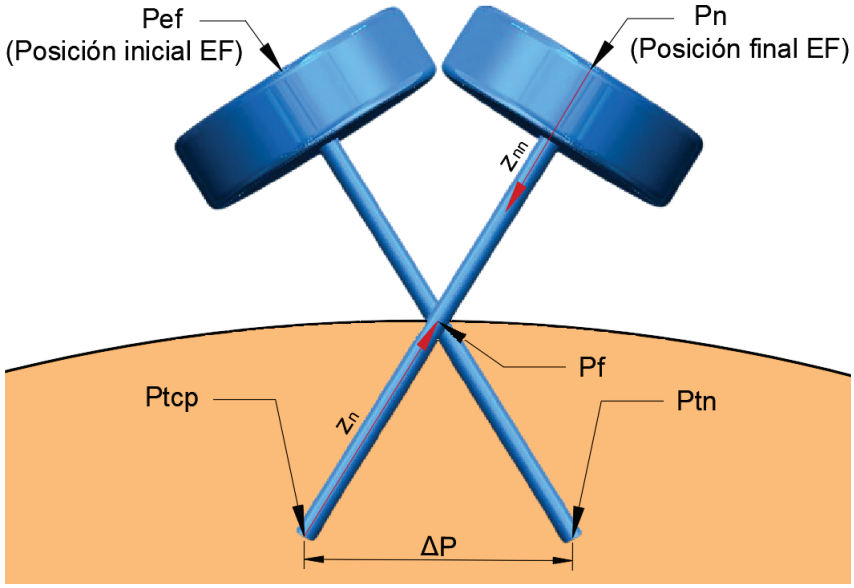


Figura 9: Posición final del EF y TCP tras realizar el movimiento desde la posición inicial.

3.3.3. Estado del robot

Este modulo se encarga de proporcionar al cirujano una comprensión completa del estado del robot quirúrgico durante el procedimiento, lo cual es esencial para garantizar la concreción y seguridad en la cirugía. Este módulo se enfoca en la retroalimentación de dos aspectos críticos: las posiciones articulares de las 7 articulaciones del robot y la posición y orientación del efector final (EF) del robot y el Tool Center Point (TCP). La posición del EF y del TCP se expresa en coordenadas cartesianas mientras que su orientación se representa a través de un cuaternio. A su vez, las posiciones articulares representan los ángulos (en radianes) de cada articulación.

Para las posiciones articulares, el módulo recopila información sobre las posiciones de las articulaciones del robot. Esto se logra mediante la recepción de datos específicos que indican los ángulos o posiciones de cada una de las articulaciones. Estos datos se estructuran en un mensaje adecuado y se transmiten al cirujano en tiempo real. Esta información permite el cirujano conocer y supervisar la configuración actual del robot, facilitando el control y la toma de decisiones durante el procedimiento.

Al igual que la información articular, la retroalimentación del EF, se obtiene directamente del robot y se transmite al cirujano en tiempo real. Sin embargo, en lo que respecta a la retroalimentación y transmisión de la posición

y orientación del TCP de la herramienta quirúrgica, el proceso es más elaborado. A continuación se presentan los pasos seguidos en este proceso ilustrado en la Figura 10:

1. Primeramente, se recibe la información que describe la posición y orientación del EF del robot. Esto se logra mediante el uso de cuaternios. Sin embargo, para que el robot comprenda y utilice esta información de manera efectiva, es necesario realizar la conversión ya comentada anteriormente en la ecuación 1 de cuaternio $q = (w, x, y, z)$ a una matriz de rotación. Esta matriz de rotación describe la orientación precisa de la herramienta quirúrgica.

2. Otro cálculo a realizar importante implica la determinación del vector de dirección relativa desde el EF hasta el TCP. Este vector define cómo se extiende la herramienta quirúrgica desde el EF hacia el TCP y es fundamental para determinar la posición del TCP.

$$\overline{z} = Lh * [0, 0, 1] \tag{16}$$

   Donde:

   - Lh: Longitud de la herramienta empleada, en metros.
   - $\overline{z}$: Vector de dirección relativa desde el EF hasta el TCP. Este vector define cómo se extiende la herramienta quirúrgica desde el EF hasta el TCP. Corresponde al eje Z (positivo) relativo al EF.

3. La matriz de rotación calculada previamente se utiliza para transformar este vector en el sistema de coordenadas global del robot.

$$\overline{z_t} = Rm * \overline{z}' \tag{17}$$

   Donde:

   - $\overline{z}$: Vector de dirección relativa desde el EF hasta el TCP.
   - Rm: Matriz de rotación obtenida a partir del cuaternio que define la orientación del EF del robot.
   - $\overline{z_t}$: Vector $\overline{z}$ transformado al sistema de coordenadas global del robot.

4. Una vez que se conoce la dirección relativa transformada y la posición del EF, se realiza el cálculo para determinar la posición exacta del TCP. Esto consiste en agregar el vector de dirección relativa a la posición del EF, lo que resulta en la ubicación exacta del TCP en el espacio tridimensional. Esta posición se expresa en coordenadas cartesianas y se utiliza para controlar la posición del TCP durante los procedimientos quirúrgicos.

$$P_{tcp} = P_{ef} + \overline{z_t}' \tag{18}$$

   Donde:

   - P_{tcp}: Posición de la punta de la herramienta, en coordenadas cartesianas con respecto al sistema de referencia de la base.
   - P_{ef}: Posición del efector final del robot, en coordenadas cartesianas con respecto al sistema de referencia de la base.
   - $\overline{z_t}$: Vector $\overline{z}$ transformado al sistema de coordenadas global del robot.

5. Finalmente, todos estos datos se actualizan y se transmiten al cirujano en tiempo real. La posición y orientación del TCP se comunican, lo que permite al cirujano tener una visión completa de la ubicación exacta de la herramienta quirúrgica en el espacio quirúrgico. Esto es esencial para llevar a cabo procedimientos quirúrgicos rigurosos y fiables, ya que el cirujano puede supervisar y ajustar continuamente la posición y orientación de la herramienta durante la cirugía.

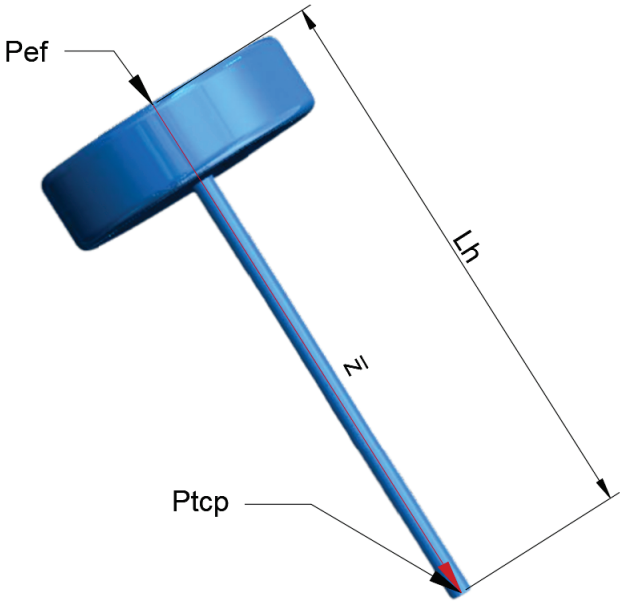


Figura 10: Obtención de P_{tcp} a partir de P_{ef}.

4. Implementación

4.1. Robot Kuka LBR iiwa 7 R800

El robot empleado en este trabajo es el KUKA LBR iiwa 7 R800, mostrado en la Figura 11. El LBR iiwa 7 R800 es un robot de la serie LBR iiwa de KUKA Roboter GmbH, que destaca por ser el primer robot fabricado en serie con capacidades sensoriales y diseñado para la colaboración entre personas y robots. El término «LBR» hace referencia a «robot de estructura liviana», mientras que «iiwa» significa «intelligent industrial work assistant». El número «7» indica su capacidad de carga de 7 kg, y el «R800» representa el alcance máximo de la zona de trabajo en milímetros.

Figura 11: Robot KUKA LBR iiwa con herramienta [24].

4.1.1. Descripción y análisis del robot

El robot LBR iiwa se clasifica como un robot ligero y se trata de un brazo robótico articulado con 7 ejes. A su vez, cada articulación se encuentra equipada con un sensor de posición en el lado de entrada, sensores de temperatura y sensores de par en el lado de salida. Esto permite que el robot sea operado con control de posición e impedancia, evitando la sobrecarga térmica gracias a los mencionados sensores de temperatura.

Tanto sus unidades motoras como todos los cables que transportan corriente eléctrica están protegidos por placas de cubierta. Además de ello, cada eje se encuentra protegido mediante sensores de rango de eje y puede ajustarse mediante sensores internos. Los cables de alimentación y control de los motores de todos los ejes (A1 a A7), ilustrados en la Figura 12, conforman la instalación eléctrica del robot. Todo este cableado se encuentra enrutado en el interior del robot y cuenta con conexiones macho y hembra.

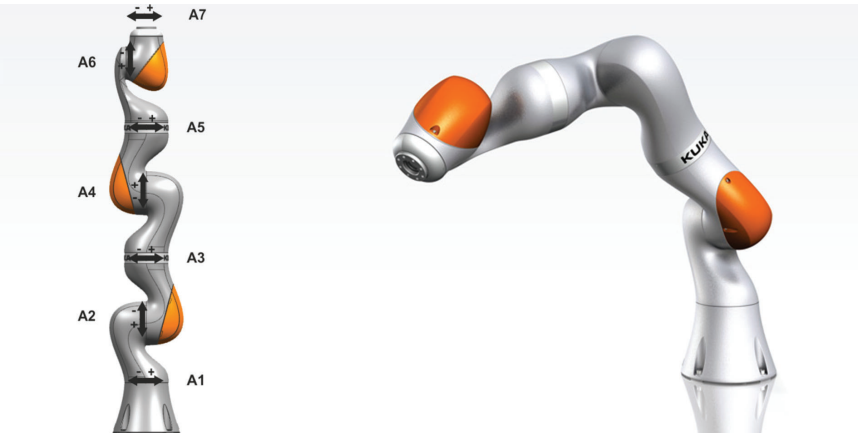


Figura 12: Ejes y articulaciones del robot [25].

El brazo robótico está equipado de los siguientes componentes principales:

- **Bastidor base:** Se trata de la base del robot y alberga la interfaz A1 en su parte trasera, utilizada para las conexiones de cables entre el robot, el controlador y el sistema suministrador de energía.
- **Módulos de unión:** Estos módulos son estructuras de aluminio que albergan las unidades de accionamiento, permitiendo así la conexión entre ellas a través de dichas estructuras.
- **Muñeca en línea:** El brazo robótico está dotado de una muñeca en línea de 2 ejes (A6 y A7), ambos motorizados.

Las principales ventajas de la serie de robots LBR iiwa incluyen un tiempo de reacción mínimo, capacidad de aprendizaje, sensibilidad y autonomía. Es un robot ligero recomendado para tareas de montaje sensibles, rápidas y exactas, en las que se pueden eliminar las vallas protectoras del espacio de trabajo, facilitando así la colaboración entre personas y robots, como se oberva en la Figura 13.

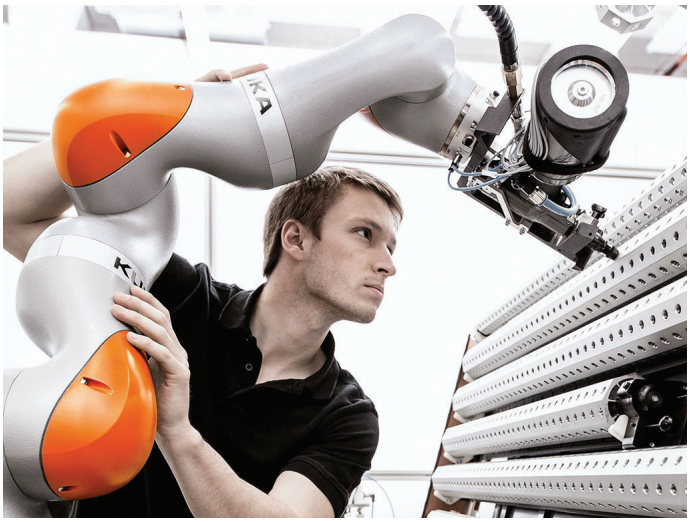


Figura 13: Trabajo en un entorno colaborativo [24].

El LBR iiwa se destaca por su amplio empleo como colaborador en entornos de producción, gracias a su destacada versatilidad para trabajar en conjunto con el operador. Esto lo convierte en la elección óptima para tareas que demandan una exactitud meticulosa y que se repiten frecuentemente a lo largo del día, obteniendo una mejora significativa en la productividad.

En el manual de especificaciones proporcionado por el fabricante, se puede encontrar información detallada sobre dimensiones, peso, restricciones de los actuadores, condiciones de servicio y otros aspectos relevantes. Esta compilación de datos se ilustra en el Anexo A.

4.1.2. Vías de comunicación con el robot

La interacción y control efectivo de un robot en el contexto de la cirugía robótica son aspectos fundamentales para el éxito de las intervenciones quirúrgicas. La capacidad de programar, operar y comunicarse con el robot KUKA iiwa LBR es esencial para garantizar el acierto, seguridad y eficacia en el quirófano. En este apartado, exploraremos las diversas vías de comunicación disponibles para interactuar con el robot KUKA iiwa LBR, lo que incluye la programación a través de Sunrise Workbench, el uso del SmartPad como interfaz de usuario física y la comunicación a través del sistema operativo de robots (ROS). Cada una de estas vías proporciona herramientas y recursos específicos para adaptarse a las necesidades de los operadores, asegurando un control riguroso y colaboración cercana en el entorno quirúrgico. A continuación, profundizaremos en cada uno de estos tres métodos de comunicación:

4.1.2.1 Programación con Sunrise Workbench

El robot KUKA iiwa LBR con manipulador redundante se programa utilizando la plataforma Sunrise Workbench de KUKA, junto con sus API Java.

Esta plataforma utiliza el software de sistema KUKA Sunrise mostrado en la Figura 14, proporcionando un entorno de desarrollo integral que se basa en la programación Java. Dicha plataforma está diseñada específicamente para la creación y personalización de aplicaciones robóticas destinadas a la serie de robots KUKA LBR iiwa y otros productos Mobility de KUKA, ofreciendo todas las funciones necesarias para el servicio de robots de estructura ligera.

Las ventajas de esta plataforma son diversas e incluyen:

- Interfaz de usuario ergonómica
- Editor de programas con muchas funciones eficientes para mayor comodidad
- Programación orientada a objetos con Java
- Rápida puesta en servicio
- Cómodo diagnóstico
- Manual del usuario integrado
- Funciones de depuración profesional

Gracias a todas estas características, trabajar con el robot se facilita en gran medida, incluso si no se tienen conocimientos previos de programación.

Figura 14: Plataforma Sunrise Workbench de KUKA [24].

Esta aplicación generará, una vez terminada la programación del robot, un proyecto Sunrise. Estos proyectos Sunrise, que pueden contener más de una aplicación robótica, se pueden sincronizar con el gabinete del robot y, posteriormente, ejecutarse y controlarse desde el SmartPad. Dichos elementos y sus conexiones se pueden visualizar en la Figura 15.

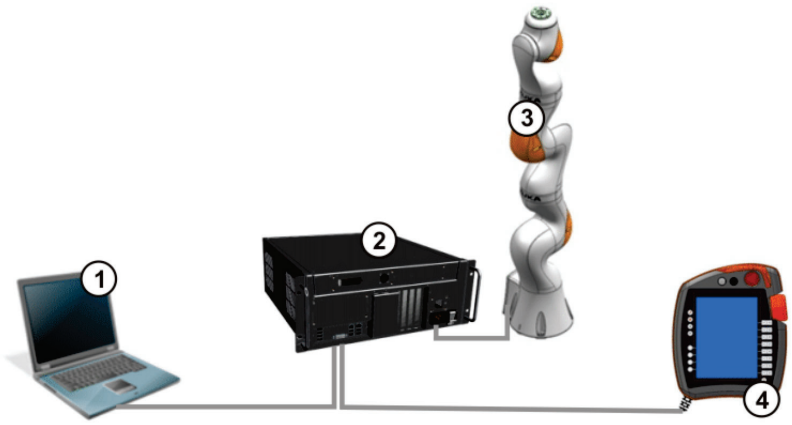


Figura 15: 1- PC con plataforma Sunrise Workbench. 2- Gabinete del robot. 3- Robot LBR iiwa. 4- SmartPAD [26]

4.1.2.2 Interfaz SmartPad

El SmartPad, ilustrado en la Figura 16, es una interfaz de usuario física que permite interactuar y controlar el robot KUKA iiwa LBR de manera intuitiva. Sus características esenciales son:

- **Control manual:** El SmartPad ofrece un control manual del robot, lo que permite a los operadores guiar físicamente al robot o realizar movimientos exactos utilizando esta interfaz.
- **Visualización de aplicaciones:** Además de las funciones de control, el SmartPad también puede mostrar aplicaciones desarrolladas en Sunrise Workbench. Los operadores pueden seleccionar y ejecutar estas aplicaciones directamente desde el SmartPad.
- **Diagnóstico:** El SmartPad proporciona información de diagnóstico y estado del robot, lo que facilita la supervisión y el mantenimiento del sistema.
- **Interfaz de usuario intuitiva:** El SmartPad se ha diseñado para ser fácil de usar, incluso para aquellos que no tienen experiencia previa en programación de robots, ofrece una interfaz gráfica que simplifica la interacción con el robot.

Figura 16: KUKA smartPAD [24].

4.1.2.3 Comunicación a través de ROS

Además de los tipos de comunicación ya expuestos, otra forma de comunicación con el robot es a través de ROS. Para ello, es posible utilizar un paquete de software llamado (iiwa_stack) de ROS para realizar la comunicación con el robot. Dentro de dicho paquete de software, se puede encontrar una aplicación robótica a través de la cual se establece una conexión con las máquinas conectadas mediante Ethernet al armario del robot usando ROS. Las máquinas que tengan instalado ROS podrán enviar y recibir mensajes ROS hacia y desde la mencionada aplicación robótica. Se utilizan mensajes personalizados ROS (iiwa_msgs) disponibles en una distribución ROS estándar, diseñados para funcionar con el brazo robótico KUKA [27].

Un usuario puede manipular los mensajes recibidos del robot y enviar otros nuevos como comandos dentro del código C++ o Python, aprovechando todas las funcionalidades de ROS.

Los pasos seguidos para la conexión y la posterior comunicación con el robot son los siguientes:

- Conexión a través de cable Ethernet entre el puerto X66 del gabinete SUNRISE del robot y la máquina ROS.
- Por defecto, la dirección IP del gabinete SUNRISE del robot es la configuración de la IP en la que se encuentre el robot. Se debe configurar la interfaz de red en la máquina ROS para que esté en la misma subred (como por ejemplo, 172.31.1.150).
- Clonación, compilación y configuración del paquete iiwa_stack en la máquina ROS.
- Clonación y configuración de un proyecto Sunrise con iiwa_stack en el gabinete SUNRISE.

Para lograr una comunicación efectiva entre la controladora del robot y un PC externo, se requiere una configuración específica. En la controladora del robot, es necesario que se ejecute la aplicación ROSSmartServo. Esto permite que el robot esté listo para comunicarse con el sistema ROS en el PC externo.

Por otro lado, en el PC externo, es necesario iniciar el «roscore», que es el núcleo de ROS, para habilitar la comunicación con el robot, representada en la Figura 17. Además, se debe lanzar la aplicación de usuario que se utilizará para controlar el robot y recopilar datos. Esta configuración establece las bases para la interacción entre el sistema ROS y la controladora del robot.

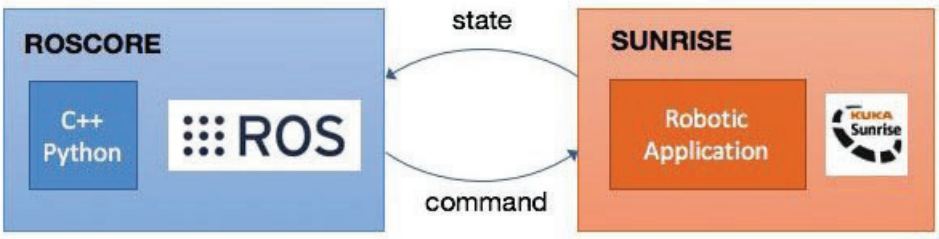


Figura 17: Arquitectura del sistema [27].

Adicionalmente, es relevante señalar que el paquete iiwa_stack de comunicación ROS, ofrece un modo de funcionamiento simulado, lo que significa que los usuarios tienen la opción de trabajar con un robot simulado en lugar de un robot físico. En este sentido, el robot KUKA LBR iiwa cuenta con su propio simulador Gazebo, que es un entorno de simulación de código abierto ampliamente utilizado en robótica. Este simulador, exhibido en la Figura 18, proporciona un entorno virtual realista y altamente personalizable con características significativas, que incluyen una física realista, integración con ROS, una amplia librería de modelos predefinidos y la capacidad de realizar simulaciones en tiempo real.

Así, la comunicación a través de ROS se convierte en la elección más lógica sobre la que se desarrolla este trabajo, gracias a su capacidad para proporcionar un entorno estandarizado y robusto, facilitando la interacción entre la controladora del robot y el PC externo, así como ofreciendo la flexibilidad necesaria para el desarrollo y control del robot.

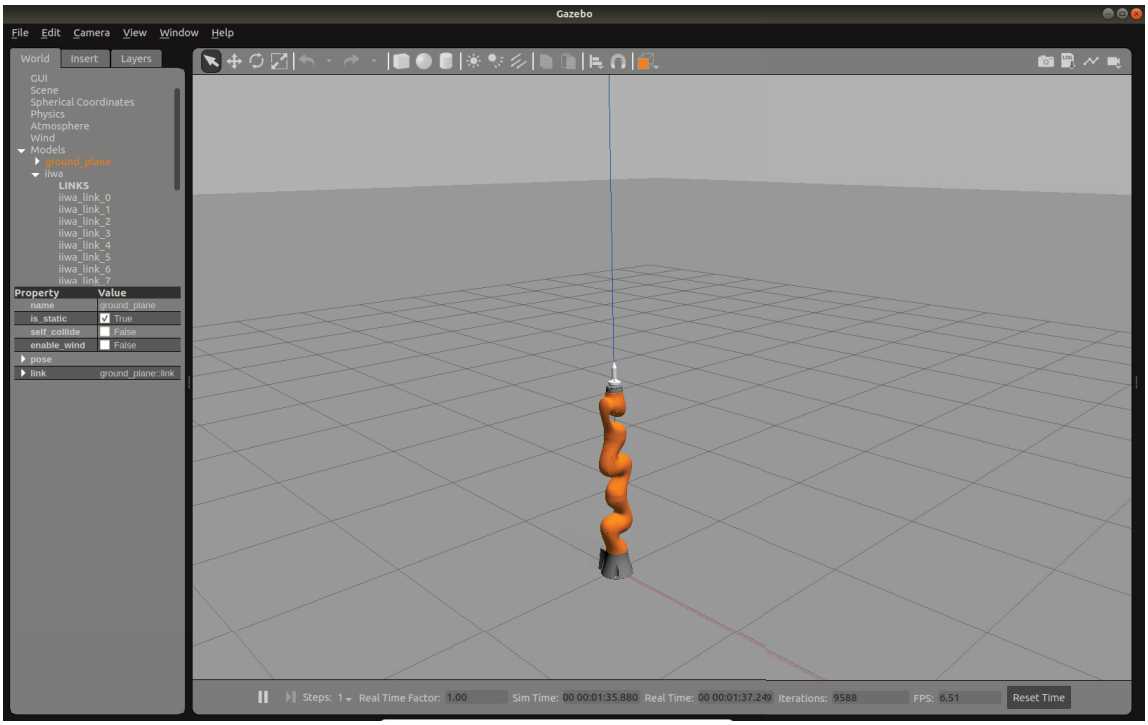


Figura 18: Simulador Gazebo.

Para obtener información más detallada sobre ROS y y su importancia en el funcionamiento del sistema, se puede consultar el próximo apartado 4.2 de la documentación, donde se proporcionará una explicación exhaustiva.

4.2. Introducción a ROS

Robot Operating System, también conocido como ROS (Figura 19), se ha convertido en una piedra angular en los campos de la robótica y la automatización en los últimos años. ROS no se considera un sistema operativo en el sentido tradicional, sino más bien una plataforma de software flexible y de código abierto expresamente diseñada para simplificar el desarrollo y control de robots. Su popularidad se debe a su enfoque modular, su comunidad activa de desarrolladores y su capacidad para integrar una amplia variedad de hardware y sensores.

Figura 19: Robot Operating System.

Una de las características fundamentales de ROS es su enfoque en la modularidad. ROS se compone de una colección de paquetes y librerías que se pueden utilizar para construir sistemas robóticos complejos. Cada paquete se centra en una tarea específica, como la percepción, el control de movimiento, la planificación de trayectorias o la comunicación entre componentes del robot. Esto permite a los desarrolladores utilizar solo los módulos relevantes para su aplicación particular, simplificando enormemente el desarrollo y la adaptación de sistemas robóticos.

Otra característica distintiva de ROS es su sistema de comunicación basado en mensajes. Los nodos (componentes de software) en un sistema ROS pueden intercambiar información a través de mensajes estandarizados. Esto facilita enormemente la comunicación entre diferentes partes de un robot o entre múltiples robots en un entorno

colaborativo. Además, ROS proporciona herramientas para registrar y reproducir datos de sensores, siendo ésto esencial para el desarrollo y la depuración de aplicaciones robóticas.

ROS es conocido también por su capacidad para integrar una amplia gama de hardware y sensores. Esto es particularmente relevante en el contexto de la robótica quirúrgica, donde la retroalimentación sensorial es crítica. Los desarrolladores pueden utilizar controladores ROS existentes o desarrollar nuevos controladores para integrar hardware específico en su sistema robótico.

Una de las claves detrás del éxito de ROS es su gran comunidad activa de desarrolladores. La comunidad contribuye con una gran cantidad de paquetes y recursos disponibles de código abierto, lo que acelera el desarrollo de proyectos robóticos. Además, ROS ofrece una amplia documentación y tutoriales que facilitan el aprendizaje y la implementación de sus características.

A continuación se van a detallar y definir los componentes principales de ROS, los cuales han sido utilizados en este proyecto.

4.2.1. Master

En el marco de ROS, el término «master» hace referencia a un componente central y esencial dentro de la arquitectura de ROS. El master actúa como el coordinador principal en un sistema ROS, desempeñando un papel fundamental en la gestión de la comunicación entre los diversos nodos que componen un sistema robótico. Las principales funciones y características del master son:

- **Coordinación de nodos:** Es el responsable de la coordinación y el seguimiento de cada uno de los nodos dentro de un sistema ROS. Cada nodo se comunica con el master para registrarse y anunciar los servicios que ofrece, además de comunicar los topics a los que se encuentra suscrito, como se ejemplifica en la Figura 20. Esto permite que los nodos encuentren y se comuniquen entre sí de manera eficiente.

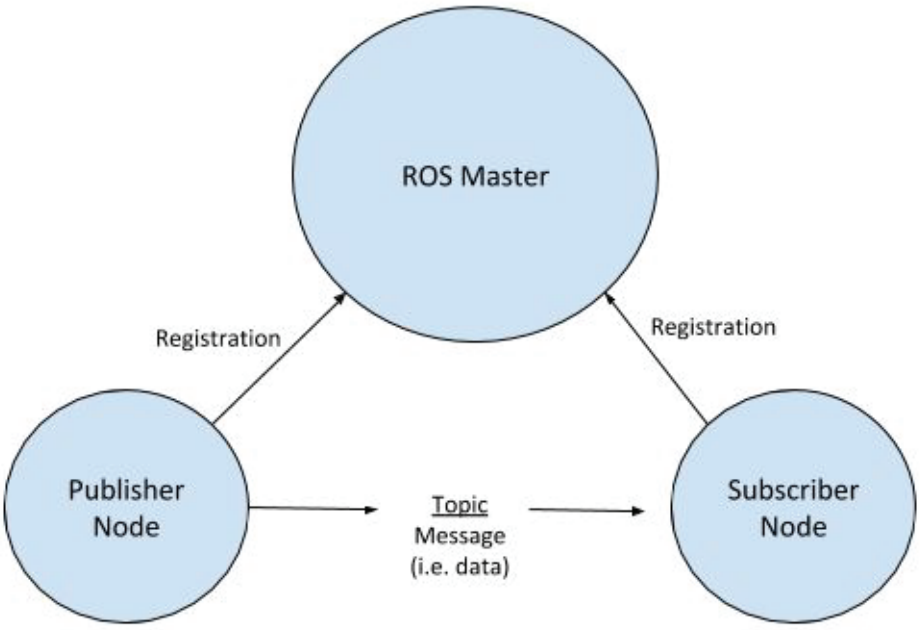


Figura 20: Registro y comunicación entre nodos [28].

- **Publicación y suscripción de los topics:** Una de las destacadas características de ROS es su sistema de publicación y suscripción de topics. El master realiza un papel esencial dentro de este proceso, ya que registra

y rastrea la información sobre los topics disponibles y los nodos que desean suscribirse a ellos. En el momento en el que un nodo desea publicar datos en un topic en concreto, el master facilita la conexión entre el nodo publicador y los nodos suscriptores interesados en dichos datos.

- **Sistema de nombres:** ROS utiliza un sistema jerárquico de nombres para organizar los nodos, topics y otros diversos recursos dentro del sistema. El master es el encargado de la resolución de nombres, traduciendo los nombres relativos de los recursos a nombres absolutos. Esto evita conflictos de nombres y permite que los nodos se comuniquen de manera efectiva y sin ninguna ambigüedad.
- **Servicios:** Además de la gestión de los topics, el master también en el encargado de los servicios en ROS. Los servicios permiten que los nodos realicen solicitudes y reciban respuestas de otros nodos. El master rastrea la información sobre los servicios disponibles y facilita la conexión entre los nodos que ofrecen y solicitan servicios.
- **Registro de gráficos de los nodos:** El master mantiene un registro de gráficos que muestra las relaciones entre los nodos existentes en el sistema. Esto facilita la visualización y la depuración, haciendo que los desarrolladores puedan comprender rápidamente cómo se lleva a cabo la comunicación de los nodos en un sistema complejo.
- **Tolerancia a fallos:** El master se ha diseñado para ser robusto y tolerante a fallos. En caso de que falle, ROS proporciona mecanismos para la recuperación automática o manual, garantizando la continuidad de la comunicación en el sistema.

4.2.2. Topics

Los «topics», en ROS, son un componente esencial que facilita la comunicación entre los diversos nodos que forman parte de un sistema robótico. Los topics se basan en un modelo de publicación y suscripción, donde un nodo publica datos en un topic y otro/s nodo/s pueden suscribirse a ese topic para recibir y procesar los datos. Las características y funciones principales de los topics son:

- **Publicación y suscripción:** Los topics hacen posible la comunicación asíncrona entre los nodos, es decir, que un nodo pueda publicar datos en un topic sin necesidad de conocer los nodos que están suscritos a ese topic. Esto facilita la modularidad y la reutilización del código desarrollado, ya que los nodos pueden comunicarse de manera eficiente y sin acoplamiento directo.
- **Interfaz estandarizada:** Cada topic tiene una interfaz estandarizada, la cual consiste en un tipo de mensaje específico. De esta forma se asegura que los nodos que se comunican a través de un mismo topic estén de acuerdo en el formato de los datos a intercambiar.
- **Multiplicidad:** Varios nodos pueden publicar y suscribirse al mismo topic, permitiendo así compartir información y cooperar en un sistema robótico.

4.2.3. Nodos

Los «nodos», en ROS, son procesos de software individuales que realizan tareas específicas y se comunican entre sí para lograr un objetivo común en un sistema robótico o en cualquier otro sistema que se base en ROS. Cada nodo es una unidad independiente de ejecución que realiza una función en concreto, como controlar un sensor, ejecutar algoritmos de procesamiento de datos, enviar comandos a actuadores o proporcionar interfaces de usuario. Los nodos se ejecutan de manera concurrente y se comunican a través de un sistema de mensajería basado en temas o servicios, lo que permite una arquitectura modular y distribuida en la programación de robots y sistemas robóticos en ROS.

4.2.4. Clases

En ROS, una «clase» se refiere a una estructura organizativa que permite definir y encapsular funciones y datos relacionados en un nodo. Aunque no es una «clase» en el sentido tradicional de la programación orientada a objetos, se utiliza para modularizar y organizar el código en ROS. Esto facilita la comprensión, el mantenimiento y la

reutilización del código en proyectos robóticos. Las «clases» en ROS ayudan a dividir tareas complejas en componentes más pequeños y se comunican entre sí a través de mensajes, servicios y acciones para lograr una coordinación efectiva en la aplicación.

4.2.5. YALM

YAML, o YAML Ain't Markup Language, es un formato de serialización de datos ampliamente empleado en ROS y en programación en general. Este formato de archivo de texto plano se utiliza para representar datos legibles por humanos y fácilmente procesables por computadoras. En el contexto de ROS, YAML se utiliza para definir parámetros y configuraciones esenciales en sistemas robóticos, abarcando componentes como nodos, topics, servicios y transformaciones de coordenadas.

4.2.6. LAUNCH

Un archivo de lanzamiento o «launch» en ROS, es un componente ampliamente utilizado para iniciar y gestionar nodos, configuraciones y sistemas robóticos de manera eficiente. Estos archivos permiten definir configuraciones específicas para aplicaciones robóticas, coordinar múltiples nodos, fomentar la reutilización de configuraciones, simplificar la inicialización del sistema y proporcionar claridad y documentación. Además, son utilizados en una variedad de escenarios, como simulaciones, configuración de robots y aplicaciones de control de brazos robóticos, facilitando así el desarrollo y la gestión eficientes de sistemas robóticos complejos.

4.3. Arquitectura software

El sistema ROS se ha utilizado de manera específica en este proyecto en la implementación de la interfaz multimodal para el control del robot KUKA como robot quirúrgico. Su enfoque modular, capacidad de comunicación y control eficiente, soporte para hardware diverso y una comunidad comprometida de desarrolladores hacen que ROS sea una elección lógica para la creación de sistemas robóticos avanzados como el de este trabajo. Además, ROS proporciona un entorno de simulación, lo cual es una especificación clave dentro del alcance de la interfaz desarrollada. Esta capacidad de simulación se integra de manera natural con el uso de ROS, permitiendo pruebas exhaustivas antes de la implementación en el entorno real.

Esta comunicación entre el cirujano y el robot se va a llevar a cabo a través de topics. Dichos topics son generados por los nodos que rigen a la interfaz multimodal (Nodo iiwa_control_node) y al robot KUKA (Nodo KUKA), los cuales se pueden observar en el esquema de la Figura 21.

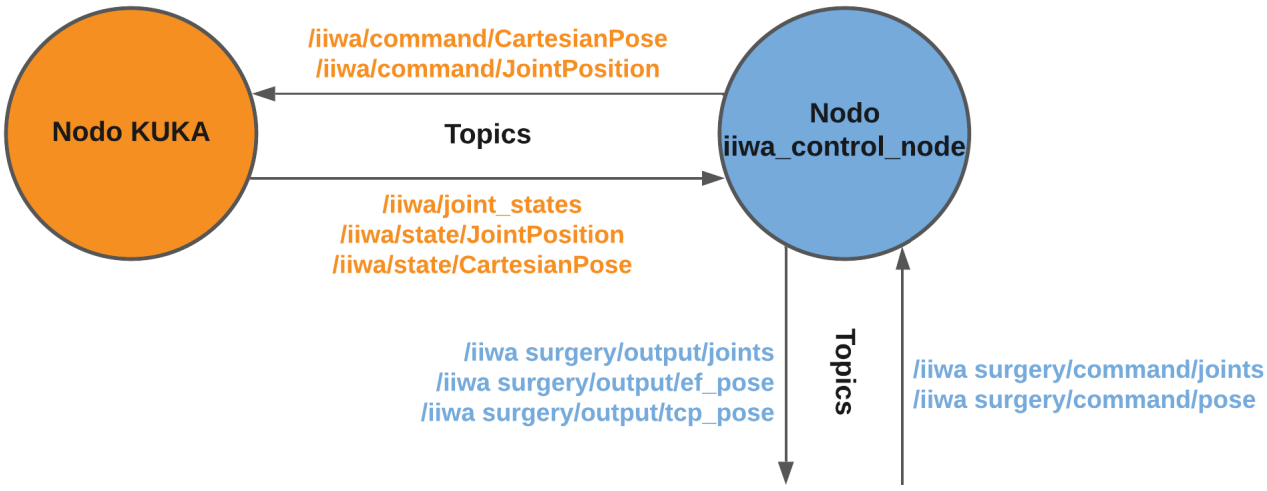


Figura 21: Comunicación con los nodos iiwa_control_node y KUKA.

El nodo KUKA que corre en la controladora del KUKA, genera una gran variedad de topics relacionados con estados y comandos. Para los fines de este TFM, se pondrá el foco en los siguientes topics que se generan tanto en simulación como en modo real, así como otros que son específicos de cada entorno:

- **/iiwa/command/CartesianPose:** Se utiliza para enviar comandos de posición en el espacio cartesiano al robot. Los mensajes en este topic son del tipo PoseStamped y contienen información sobre la posición y orientación a la que se desea mover el efector final del robot. Es posible comunicarse directamente a través de él con el robot (simulado o real) por la consola de comandos y sin necesidad de ejecutar la interfaz desarrollada.
- **/iiwa/command/JointPosition:** Se emplea para enviar comandos de posición de articulaciones al robot. Los mensajes en este topic son del tipo JointPosition y especifican las posiciones de las articulaciones a las que se desea mover el robot. Es posible comunicarse directamente a través de él con el robot (simulado o real) por la consola de comandos y sin necesidad de ejecutar la interfaz desarrollada.
- **/iiwa/joint_states:** Se emplea para publicar el estado de las articulaciones del robot simulado. Los mensajes en este topic son del tipo JointState y contienen información sobre las posiciones, velocidades y esfuerzos de las articulaciones del robot simulado en tiempo real. Es esencial para el monitoreo y el control de las articulaciones del robot simulado. Es posible recibir los datos publicados por el robot simulado directamente a través de él por la consola de comandos y sin necesidad de ejecutar la interfaz desarrollada.
- **/iiwa/state/JointPosition:** Se emplea para publicar la posición de las articulaciones del robot real. Los mensajes en este topic son del tipo JointPosition y especifican las posiciones de las articulaciones del robot real en un formato específico y en tiempo real. Es posible recibir los datos publicados por el robot real directamente a través de él por la consola de comandos y sin necesidad de ejecutar la interfaz desarrollada.
- **/iiwa/state/CartesianPose:** Se emplea para publicar la posición y orientación del efector final del robot en el espacio cartesiano. Los mensajes en este topic son del tipo PoseStamped y contienen información sobre la posición y orientación del efector final del robot en tiempo real. Este topic viene predeterminado en la configuración del robot tanto real como en su modo de simulación y es posible recibir los datos publicados por el robot (real o simulado) directamente a través de él por la consola de comandos y sin necesidad de ejecutar la interfaz desarrollada.

Por otro lado, los topics generados por la interfaz multimodal mediante la ejecución del nodo iiwa_control_node y que toman un importante papel en los movimientos a realizar por el robot y en la supervisión del estado del mismo son:

- **/iiwa_surgery/command/joints:** Se utiliza para recibir comandos de posición de articulaciones desde una fuente externa. Los mensajes en este topic son del tipo JointPosition y contienen información sobre las posiciones de las articulaciones a las que se desea mover el robot (real o simulado).
- **/iiwa_surgery/command/pose:** Se utiliza para recibir comandos de posición en el espacio cartesiano desde una fuente externa. Los mensajes en este topic son del tipo PoseStamped y especifican la posición y orientación a las que se desea mover el Tool Center Point del robot (real o simulado).
- **/iiwa_surgery/output/joints:** Se utiliza para publicar las posiciones de las articulaciones del robot en tiempo real. Los mensajes en este topic son del tipo JointPosition y proporcionan información sobre el estado de las articulaciones del robot (real o simulado).
- **/iiwa_surgery/output/ef_pose:** Se utiliza para publicar la posición y orientación del efector final del robot en tiempo real. Los mensajes en este topic son del tipo PoseStamped y proporcionan información sobre la posición y orientación del efector final del robot (real o simulado).
- **/iiwa_surgery/output/tcp_pose:** Se utiliza para publicar la posición y orientación del Tool Center Point del robot en tiempo real. Los mensajes en este topic son del tipo PoseStamped y proporcionan información sobre la posición y orientación del Tool Center Point del robot (real o simulado).

4.3.1. Nodo de la interfaz multimodal

La interfaz multimodal desarrollada para el control del robot quirúrgico KUKA LBR iiwa 7 R800 se estructura de manera eficiente, permitiendo una comunicación precisa y una operación segura. En el núcleo de esta interfaz se encuentra el nodo ROS denominado «iiwa_control_node». Este nodo, representado en la Figura 22, es el encargado de controlar y gestionar todas las interacciones entre la interfaz y el robot. Para lograrlo, implementa la clase «iiwa_surgery_class», que contiene los métodos necesarios para la comunicación y el control del robot. Los parámetros clave que influyen en el comportamiento del robot se almacenan en el archivo YAML «iiwa_surgery_params.yaml». Estos parámetros son cruciales para personalizar la operación del robot y se cargan desde el archivo al lanzar el nodo.

Por otro lado, el archivo «iiwa_surgery.launch» gestiona la operación del robot en dos modos: simulación y modo real. La selección de modo se basa en el argumento «simulation_mode», lo que permite adaptar la interfaz a las necesidades específicas de cada situación. En el modo de simulación, se crea un entorno virtual en Gazebo, y el nodo «iiwa_control_node» se ejecuta en ROS para poder interaccionar con este entorno, permitiendo al cirujano practicar y verificar procedimientos quirúrgicos de manera segura. En el modo real, el nodo se ejecuta en ROS para controlar el robot KUKA LBR iiwa 7 R800 físicamente, permitiendo la realización de intervenciones quirúrgicas reales. La comunicación entre el nodo de la interfaz y el nodo del robot se establece mediante topics. Esta estructura modular y bien definida de la interfaz multimodal proporciona un control minucioso y flexible del robot quirúrgico.

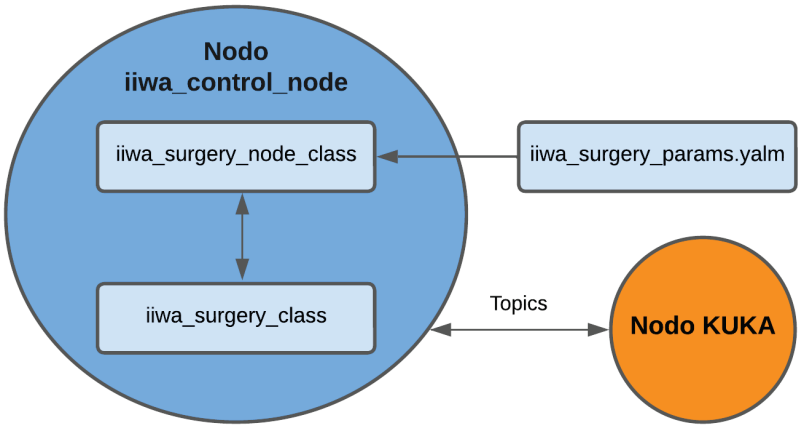


Figura 22: Nodo de la interfaz multimodal.

A continuación, se explorarán en detalle los componentes clave de la interfaz multimodal desarrollada. Esto incluye una descripción exhaustiva del nodo «iiwa_control_node», que actúa como el núcleo de la interfaz, la clase «iiwa_surgery_class», que contiene los métodos y funciones esenciales para controlar el robot y el archivo YAML «iiwa_surgery_params.yaml», que configura los parámetros fundamentales. Cada uno de estos componentes desempeña un papel crítico en el funcionamiento de la interfaz, permitiendo una comunicación efectiva y un control conciso del robot quirúrgico KUKA LBR iiwa 7 R800.

4.3.1.1 Nodo de control (iiwa_surgery_node)

Este nodo, cuyo código se puede encontrar en el Anexo B, tiene la función de actuar como una interfaz de control entre un sistema de control más amplio y un robot quirúrgico KUKA LBR iiwa 7 R800. Utiliza el entorno ROS para gestionar las instrucciones que le llegan del planificador de movimientos. Su tarea principal es traducir estas instrucciones en acciones específicas que el robot deberá llevar a cabo.

Este nodo es capaz de cambiar entre diferentes modos de funcionamiento del robot, permitiendo controlar tanto la posición de sus articulaciones como la posición cartesiana de su Tool Center Point en el espacio tridimensional. Puede ajustar parámetros relacionados con la herramienta utilizada por el robot, como su longitud y orientación, lo que es fundamental para realizar movimientos exactos.

Además, recopila información sobre el estado actual del robot en tiempo real, incluyendo la posición de sus articulaciones y la posición de su efector final (EF), y publica posteriormente esta información a través de ROS para que otros componentes del sistema estén al tanto de la situación del robot.

Durante las cirugías, el nodo va a realizar cálculos especiales para el modo «pivot» permitiendo que el robot realice movimientos alrededor de un punto de fulcro.

Por último, el nodo publica los resultados de sus acciones, como la posición de las articulaciones, la posición del efector final y la posición del Tool Center Point (TCP) a través de ROS. Este componente además de ser esencial para lograr un control determinado del robot quirúrgico en aplicaciones de robótica quirúrgica, es la base para el futuro desarrollo de un planificador de movimientos.

El nodo en cuestión va a estar dividido en las siguientes partes:

1. **Librerías**: Algunas de las librerías utilizadas se limitan a cumplir requisitos esenciales para la operación del nodo en el entorno de ROS. Sin embargo, otras desempeñan un papel crucial al abordar tareas computacionales más sofisticadas, tales como las conversiones entre diversas representaciones de la orientación. El conjunto de librerías empleadas en este programa comprende:

   - **rospy (ROS Python Client Library)**: Esencial para cualquier nodo generado en Python que funcione con ROS. Hace posible la comunicación con el sistema de ROS, incluyendo la inicialización del nodo, la configuración de suscriptores y publicadores, y la gestión de parámetros. A través de esta librería, el nodo puede publicar y recibir mensajes, controlar el ciclo de ejecución y acceder a información importante del sistema.
   - **numpy (Numerical Python)**: Muy utilizada para operaciones numéricas y matemáticas en Python. En el caso de este nodo, se emplea para realizar cálculos matemáticos y manipular arreglos de datos, lo que es útil para el procesamiento de posiciones y vectores en el espacio tridimensional.
   - **geometry_msgs**: Librería que proporciona un conjunto de mensajes definidos en ROS. Proporciona estructuras de datos para representar información geométrica, como posiciones en el espacio 3D. De todos estos mensajes, el nodo utiliza los mensajes de tipo «PoseStamped» para intercambiar información sobre la posición y orientación del robot, su efector final y el Tool Center Point.
   - **iiwa_msgs**: Similar a geometry_msgs, iiwa_msgs es una librería que proporciona un conjunto de mensajes específicos para el robot KUKA LBR iiwa 7 R800. De todos estos mensajes, el nodo utiliza los mensajes de tipo «CartesianPose» y «JointPosition», los cuales contienen información relevante para el control y la retroalimentación del robot, como la posición articular y las posiciones cartesiana y de fulcro.
   - **sensor_msgs**: Al igual que las dos librerías anteriores, sensor_msgs ofrece estructuras de datos para transmitir información de sensores. De todos estos mensajes, el nodo utiliza los mensajes de tipo «JointState» que contienen la información del estado de las articulaciones del robot. Resulta crucial para mantenerse al tanto de la situación del robot y para poder publicar su estado.
   - **conversions**: Librería personalizada desarrollada a partir de la librería «Pytransform3d» [29] para realizar conversiones específicas, como «matrix_from_quaternion» para convertir cuaternios en matrices de rotación.

Algoritmo 1: Importación de librerías y clases.

```
import rospy
import numpy as np
from geometry_msgs.msg import PoseStamped
from iiwa_msgs.msg import CartesianPose
from iiwa_msgs.msg import JointPosition
from iiwa_control_class import iiwa_surgery_class
from sensor_msgs.msg import JointState
from conversions import matrix_from_quaternion
```

2. **Configuración de parámetros del nodo**: Los parámetros determinan aspectos críticos del funcionamiento del nodo, como si está en modo de simulación o real, la longitud de la herramienta, la orientación de la misma, el punto de fulcro, la dirección IP del robot y el modo de trabajo. A través de la configuración de estos parámetros, que se recogen previamente en un YALM, el nodo puede adaptarse y funcionar de manera versátil en distintos contextos y entornos. Los parámetros a configurar son:

   - **self.simulation_mode**: Determina si el robot está funcionando en modo de simulación o en el entorno real.
   - **self.tool_length**: Indica la longitud de la herramienta adjunta al robot.
   - **self.tool_orientation**: Define la orientación de la herramienta en el espacio.
   - **self.fulcrum_fi**: Indica en qué punto de la longitud de la herramienta está el punto de fulcro.
   - **self.robot_ip**: La dirección IP del robot real, necesaria para la conexión con el hardware físico.
   - **self.work_mode**: Define el modo de funcionamiento del robot, como el modo libre («free») o modos pivotaje («pivot»).

Algoritmo 2: Obtención de valores de parámetros desde el sistema de parámetros de ROS

self.simulation_mode ← obtenerSimulationModeFromLAUNCH() (Valor Predeterminado: True)
self.tool_length ← obtenerToolLengthFromLAUNCH() (Valor Predeterminado: 0)
self.tool_orientation ← obtenerToolOrientationFromLAUNCH() (Valor Predeterminado: [0, 0, 0])
self.fulcrum_fi ← obtenerFulcrumFiFromLAUNCH() (Valor Predeterminado: 0)
self.robot_ip ← obtenerRobotIPFromLAUNCH() (Valor Predeterminado:)
self.work_mode ← obtenerWorkModeFromLAUNCH() (Valor Predeterminado: "free")

3. **Transmisión de parámetros a la clase**: Esta parte del código es fundamental para configurar los parámetros de la instancia de la clase iiwa_surgery_class. A partir de estas lineas de codigo, se transmiten los valores de parámetros clave antes vistos, como son el modo de operación, la longitud de la herramienta, la orientación, el punto de fulcro y la dirección IP del robot. Esto permite que la instancia de la clase esté informada y adaptada a las condiciones específicas de funcionamiento, asegurando un eficaz control del robot en su entorno.

Algoritmo 3: Configuración de la instancia de la clase iiwa_surgery_class con parámetros de ROS

self.iiwa = **iiwa_surgery_class()**
self.iiwa.simulation_mode = `self.simulation_mode`
self.iiwa.tool_length = `self.tool_length`
self.iiwa.set_tool_data(self.tool_length, self.tool_orientation)
self.iiwa.set_fulcrum_fi(self.fulcrum_fi)
self.iiwa.set_robot_ip(self.robot_ip)
self.iiwa.set_work_mode(self.work_mode)

4. **Configuración de publicadores y suscriptores**: Estas líneas de código se encargan de establecer los canales de comunicación del nodo con otros componentes del sistema en el entorno de ROS. El nodo utiliza suscriptores para recibir datos, como posiciones articulares y posiciones cartesianas, y publicadores para enviar información procesada, todo esto a través de los topics descritos previamente. La elección de suscriptores y publicadores depende en parte del modo de operación del robot, que puede ser simulado o real. Dichos publicadores y suscriptores son:

   - Suscriptores:
     - **self.joint_sub**: Escucha el topic «/iiwa_surgery/command/joints» y espera datos de tipo «JointPosition». Los datos recibidos activan la función «self.joint_command_callback».
     - **self.pose_sub**: Escucha el topic «/iiwa_surgery/command/pose» y espera datos de tipo «PoseStamped». Los datos recibidos activan la función «self.pose_command_callback».
     - Si el nodo está en modo de simulación:

- **self.joint_state_sub**: Escucha el topic «/iiwa/joint_states» y espera datos de tipo «JointState». Los datos recibidos activan la función «self.joint_state_callback»‘.
- **self.cartesian_pose_sub**: Escucha el topic «/iiwa/state/CartesianPose» y espera datos de tipo «PoseStamped». Los datos recibidos activan la función «self.cartesian_pose_callback».

- Si el nodo no está en modo de simulación:
  - **self.joint_position_sub**: Escucha el topic «/iiwa/state/JointPosition» y espera datos de tipo «JointPosition». Los datos recibidos activan la función «self.joint_position_callback».
  - **self.cartesian_pose_sub**: Escucha el topic «/iiwa/state/CartesianPose» y espera datos de tipo «CartesianPose». Los datos recibidos activan la función «self.cartesian_pose_callback».

- Publicadores:
  - **self.joint_pub**: Publica datos en el topic «/iiwa_surgery/output/joints» en forma de «JointPosition». Otros nodos o componentes pueden suscribirse a este topic para recibir los datos procesados.
  - **self.ef_pose_pub**: Publica datos en el topic «iiwa_surgery/output/ef_pose» en forma de «PoseStamped». Estos datos representan la posición del efector final del robot. Otros nodos pueden utilizar esta información.
  - **self.tcp_pose_pub**: Publica datos en el topic «iiwa_surgery/output/tcp_pose» en forma de «PoseStamped». Estos datos representan la posición del Tool Center Point (TCP) del robot. Otros componentes pueden suscribirse a este topic para acceder a esta información.

Algoritmo 4: Configuración de suscriptores y publicadores de la interfaz

```
self.joint_sub ← Suscriptor. Topic: /iiwa_surgery/command/joints.
self.pose_sub ← Suscriptor. Topic: /iiwa_surgery/command/pose.
self.joint_pub ← Publicador. Topic: /iiwa_surgery/output/joints.
self.ef_pose_pub ← Publicador. Topic: /iiwa_surgery/output/ef_pose.
self.tcp_pose_pub ← Publicador. Topic: /iiwa_surgery/output/tcp_pose.
if self.simulation_mode es verdadero then
    self.joint_state_sub ← Suscriptor. Topic: /iiwa/joint_states.
    self.cartesian_pose_sub ← Suscriptor. Topic: /iiwa/state/CartesianPose.
else
    self.joint_position_sub ← Suscriptor. Topic: /iiwa/state/JointPosition.
    self.cartesian_pose_sub ← Suscriptor. Topic: /iiwa/state/CartesianPose.
```

5. **Funciones callback**: Las funciones callback en ROS actúan como interruptores inteligentes que se activan automáticamente en respuesta a eventos específicos, como la llegada de datos a través de suscriptores. Estas funciones son esenciales para procesar, interpretar y tomar acciones basadas en la información recibida en tiempo real. En el caso de este nodo se tienen las siguientes funciones callback:

   - **joint_command_callback**: Esta función es llamada al recibir la interfaz un mensaje a través del topic «/iiwa_surgery/command/joints». Dicho mensaje especifica la posición deseada de las articulaciones del robot. La función procesa ese mensaje y utiliza la información contenida en él para enviar comandos al robot y lograr que se mueva a la posición deseada. Para ello, la función crea una instancia del mensaje «JointPosition» y lo iguala al mensaje recibido. Luego, este mensaje se envía al robot a través de la función «move_joint» de la clase «iiwa_surgery_class». El resultado es que el robot ajusta sus articulaciones para alcanzar la configuración de posición indicada en el mensaje.

Algoritmo 5: Función `joint_command_callback`

```
Data: Mensaje msg con configuración articular
Result: Movimiento del robot a posición articular específica
joint_config = new JointPosition()
joint_config = msg
self.iiwa.move_joint(joint_config)
```

- **pose_command_callback**: La función es esencial para controlar el movimiento del robot en función del modo de operación. Esta función es llamada al recibir la interfaz un mensaje a través del topic «iiwa_surgery/command/pose». Cuando el modo de operación es «free», la función toma un mensaje que especifica una posición cartesiana deseada y utiliza ese mensaje para mover el robot a esa posición. Por otro lado, si el modo de operación es «pivot», la función permite al robot moverse alrededor de un punto de fulcro. En este caso, se ajusta la posición del robot en función de la posición especificada en el mensaje y se calcula un vector de incrementos para lograr el movimiento alrededor del punto de fulcro. Si el modo de operación no es válido, se emite una advertencia. A diferencia de la función anterior, en este caso no se hace una traslación directa de topics ya que se tiene que pasar de TCP a EF en cualquier caso.

Algoritmo 6: Función `pose_command_callback`

```
Data: Mensaje recibido msg con configuración de pose
Result: Movimiento del robot según el modo de trabajo
pose = new PoseStamped()
pose = msg
if self.work_mode es "free" then
    self.iiwa.move_cartesian(pose)
else if self.work_mode es "pivot" then
    position = msg.pose.position
    if self.first_position es None then
        self.first_position = position
        increment_vector = [0, 0, 0]
        j = 0
        self.iiwa.move_cartesian_fulcrum(pose, increment_vector, j)
    else
        increment_vector = position - self.first_position
        j = 1
        self.iiwa.move_cartesian_fulcrum(pose, increment_vector, j)
else
    rospy.logwarn("Modo de trabajo no válido.")
```

- **joint_state_callback**: Esta función es llamada cuando se trabaja en modo simulación y al recibir la interfaz un mensaje a través del topic «/iiwa/joint_states». Se encarga de procesar y reorganizar los datos de posición articular del robot. Cuando se recibe un mensaje que contiene información sobre la posición de las articulaciones del robot, esta función crea un nuevo mensaje de tipo «JointPosition» y copia la información recibida en los campos correspondientes de este nuevo mensaje. Luego, publica este mensaje procesado en el topic «/iiwa_surgery/output/joints».

Algoritmo 7: Función `joint_state_callback`

```
Data: Mensaje msg con estado de posición articular
Result: Publicar estado de posición articular
joint_position_msg = new JointPosition()
joint_position_msg.header = msg.header
for i de 1 a 7 do
    joint_position_msg.position.ai = msg.position[i-1]
self.joint_pub.publish(joint_position_msg)
```

- **joint_position_callback**: Esta función es llamada cuando se trabaja en modo real y al recibir la interfaz un mensaje a través del topic «/iiwa/state/JointPosition». Es responsable de publicar el estado de la posición articular recibida. Cuando se recibe un mensaje que contiene información sobre la posición de las articulaciones del robot, esta función toma ese mensaje y lo publica directamente el topic «/iiwa_surgery/output/joints».

Algoritmo 8: Función `joint_position_callback`

Data: Mensaje `msg` con estado de posición articular
Result: Publicar estado de posición articular
self.joint_pub.publish(msg)

- **cartesian_pose_callback**: Esta función es llamada al recibir la interfaz un mensaje a través del topic «/iiwa/state/CartesianPose». La función es responsable de procesar y publicar información relacionada con la posición cartesiana del robot. Su funcionamiento se adapta en función del modo de simulación configurado. Si se está en modo simulación, la función toma el mensaje que describe la información del efector final directamente y lo publica. En caso de estar en modo operativo con un robot real realiza una serie de cálculos para transformar y ajustar la información recibida del efector final antes de publicarla. Además, sea cual sea el modo de operación (simulado o real), también se encarga del cálculo del Tool Center Point a partir de la información recibida del EF y de su posterior publicación.

Algoritmo 9: Función `cartesian_pose_callback`

Data: Mensaje `msg` con configuración de posición cartesiana
Result: Publicar estado de posición cartesiana transformada en TCP
cartesian_pose = new `PoseStamped()`
if ***self.simulation_mode*** *es verdadero* **then**
  cartesian_pose = `msg`
else
  cartesian_pose = `msg.poseStamped` (header, posición y orientación)
self.ef_pose_pub.publish(cartesian_pose)
position = `cartesian_pose.pose.position`
orientation = `cartesian_pose.pose.orientation`
Rm = `matrix_from_quaternion(orientation)`
direction_transformed = `[0,0,self.tool_length] Rm`
Pef = `position`
Ptcp = `Pef + direction_transformed`
position = `Ptcp`
tcp_pose = new `PoseStamped()`
tcp_pose = `actualizarMensaje(cartesian_pose.header, position, orientation)`
self.tcp_pose_pub.publish(tcp_pose)

6. **Bucle de Ejecución**: Este bucle mantiene el nodo en funcionamiento a una frecuencia de 10 Hz y permite realizar procesamiento adicional si es necesario mientras no se detenga la ejecución de ROS.

Algoritmo 10: Función `run`

Data: Ninguno
Result: Bucle de ejecución con frecuencia de 10 Hz
rate = `rospy.Rate(10)`
while *no se detenga el nodo de ROS* **do**
  rate.sleep()
end

7. **Inicialización y ejecución del nodo**: Contiene la inicialización y ejecución del nodo. Crea una instancia del nodo y luego llama a su función «run» para mantenerlo en funcionamiento, permitiendo que el nodo procese información y realice acciones específicas mientras ROS está en ejecución.

Algoritmo 11: Ejecución del nodo principal de la aplicación.

```
if __name__ == '__main__' then
    node = IiwaSurgeryNode()
    node.run()
```

La ejecución de este nodo se encuentra gestionada por el archivo de lanzamiento «iiwa_surgery.launch», cuyo código se puede encontrar en el Anexo E.

Dicho archivo, comienza definiendo un argumento llamado «simulation_mode» con un valor predeterminado de «false». Este argumento permite seleccionar entre dos modos de operación: simulación (simulación_mode = true) o modo real (simulación_mode = false).

Posteriormente, carga los parámetros de configuración desde el archivo «iiwa_surgery_params.yaml», los cuales van a configurar el comportamiento del robot y definir sus características.

El archivo de lanzamiento se divide en dos grupos condicionales. Si «simulation_mode» se establece en true, el sistema se inicia en modo de simulación. Esto implica el lanzamiento de un entorno de simulación en Gazebo utilizando un archivo de lanzamiento específico y la ejecución del nodo «iiwa_control_node» en modo simulación.

Por otro lado, si «simulation_mode» se establece en false, el sistema se inicia en modo real. En este caso, se ejecuta el nodo «iiwa_control_node» en modo real, lo que implica el control y operación del robot KUKA LBR iiwa 7 R800 en un entorno físico.

Algoritmo 12: Archivo `iiwa_surgery.launch`

```
Data: Argumento simulation_mode con valor predeterminado de false
obtenerParametrosFromYALM()
if simulation_mode es true then
    launch GazeboConSunrise
    run iiwa_control_node
else
    run iiwa_control_node
```

4.3.1.2 Clase de control (iiwa_surgery_class)

La clase «iiwa_surgery_class», la cual se define mediante el código expuesto en el Anexo C, tiene como objetivo proporcionar funcionalidad para controlar el robot KUKA iiwa en el contexto de la cirugía robótica. Esta clase permite configurar el modo de trabajo del robot (ya sea en simulación o en un robot real), definir datos de la herramienta, establecer el punto de fulcro, y controlar el movimiento del robot en modos articulares y cartesianos.

Además, la clase incluye funciones para mover el robot a posiciones articulares y cartesianas específicas, teniendo en cuenta datos como la longitud de la herramienta y la orientación. También se incluye la capacidad de mover el robot en modo de pivotaje («pivot») alrededor de un punto de fulcro, calculando las nuevas posiciones del efector final (EF) y del Tool Center Point (TCP).

La clase en cuestión va a estar dividida en las siguientes partes:

1. **Librerías**: Establecen las bases para el funcionamiento del programa, importando las herramientas necesarias para interactuar con ROS y el robot Kuka. También incluyen funcionalidades matemáticas y de manejo de archivos para realizar cálculos y almacenar datos. Además, se importan funciones para la conversión de datos geométricos y posiciones articulares, esenciales para el control del robot KUKA en este contexto de programación. El conjunto de librerías empleadas en esta clase comprende:

- **rospy**: Esencial para cualquier código generado en Python que funcione con ROS. Permite la creación de nodos y clases en ROS, la publicación y suscripción de mensajes, y otras operaciones clave para la interacción con el entorno de ROS.
- **numpy**: Fundamental para la computación científica en Python. Proporciona estructuras de datos para realizar cálculos numéricos eficientes y es ampliamente utilizada para operaciones matemáticas y algebraicas en este contexto.
- **math**: Utilizada para operaciones matemáticas. Proporciona funciones y constantes matemáticas, lo que es útil para realizar cálculos matemáticos simples.
- **geometry_msgs**: Librería que proporciona un conjunto de mensajes definidos en ROS. Proporciona estructuras de datos para representar información geométrica, como posiciones en el espacio 3D. De todos estos mensajes, la clase utiliza los mensajes de tipo «PoseStamped» para intercambiar información sobre la posición y orientación del robot, su efector final y el Tool Center Point.
- **iiwa_msgs**: Similar a geometry_msgs, iiwa_msgs es una librería que proporciona un conjunto de mensajes específicos para el robot KUKA LBR iiwa 7 R800. De todos estos mensajes, la clase utiliza los mensajes de tipo «JointPosition», los cuales contienen información relevante para el control y la retroalimentación del robot, como la posición articular.
- **conversions**: Librería personalizada desarrollada a partir de la librería «Pytransform3d» [29] para realizar conversiones específicas, como «matrix_from_quaternion» para convertir cuaternios en matrices de rotación o «quaternion_from_matrix» para realizar el proceso inverso.

Algoritmo 13: Importación de librerías.

```
import rospy
import numpy as np
import math
from conversions import matrix_from_quaternion
from conversions import quaternion_from_matrix
from geometry_msgs.msg import PoseStamped
from iiwa_msgs.msg import JointPosition
```

2. **Inicialización de parámetros clave del robot robot**: La función constructor «__init__» de esta clase tiene la función principal de inicializar los atributos de la clase y configurar su entorno. Entre los aspectos más destacados, encontramos:

   - **self.simulation_mode**: Permite configurar el modo de ejecución, ya sea en modo de simulación o en un robot real. Se inizializa con valor booleano de True.
   - **self.tool_length** y **self.tool_orientation**: Almacenan información sobre la herramienta utilizada en el robot, como su longitud y orientación. Se inicializan con valores de longitud y orientación de 0.
   - **self.fulcrum_fi**: Punto de la longitud de la herramienta en el que se encuentra el punto de fulcro. Su valor debe estar dentro del rango entre 0 y 1, donde 0 representa el EF y 1 el TCP. Se inicia con un valor predeterminado de 0.
   - **self.robot_ip**: Contiene la dirección IP del robot con el que se va a trabajar. Se inicializa como una cadena vacía.
   - **self.work_mode**: Define el modo de funcionamiento del robot, siendo «free» el valor predeterminado. Se inicia en el modo «free» por defecto, pero puede cambiarse a «pivot».
   - **self.cartesian_pub** y **self.joint_pub**: Publicadores ROS que permiten enviar información sobre la posición cartesiana y las posiciones articulares del robot a través de ROS.

Algoritmo 14: Inicialización de la clase con configuración predeterminada.

Data: Parámetro `simulation_mode` con valor por defecto `True`
Result: Inicialización de la clase
self.simulation_mode = `simulation_mode`
self.tool_length = 0
self.tool_orientation = [0, 0, 0]
self.fulcrum_fi = 0
self.robot_ip= "192.228.17.57"
self.work_mode = "free"
self.cartesian_pub ← Publicador. Topic: /iiwa/command/CartesianPose.
self.joint_pub ← Publicador. Topic: /iiwa/command/JointPosition.

3. **Configuración de parámetros clave del robot robot**: Se definen una serie de funciones para configurar los parámetros clave en el control del robot KUKA. Estos ajustes son esenciales para adaptar el comportamiento del robot a las necesidades específicas de la aplicación o tarea que realizará. Estas funciones son:

   - **Función set_tool_data**: Configura los parámetros «self.tool_length» y «self.tool_orientation».

Algoritmo 15: Función `set_tool_data`.

Data: Valores de `tool_length` y `tool_orientation`
Result: Datos de la herramienta configurados
self.tool_length = `tool_length`
self.tool_orientation = `tool_orientation`

   - **Función set_fulcrum_fi**: Configura el parámetro «self.fulcrum_fi».

Algoritmo 16: Función `set_fulcrum_fi`.

Data: Valor de `fulcrum_fi`
Result: Punto de la longitud de la herramienta en la que se encuentra el punto de fulcro configurado
if $0 \leq fulcrum\_fi \leq 1$ **then**
    self.fulcrum_fi = `fulcrum_fi`
else
    rospy.logwarn("El punto de fulcro debe ser un valor decimal entre 0 y 1.")

   - **Función set_robot_ip**: Configurar el parámetro «robot_ip».

Algoritmo 17: Función `set_robot_ip`.

Data: Dirección IP del robot (`robot_ip`)
Result: Dirección IP del robot configurada
self.robot_ip = `robot_ip`

   - **Función set_work_mode**: Configura el parámetro «work_mode».

Algoritmo 18: Función `set_work_mode`.

```
Data: Modo de trabajo (work_mode)
Result: Modo de trabajo del robot configurado
    if work_mode está en ["free", "pivot"] then
        self.work_mode = work_mode
    else
        rospy.logwarn("Modo de trabajo no válido.")
```

4. **Funciones de control de movimiento del robot**: Por último se proporcionan funciones para controlar el robot en tres modos diferentes. Cada función utiliza mensajes ROS para comunicarse y poder ejecutar el movimiento del robot. Estas funciones son:

   - **Función move_joint**: Se utiliza para mover el robot en posiciones articulares. Toma una configuración de articulaciones como entrada y crea un mensaje de posición de articulaciones. Luego, publica este mensaje utilizando ROS para controlar el movimiento del robot en sus articulaciones.

Algoritmo 19: Función `move_joint`.

```
Data: Mensaje de posición articular (joint_config)
Result: Movimiento del robot a la posición articular deseada
    joint_msg = new JointPosition()
    joint_msg = joint_config
    self.joint_pub.publish(joint_msg)
```

   - **Función move_cartesian**: Se encarga de mover la herramienta del robot a una posición y orientación determinadas en el espacio cartesiano, de manera libre y sin restricciones. Para lograrlo, toma un mensaje de posición «pose» como entrada, que incluye información sobre la posición y la orientación del Tool Center Point (TCP). Luego, calcula la posición del EF en base a esta información y la longitud de la herramienta utilizada. Una vez obtenida, esta nueva posición y orientación se envía como mensaje a través de ROS para llevar a cabo el movimiento del robot.

Algoritmo 20: Función `move_cartesian`.

```
Data: Mensaje de posición cartesiana (pose)
Result: Movimiento del robot a una posición cartesiana específica
    cartesian_msg = new PoseStamped()
    position = pose.pose.position
    orientation = pose.pose.orientation
    Rm = matrix_from_quaternion(orientation)
    direction_transformed = [0,0,-self.tool_length] Rm
    Ptcp = position
    Pef = Ptcp + direction_transformed
    position = Pef
    cartesian_msg = actualizarMensaje(pose.header, iiwa_link_0, position, orientation)
    self.cartesian_pub.publish(cartesian_msg)
```

   - **Función move_cartesian_fulcrum**: Se encarga de mover la herramienta del robot de manera concreta alrededor de un punto de fulcro específico.

     En su primera parte, el código toma una posición y orientación como entrada. Esta información representa la ubicación deseada para el TCP en el espacio tridimensional. Luego, se realiza una serie de cálculos para ajustar la posición del EF. Esto implica transformar el cuaternio de orientación en una matriz de rotación, calcular la dirección relativa al EF del robot, y luego actualizar la posición del EF en función de estos cálculos. Teniendo las posiciones y orientaciones tanto de EF como del TCP, se calcula la posición

del llamado «punto de fulcro». Finalmente, se publica la posición deseada del EF.

La segunda parte del código se ocupa de garantizar que la herramienta del robot se mueva alrededor del punto de fulcro. Cuando se recibe una nueva posición y orientación del TCP, el código calcula cómo mover el EF del robot alrededor de este punto de fulcro. Realiza los cálculos específicos para determinar la nueva posición del EF en relación con el punto de fulcro y ajusta la orientación en consecuencia, para acabar publicando dicha información.

Algoritmo 21: Función `move_cartesian_fulcrum`

Data: Mensaje de posición (`pose`), vector de incrementos (`increment_vector`), contador (`j`)
Result: Movimiento del robot alrededor de un punto de fulcro

```
cartesian_msg = new PoseStamped()
if j == 0 then
    position = pose.pose.position
    orientation = pose.pose.orientation
    Rm = matrix_from_quaternion(orientation)
    direction_transformed = [0,0,-self.tool_length] Rm
    self.Ptcp = position
    Pef = self.Ptcp + direction_transformed
    direction_transformed = [0,0,self.fulcrum_fi * self.tool_length] Rm
    self.Pf = Pef + direction_transformed
    cartesian_msg = actualizarMensaje(pose.header, iiwa_link_0, Pef, orientation)
if j == 1 then
    Ptn = self.Ptcp + increment_vector
    zn = self.Pf - Ptn
    Mzn = sqrt(zn[0]^2 + zn[1]^2 + zn[2]^2)
    ro = self.tool_length - Mzn
    Pn = self.Pf + ro * zn / Mzn
    znn = -zn / Mzn
    xnn = ([0, 0, 1] x znn) / ||[0, 0, 1] x znn||
    ynn = (znn x xnn) / ||znn x xnn||
    M = [xnn ynn znn]
    q = quaternion_from_matrix(M)
    cartesian_msg = actualizarMensaje(pose.header, iiwa_link_0, Pn, q)
self.cartesian_pub.publish(cartesian_msg)
```

4.3.1.3 Fichero de configuración YALM (iiwa_surgery_params.yalm)

El archivo YAML «iiwa_surgery_params.yaml», cuyo código se puede encontrar en el Anexo D, desempeña un papel fundamental en tu trabajo al definir y configurar los parámetros clave utilizados posteriormente en la operación del robot quirúrgico KUKA LBR iiwa 7 R800. Este archivo se utiliza para proporcionar información específica que afecta directamente al comportamiento y funcionamiento del nodo «iiwa_control_node» a través del archivo de lanzamiento «iiwa_surgery.launch».

En este archivo YALM, el operedor va a poder configurar los siguientes parámetros:

- **Modo de simulación**: El parámetro «simulation_mode» define si el robot opera en modo de simulación o en modo real. Esto permite cambiar entre entornos de simulación y configuraciones del robot en el mundo real, lo que es crucial para pruebas y experimentación sin riesgos en entornos virtuales antes de la implementación en el robot real.
- **Longitud de la herramienta**: El parámetro «tool_length» indica la longitud de la herramienta en relación con el efector final (EF) del robot. Esta información es esencial para la cinemática y la planificación de trayectorias, ya que la longitud de la herramienta afecta directamente a la geometría del robot.

- **Orientación de la herramienta**: El parámetro «tool_orientation» describe la orientación de la herramienta en relación con el EF. Esto es crítico para definir cómo se coloca y orienta la herramienta en el espacio tridimensional, lo que influye en su capacidad para realizar tareas quirúrgicas específicas.
- **Punto de fulcro**: El parámetro «fulcrum_fi» especifica en qué punto a lo largo de la longitud de la herramienta se encuentra el punto de fulcro. Esto es esencial para controlar los movimientos de pivote alrededor de un punto de fulcro, uno de los aspectos clave de tu trabajo.
- **IP del robot**: El parámetro «robot_ip» indica la dirección IP del robot. Esta información será necesaria para establecer una conexión y comunicación efectiva con el robot KUKA LBR iiwa 7 R800, ya que se debe conocer la dirección a la que enviar comandos y recibir datos.
- **Modo de trabajo**: El parámetro «work_mode» define si el robot opera en modo «pivot» (pivotaje alrededor del punto de fulcro) o en modo «free» (modo libre). Esto determina cómo el robot responde a las instrucciones y control.

4.4. Documentación

La documentación desempeña un papel básico en el desarrollo de proyectos de software, especialmente en campos altamente técnicos como la robótica quirúrgica. Proporciona una guía completa y detallada, aportando claridad y comprensión sobre cómo funcionan los componentes del sistema, su propósito y cómo utilizarlos. En este sentido, para poder conseguir dicha finalidad en la investigación desarrollada, se ha utilizado Doxygen para generar documentación de código fuente de alta calidad.

4.4.1. Doxygen: ¿Qué es y para qué se utiliza?

Doxygen es una herramienta de generación de documentación de código abierto que es utiliza para crear documentación técnica a partir de comentarios incrustados en el código fuente. Su principal objetivo es facilitar la comprensión del código, ayudar en su mantenimiento y permitir que otros colaboradores puedan utilizar y extender el software de manera efectiva.

Doxygen es ampliamente utilizado en proyectos de desarrollo de software, especialmente en lenguajes de programación como C++, C, Python, Java, etc. Permite generar documentación en diversos formatos, como HTML, PDF, LaTeX, y otros, lo que lo convierte en una herramienta de gran versatilidad y adaptabilidad a diversas necesidades [30].

Esta herramienta es altamente adaptable y es compatible con la mayoría de los sistemas Unix, además de funcionar en entornos Windows y Mac OS X. Presenta una serie de ventajas y características destacadas:

- **Generación de documentación en diversos formatos:** Doxygen puede generar tanto documentación en línea, como un manual de referencia fuera de línea, a partir de un conjunto de archivos fuente debidamente documentados. Además, admite la generación de salida en varios formatos, como HTML, PDF, RTF (MS-Word), PostScript, HTML comprimido y páginas de manual UNIX. Lo que destaca es que la documentación se extrae directamente de los fuentes, lo cual facilita la coherencia entre la documentación y el código fuente.
- **Estructuración automática del código:** Doxygen se puede configurar para extraer la estructura del código a partir de archivos fuente no documentados. Esta característica resulta especialmente útil cuando se trabaja con proyectos de gran envergadura.
- **Documentación general:** Más allá de la generación de documentación técnica, Doxygen se puede emplear para crear documentación convencional. Esto significa que puede ser útil en una amplia gama de contextos, desde proyectos de software hasta la creación de documentación general para distintas aplicaciones.

4.4.2. Generación de documentación Doxygen

Para asegurar que la interfaz desarrollada en esta investigación quede bien documentada y sea accesible para otros miembros de la comunidad y colaboradores externos, se ha generado documentación Doxygen para los archivos iiwa_control_node.py y iiwa_control_class.py. Dichos archivos suponen el núcleo central de la interfaz multimodal llevada a cabo y es de gran importancia disponer de una detallada documentación de los mismos. Esta documentación generada se puede encontrar al final de esta memoria en el Anexo F y más detalladamente en el enlace Documentación Doxygen - TFM_JavierLara.

A continuación, se describe cómo se ha realizado este proceso de generación de la documentación en cuestión:

4.4.2.1 Bloques de comentarios Doxygen

Un bloque de comentarios Doxygen es un bloque de comentarios en Pyhton, C, C++, VHDL o Fortran con algunas marcas adicionales. Esto permite que Doxygen reconozca el fragmento como un texto estructurado que debe terminar en la documentación a generar.

Para ambos archivos Python que se pretenden documentar, se han agregado estos comentarios Doxygen en su código. Estos comentarios incluyen descripciones detalladas de las clases, funciones, variables y otros elementos del código, lo que permite que Doxygen genere documentación coherente y comprensible.

A continuación, se detalla cómo se han utilizado los bloques de comentarios especiales en Python para lograr una correcta documentación:

- En Python, se utilizan cadenas de documentación (") como un método estándar para documentar el código. Doxygen es capaz de extraer estos comentarios, asumiendo que deben representarse de forma preformateada en la documentación generada. Un ejemplo de documentación en Python en el código de uno de los archivos a comentar podría ser el siguiente:

```
1  class iiwa_surgery_node_class:
2      """!
3      Clase que define el nodo ROS para el control del robot KUKA LBR iiwa con fines
       quirurgicos.
4
5      Es importante destacar que los parametros de configuracion utilizados en esta clase se
       obtienen del archivo iiwa_surgery_params.yaml, los cuales son cargados en el sistema de
       parametros de ROS a partir del archivo de lanzamiento iiwa_surgery.launch. A continuacion
       , se muestra como se enlazan estos parametros con las variables de la clase:
6      - ‘simulation_mode‘: Indica si el robot esta en modo de simulacion o no (True para
       simulacion, False para el robot real).
7      - ‘tool_length‘: Almacena la longitud de la herramienta utilizada en el robot.
8      - ‘tool_orientation‘: Lista que almacena la orientacion de la herramienta en radianes en
       el formato [roll, pitch, yaw].
9      - ‘fulcrum_fi‘: Valor que indica en que punto de la longitud de la herramienta se
       encuentra el punto de fulcro, siendo 0 para el EF y 1 para el TCP.
10     - ‘robot_ip‘: Almacena la direccion IP del robot iiwa.
11     - ‘work_mode‘: Indica el modo de trabajo predeterminado del robot ("free" para movimiento
        libre y "pivot" para movimiento alrededor de un punto de fulcro).
12     """
13
14     def __init__(self):
15         """!
16         Constructor de la clase.
17         @protected
18         """
19         rospy.init_node('iiwa_surgery_node')
20
```

El fragmento de código proporcionado del archivo iiwa_control_node.py contiene comentarios de estilo Doxygen que ayudan a documentar la clase iiwa_surgery_node_class y su constructor __init__ proporcionando detalles sobre su propósito y funcionamiento y los parámetros de configuración que utilizan. También se utiliza

el comando especial @protected para indicar que este constructor es una parte protegida de la clase.

- Además de las cadenas de documentación ("), Python también admite comentarios de estilo Doxygen utilizando ##. Esto permite utilizar comandos especiales de Doxygen en lugar de mostrar el texto tal cual. A continuación se muestra un ejemplo de este tipo de documentación en Python en el código de uno de los archivos a comentar:

```
        ## @protected
        # Indica si el robot esta en modo de simulacion o no, siendo True para simulacion y
    False para el robot real.
        self.simulation_mode = rospy.get_param("simulation_mode", True)

        ## @protected
        # Almacena la longitud de la herramienta utilizada en el robot.
        self.tool_length = rospy.get_param("tool_length", 0.0)

        ## @protected
        # Lista que almacena la orientacion de la herramienta en radianes con el en formato [
    roll, pitch, yaw].
        self.tool_orientation = rospy.get_param("tool_orientation", [0.0, 0.0, 0.0])

        ## @protected
        # Valor que indica en que punto de la longitud de la herramienta se encuentra el
    punto de fulcro. Debe estar entre 0 y 1, siendo 0 el EF y el 1 el TCP.
        self.fulcrum_fi = rospy.get_param("fulcrum_fi", 0.0)

        ## @protected
        # Almacena la direcci n IP del robot iiwa.
        self.robot_ip = rospy.get_param("robot_ip", "")

        ## @protected
        # Indica el modo de trabajo predeterminado del robot ("free" para movimiento libre y
    "pivot" para movimiento alrededor de punto de fulcro).
        self.work_mode = rospy.get_param("work_mode", "free")

```

Este fragmento de código proporcionado del archivo iiwa_control_node.py contiene comentarios que utilizan la sintaxis de Doxygen para documentar variables específicas y proporcionar información sobre su uso y significado en el código. También se utilizan etiquetas como @protected, indicando que estas variables son parte del código que debe tratarse con cierto nivel de restricción o protección.

4.4.2.2 Instalación de Doxygen

Una vez completada la fase de documentación del código, se procede a la instalación del software Doxygen, que permitirá la generación de la documentación. Aunque es posible su instalación en multitud de sistemas operativos, para este caso se ha usado un sistema Windows, en el que los pasos llevados a cabo para su instalación han sido:

1. Descarga de Doxygen:
   - Ir a la página de descargas de Doxygen https://www.doxygen.nl/download.html y descargar la última distribución disponible. Comprobar que la descarga de Doxygen sea la adecuada para la versión de Windows utilizada.

2. Instalación de CMake:
   - Antes de compilar Doxygen en Windows, se necesita tener CMake instalado. Es posible descargar CMake desde su sitio oficial https://cmake.org/download/. Seguir las instrucciones de instalación.

3. Instalación de Bison y Flex:
   - Descargar e instalar las versiones modernas de Bison y Flex para Windows. Se pueden obtener en https://sourceforge.net/projects/winflexbison/.
   - Después de la instalación, asegúrate de agregar las rutas de instalación al PATH de Windows.

4. Instalación de Python:
   - Comprobar tener Python instalado en el sistema. Doxygen requiere Python (versión 2.7 o superior) para compilar. Se puede obtener Python desde https://www.python.org.

5. Descomprimir Doxygen:
   - Colocar el archivo tarball de las fuentes de Doxygen en una ubicación de su elección. Posteriormenete, descomprimir las fuentes utilizando una herramienta como tar o 7-Zip.

6. Configuración de CMake:
   - Abrir el «Visual Studio Native Command Shell» correspondiente a su versión de Visual Studio.
   - Navegar a la carpeta donde se descomprimió las fuentes de Doxygen.
   - Crear un directorio para construir el proyecto y cambiar a él: mkdir build y luego cd build.
   - Ejecutar CMake para generar los archivos del proyecto.

7. Compilación con Visual Studio:
   - Abre el archivo de proyecto generado con Visual Studio.

8. Compilación desde la línea de comandos (opcional):
   - Si se prefiere compilar desde la línea de comandos, se puede utilizar los siguientes comandos:
   - mkdir build
   - cd build
   - cmake -G "NMake Makefiles"..
   - nmake

9. Instalación de Doxygen:
   - Después de la compilación, ejecutar make install para instalar Doxygen. Esto instalará Doxygen en el sistema.

4.4.2.3 Herramientas de Doxygen

El ejecutable «doxygen» es la aplicación principal que analiza las fuentes y genera la documentación. Para este caso, se ha hecho uso de «doxywizard», que se trata de una interfaz gráfica para editar el archivo de configuración utilizado por Doxygen y ejecutar Doxygen en un entorno gráfico.

La Figura 23, expuesta a continuación, muestra la relación entre estas herramientas y el flujo de información entre ellas:

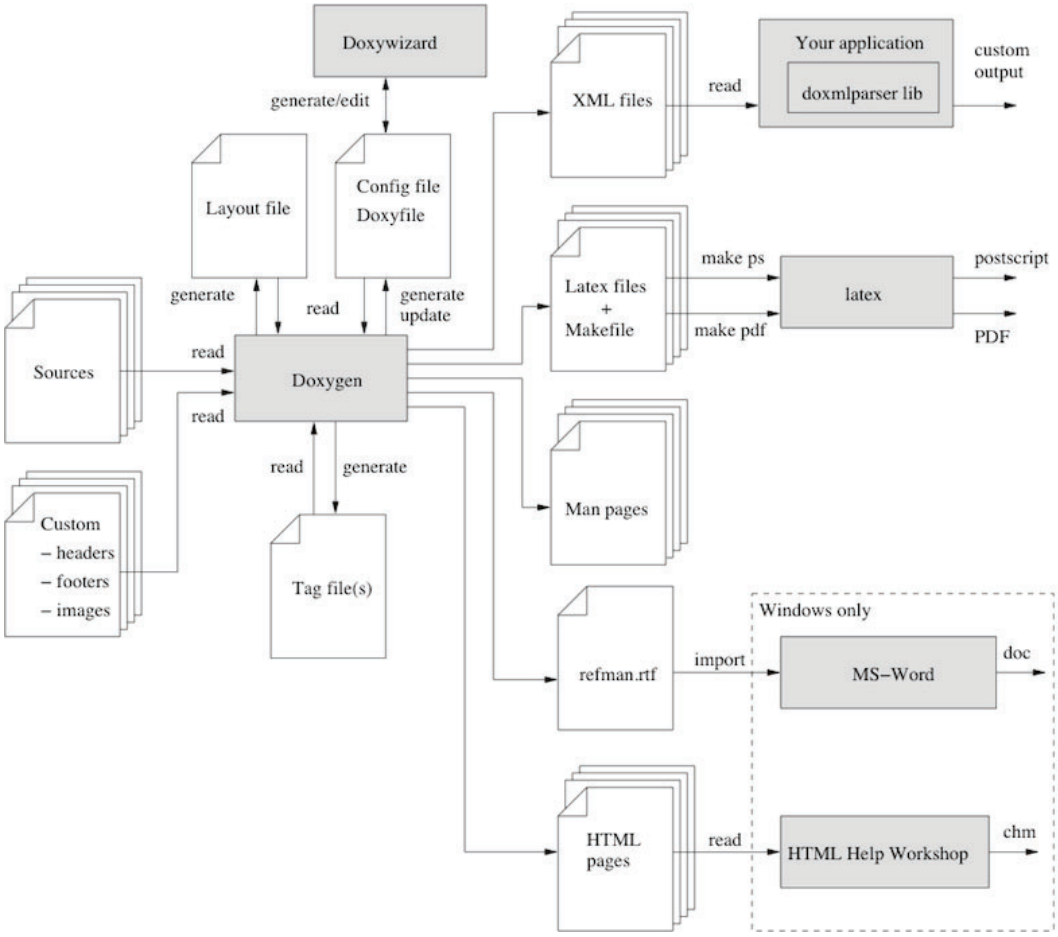


Figura 23: Flujos de información en Doxygen [31].

4.4.2.4 Crear un archivo de configuración Doxyfile en Doxywizard

Para la creación y configuración de un archivo Doxyfile en Doxywizard para el código Python a documentar se muestran a continuación algunas etiquetas importantes que se deben especificar al momento de la configuración:

- **Nombre del Proyecto:** "Nombre de tu Proyecto"
- **Número de Versión:** "1.0"
- **Directorio de Salida:** /ruta/a/tu/carpeta/de/salida
- **Idioma de la Documentación:** Español
- **Optimizar salida para JAVA:** YES
- **Directorio de Códigos Python:** /ruta/a/tu/codigo/python
- **Tipos de Archivos a Documentar (Python):** *.py
- **Mostrar Código Fuente en Línea:** YES
- **Extraer Todo (Incluso sin Documentación):** YES
- **Generar Documentación en formatos:**
  - HTML: YES
  - RTF: YES

- LaTeX: YES

Obtención de la documentación

Habiendo configurado el archivo Doxyfile, se puede obtener la documentación deseada ejecutando «Run doxygen» en la pestaña «Run» de Doxywizard.

Dependiendo de la configuración, Doxygen creará directorios en formato HTML, RTF, LaTeX, XML, páginas de manual Unix y DocBook dentro del directorio de salida especificado. Como se ha seleccionado formatos HTML, RTF y LaTeX, se crean directorios con dichos nombres en los que se encuentra la documentación obtenida para cada formato.

La documentación HTML generada se puede visualizar al abrir un navegador web y acceder al archivo index.html en el directorio HTML. Para obtener los mejores resultados, se recomienda utilizar un navegador que admita hojas de estilo en cascada (CSS). Se han probado navegadores como Mozilla Firefox y Google Chrome para verificar la salida generada.

5. Experimentación y resultados

En este capítulo, se presentarán los resultados obtenidos de la implementación y experimentación llevados a cabo en este estudio dedicado a la robótica quirúrgica y la interfaz multimodal para el control del robot KUKA LBR iiwa 7 R800 con fines quirúrgicos a través de ROS. En este capítulo se evalúan las capacidades de la interfaz multimodal primeramente de manera simulada y, posteriormente, en un entorno de experimentación in-vitro.

Los resultados obtenidos en esta etapa proporcionan una evaluación integral de la eficacia y la viabilidad de la interfaz, centrándose en su capacidad para realizar movimientos de pivote alrededor de un punto de fulcro.

Por otra parte, es importante destacar que para esta fase de la investigación, se ha desarrollado una simplificación de una herramienta quirúrgica específica para llevar a cabo las pruebas y evaluaciones. Además de usarse para el modo de simulación, esta herramienta ha sido diseñada e impresa en 3D para el modo real con dos variantes de la misma, una con una longitud de 0.236 metros y otra con una longitud de 0.186 metros. La herramienta en cuestión se observa en la Figura 24.

Figura 24: Simplificación de una herramienta quirúrgica diseñada para la experimentación.

Los códigos creados durante la ejecución de este proyecto, junto con algunos vídeos que presentan las operaciones y funcionalidades de la interfaz para alcanzar el objetivo deseado, se pueden obtener y revisar en el siguiente enlace: TFM - Javier Lara Juárez.

5.1. Experimentación y resultados en el robot en modo simulación

Esta etapa inicial, se enfoca en la experimentación en un entorno simulado (Gazebo) para evaluar la interfaz multimodal del robot KUKA LBR iiwa 7 R800 con fines quirúrgicos controlado a través de ROS. Las simulaciones desempeñan un papel esencial en la investigación de la robótica quirúrgica, permitiendo realizar pruebas sin los riesgos asociados del mundo real.

A continuación se van a detallar los procedimientos de prueba, métricas y resultados, sentando las bases para su futura aplicación en un entorno real. Estos resultados van a respaldar la transición a la experimentación en el robot real.

5.1.1. Prueba de arranque del modo de simulación

Objetivo

Verificar si la interfaz es capaz de iniciar el modo de simulación y cargar el robot en Gazebo correctamente con la herramienta requerida.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté correctamente configurada para el modo de simulación, es decir, el archivo `iiwa_surgery_params.yaml` debe tener el campo `simulation_mode` establecido en «true».
2. **Ejecución de la interfaz:** Ejecución del archivo `iiwa_surgery.launch` que carga los parámetros y lanza Gazebo mediante el comando → `roslaunch iiwa_surgery iiwa_surgery.launch simulation_mode:=true`.
3. **Observación de la interfaz:** Comprobación de que la interfaz inicia el modo de simulación y de que el robot es cargado en el entorno de Gazebo.
4. **Verificación de la herramienta:** Comprobación de que el robot esté equipado con la herramienta deseada.

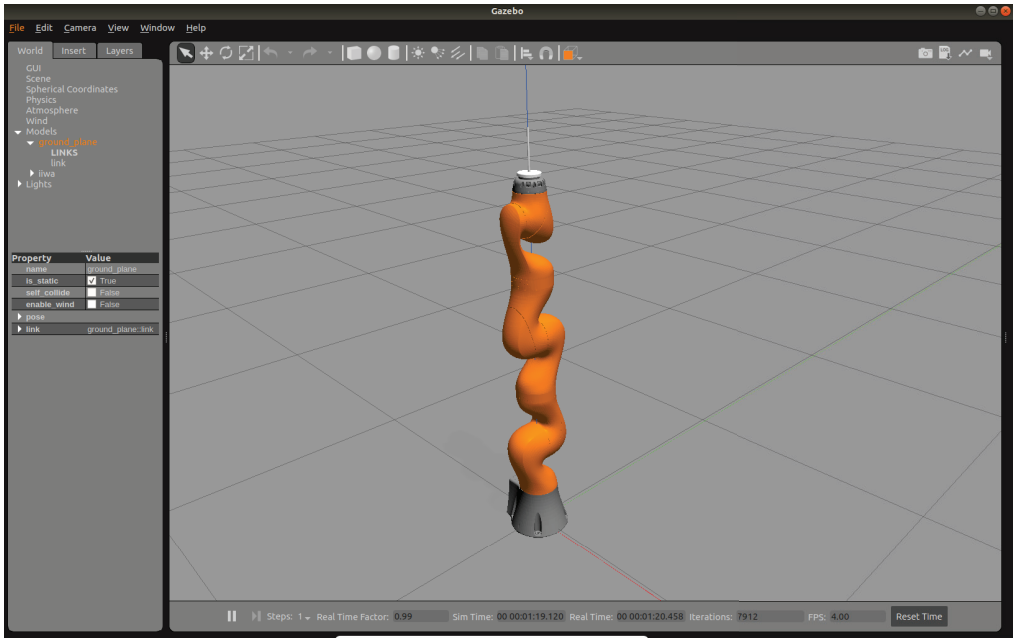


Figura 25: Prueba de arranque del modo de simulación.

Resultados

1. La interfaz inicia el modo de simulación correctamente sin errores.
2. El entorno de Gazebo se carga sin problemas y el robot está presente en la simulación.
3. El robot está equipado con la herramienta deseada.

Análisis de los resultados

Al completarse la prueba con éxito, significa que la interfaz es capaz de iniciar el modo de simulación y cargar el robot en Gazebo de manera adecuada como se demuestra en la Figura 25.

5.1.2. Prueba de movimiento articular

Objetivo

Verificar que la interfaz es capaz de recibir comandos de posición articular, procesarlos y transmitirlos al robot. Luego, confirmar que el robot se ha movido a la configuración articular deseada y que la información de configuración articular proporcionada por la interfaz coincide con la configuración enviada.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté en el modo de simulación (`simulation_mode` establecido en «true») y que el robot se haya cargado correctamente en Gazebo. Verificar que la herramienta esté adjunta al robot y su longitud coincida con el valor especificado en el archivo de configuración.

2. **Envío de mensaje de posición articular:** A través del topic `/iiwa_surgery/command/joints` creado por la interfaz y mediante la consola de comandos, publicar un mensaje de tipo `iiwa_msgs/JointPosition` con la siguiente información de configuración articular:

   - $a_1 = 0.0$
   - $a_2 = 0.9$
   - $a_3 = 0.5$
   - $a_4 = 0.8$
   - $a_5 = 0.7$
   - $a_6 = -0.5$
   - $a_7 = 0.0$

3. **Monitoreo de la configuración articular:** Para observar la información de configuración articular del robot en tiempo real, utilizar el comando → `rostopic echo /iiwa_surgery/output/joints`. Esta información proporcionada por la interfaz después de que el robot haya completado su movimiento debe ser igual a la configuración articular enviada por el operador al comienzo del proceso.

4. **Verificación de resultados:** Verificar que la información de configuración articular reportada en el topic `/iiwa_surgery/output/joints` coincida con la configuración articular enviada a la interfaz. Comprobar además que el robot haya alcanzado la configuración articular deseada usando el comando → `rostopic echo /iiwa/joint_states`. Entre la información generada por este comando se encuentra la información articular del manipulador en tiempo real proporcionada por el robot.

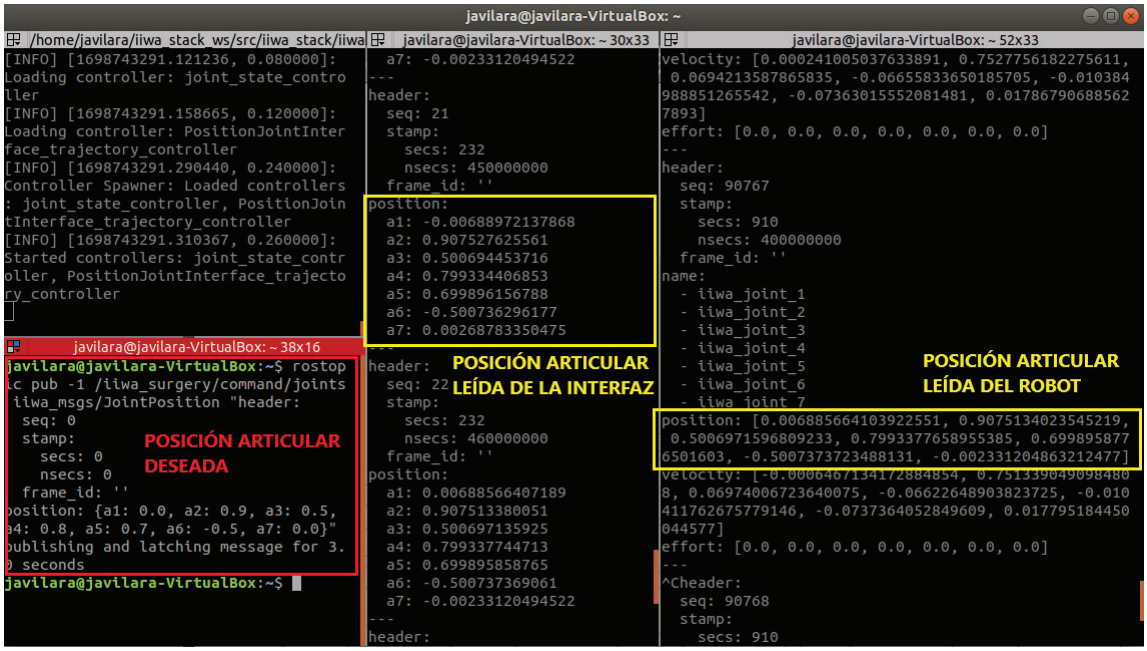


Figura 26: Prueba de movimiento articular en modo simulación.

Resultados

1. La información de configuración articular publicada en `/iiwa_surgery/output/joints` una vez que el manipulador ha terminado su movimiento, coincide con la configuración articular enviada a la interfaz a través del topic `/iiwa_surgery/command/joints`. Esto se observa en la Figura 26 en el centro de la misma, al emplear el comando → `rostopic echo /iiwa_surgery/output/joints` una vez que el brazo robótico ha terminado su movimiento.
2. El robot ha alcanzado la configuración articular deseada. Esto se observa en la Figura 26 a la derecha de la misma, al emplear el comando → `rostopic echo /iiwa/joint_states` una vez que el brazo robótico ha terminado su movimiento.

Análisis de resultados

Al completarse la prueba con éxito y al coincidir los resultados con las expectativas, esto indica que la interfaz es capaz de recibir comandos de posición articular, transmitirlos al robot de manera correcta y que el robot puede moverse a la configuración articular que se desee.

5.1.3. Prueba de movimiento cartesiano libre

Objetivo

Verificar que la interfaz es capaz de recibir comandos de posición cartesiana relacionados con el Tool Center Point, procesarlos y transmitirlos al robot simulado. Luego, confirmar que el robot se ha movido a la posición cartesiana deseada y que la información de posición y orientación proporcionada por la interfaz coincide con la información enviada.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté en el modo de simulación (`simulation_mode` establecido en «true») y que el robot se haya cargado correctamente en Gazebo. Verificar que la herramienta esté adjunta al robot y su longitud (0.186 m) coincida con el valor especificado en el archivo de configuración. Por último, comprobar que el robot esté trabajando en modo libre (`work_mode` establecido en «free»).
2. **Envío de mensaje de posición:** A través del topic `/iiwa_surgery/command/pose` creado por la interfaz y mediante la consola de comandos, publicar un mensaje de tipo `geometry_msgs/PoseStamped` con la siguiente información:
   - **Posición:** $x = 0.4$, $y = 0.0$, $z = 1.0$
   - **Orientación (cuaternio):** $x = 0.0$, $y = 0.0$, $z = 0.0$, $w = 1.0$

   Este mensaje tiene como objetivo desplazar el TCP del robot hasta la posición y orientación especificadas.
3. **Monitoreo de posición del TCP:** Para observar la posición y orientación del TCP del robot en tiempo real, usar el comando → `rostopic echo /iiwa_surgery/output/tcp_pose`. Esta información proporcionada por la interfaz después de que el robot haya completado su movimiento debe ser igual a la configuración enviada por el operador al comienzo del proceso.
4. **Monitoreo de posición del efector final (EF):** También es posible observar la posición y orientación del EF del robot en tiempo real usarndo el comando → `rostopic echo /iiwa_surgery/output/ef_pose`.
5. **Verificación de resultados:** Verificar que la información de posición y orientación reportada en el topic `/iiwa_surgery/output/tcp_pose` coincida con la configuración enviada a la interfaz. Comprobar además que el robot haya alcanzado la configuración deseada. Para ello, y puesto que el nodo del robot no genera ningún topic que reporte el estado en tiempo real del TCP y como la orientación del TCP es la misma que la del EF, se compara la información de los topics `/iiwa_surgery/output/ef_pose` (creado por la interfaz) y `/iiwa/state/CartesianPose` (creado por el robot) una vez el manipulador haya terminado su movimiento. Estos topics reportan la información relativa a la posición y orientación en tiempo real del EF, debiendo ser la información de ambos iguales.

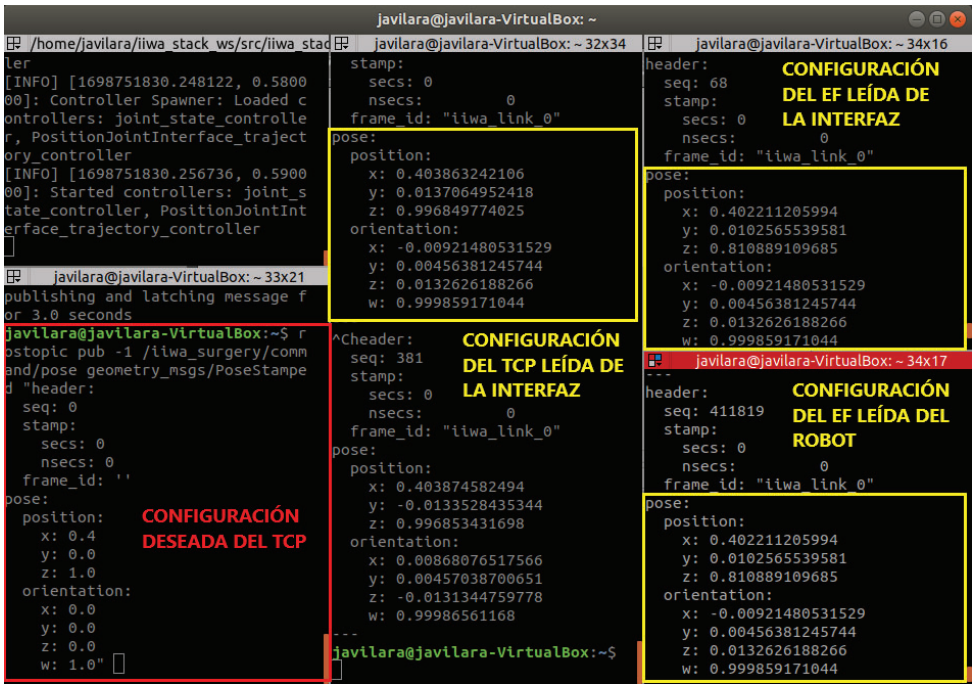


Figura 27: Prueba de movimiento cartesiano libre en modo simulación.

Resultados

1. La posición y orientación del TCP reportada en `/iiwa_surgery/output/tcp_pose` coincide con la posición y orientación enviada en el mensaje de posición al robot a través del topic `/iiwa_surgery/command/pose` una vez que el manipulador haya terminado su desplazamiento. Esto se observa en la Figura 27 en el centro de la misma, al emplear el comando → `rostopic echo /iiwa_surgery/output/tcp_pose` una vez que el brazo robótico ha terminado su movimiento.

2. El robot ha alcanzado la posición deseada. Esto se observa en la Figura 27 a la derecha de la misma, al emplear el comando → `rostopic echo /iiwa_surgery/output/ef_pose` (arriba a la derecha) y el comando → `rostopic echo /iiwa/state/CartesianPose` (abajo a la derecha) una vez que el brazo robótico ha terminado su movimiento. Las posiciones y orientaciones del EF que reportan ambos topics son iguales.

Análisis de resultados

Al completarse la prueba con éxito y al coincidir los resultados con las expectativas, esto indica que la interfaz es capaz de recibir comandos de posición cartesiana, transmitirlos al robot y que el robot puede moverse a la posición y orientación deseada en el modo de trabajo libre.

5.1.4. Prueba de movimiento cartesiano alrededor de un punto de fulcro

Objetivo

Verificar que la interfaz es capaz de recibir comandos de posición cartesiana relacionados con el Tool Center Point, procesarlos y transmitirlos al robot simulado. Luego, confirmar que el robot se ha movido adecuadamente siguiendo las restricciones alrededor del punto de fulcro independientemente de la longitud de la herramienta.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté en el modo de simulación (`simulation_mode` establecido en «true») y que el robot se haya cargado correctamente en Gazebo. Verificar que la herramienta esté adjunta al robot y que su longitud coincida con el valor especificado en el archivo de configuración (0.236

m o 0.186 m para esta prueba según corresponda). Definir el punto de fulcro en el archivo de configuración (`fulcrum_fi` igual a 0.5 para esta prueba, situando así el punto de fulcro en mitad de la herramienta). Confirmar que el robot esté en trabajando en modo de pivote (`work_mode` establecido en «pivot»).

2. **Generación de puntos:** A través del topic `/iiwa_surgery/command/pose` creado por la interfaz y mediante la consola de comandos, publicar mensajes de tipo `geometry_msgs/PoseStamped` con la información de posición y orientación deseadas del TCP. Este mensaje tiene como objetivo desplazar el TCP del robot hasta la configuración especificada. El proceso comienza con el primer mensaje recibido por la interfaz a través de este topic, el cual establece el punto de fulcro. Para los puntos subsiguientes que se quieran trasmitir a la interfaz para generar el movimiento de pivote, solo es necesario especificar la posición del TCP, ya que la orientación será calculada por la interfaz.
3. **Ejecución de pruebas:** Generar una secuencia de puntos del TCP para el movimiento alrededor del punto de fulcro. Realizar la prueba para varios conjuntos de incrementos de posición (fijándose para esta caso en incrementos de 0.1 m, 0.005 m y 0.001 m para la herramienta larga y de 0.001 m para la herramienta corta). Durante la ejecución de la prueba, observar el movimiento del robot en el simulador y registrar los datos publicados en tiempo real por la interfaz en los topics `/iiwa_surgery/output/ef_pose` y `/iiwa_surgery/output/tcp_pose` en un archivo de registro ROS Bag.
4. **Análisis de datos:** Utilizar los datos recopilados para generar gráficas 3D que representen la trayectoria del robot alrededor del punto de fulcro. Además, generar gráficas 2D en ejes XZ e YZ para un análisis más detallado.
5. **Verificación de resultados:** Observar las gráficas generadas y verificar si el robot realiza adecuadamente el movimiento alrededor del punto de fulcro. Prestar atención a la separación de la herramienta con respecto al punto de fulcro y cómo varía con diferentes incrementos de posición.

Gráficas para la herramienta de 0.236 m con incrementos de 0.1 m:

Inicialmente, se generan representaciones gráficas que reflejan los desplazamientos ejecutados por el robot, considerando incrementos de posición de 0.1 m y una longitud de herramienta de 0.236 m. Con el propósito de facilitar la observación detallada de estos movimientos, se configura una representación tridimensional (Figura 28) junto con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 29). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

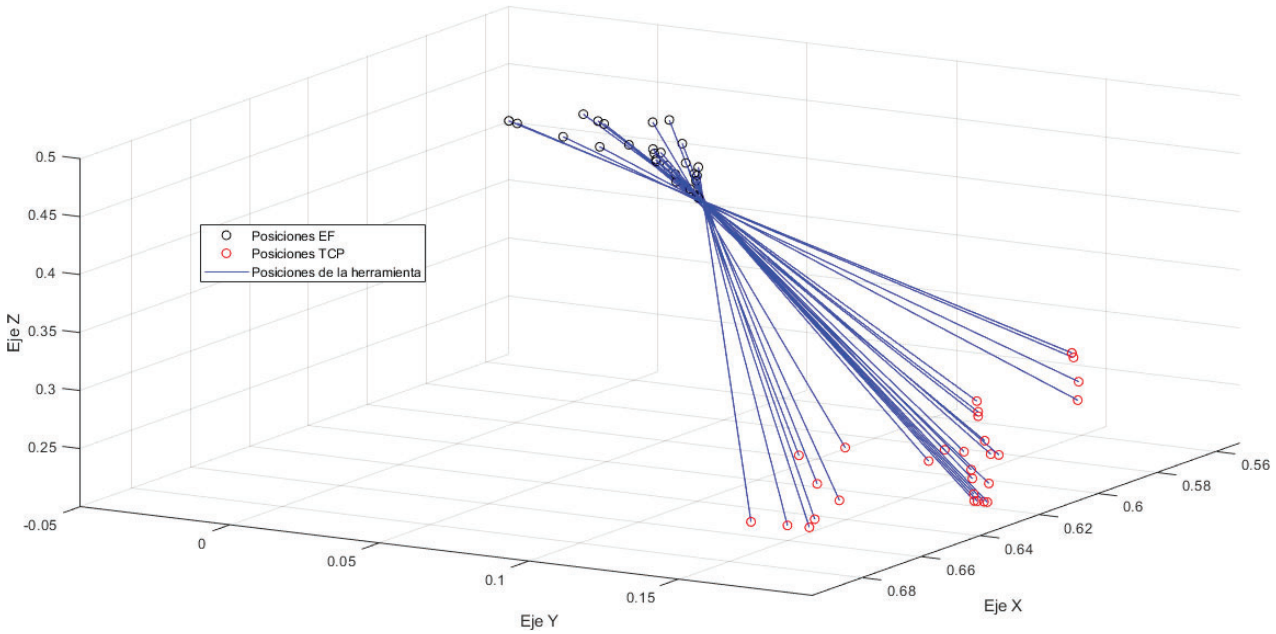
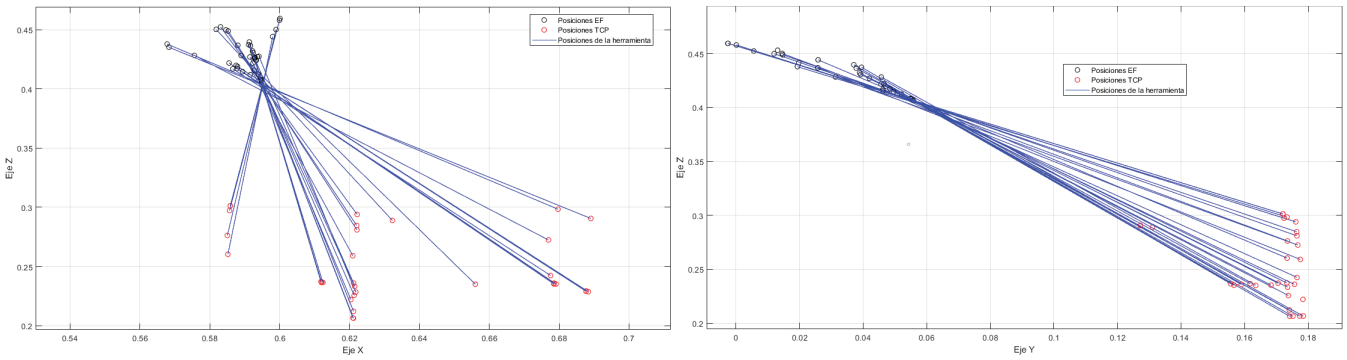


Figura 28: Gráfica 3D con herramienta de 0.236 m e incrementos de 0.1 m en modo simulación.



(a) Plano XZ (b) Plano YZ

Figura 29: Gráficas 2D con herramienta de 0.236 m e incrementos de 0.1 m en modo simulación.

A continuación, se aborda la evaluación de los errores cometidos con respecto al punto de fulcro de los movimientos anteriores, donde se ha generado para ello una gráfica que visualiza los errores para cada una de las 10 posiciones comandadas y la transición entre ellas. Esta representación, ilustrada en la Figura 30, permite identificar de manera clara las discrepancias entre la posición real y la posición deseada, destacando cualquier desviación con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido es aproximadamente de 4 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

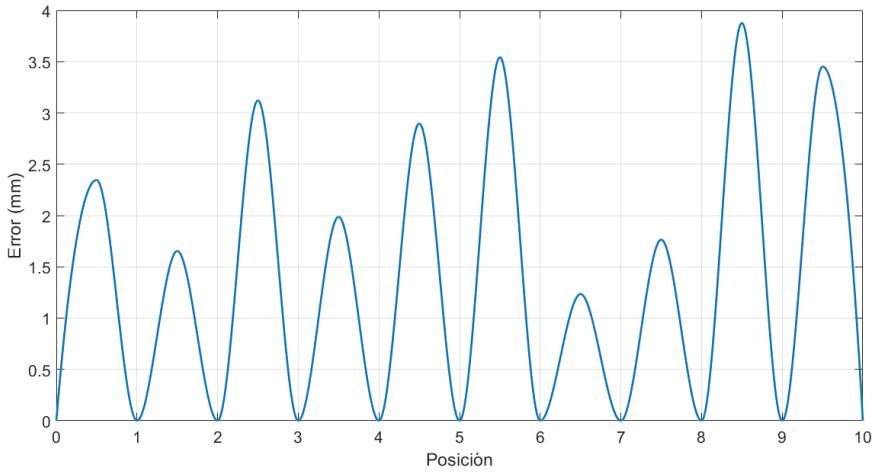


Figura 30: Errores por posición con herramienta de 0.236 m e incrementos de 0.1 m en modo simulación.

Posteriormente, se ha elaborado una tabla detallada que presenta el error medio cometido en cada desplazamiento entre posiciones consecutivas, proporcionando una visión más cuantitativa de las discrepancias. Esta tabla ofrece una perspectiva detallada del rendimiento del robot en términos de precisión, permitiendo una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa en 2.423479 mm.

Posición	Error medio (mm)
$0 \rightarrow 1$	1.472621
$1 \rightarrow 2$	1.034367
$2 \rightarrow 3$	1.952345
$3 \rightarrow 4$	1.242626
$4 \rightarrow 5$	1.811396
$5 \rightarrow 6$	2.214924
$6 \rightarrow 7$	0.772021
$7 \rightarrow 8$	1.103302
$8 \rightarrow 9$	2.423479
$9 \rightarrow 10$	2.148148

Cuadro 1: Errores medios con herramienta de 0.236 m e incrementos de 0.1 m en modo simulación.

Gráficas para la herramienta de 0.236 m con incrementos de 0.005 m:

Habiendo realizado la prueba para incrementos de 0.1 m, ahora se repite para incrementos de posición de 0.005 m.

Se generan representaciones gráficas que capturan los desplazamientos del robot, considerando incrementos de posición ahora establecidos en 0.005 m, manteniendo una longitud de herramienta de 0.236 m. Con el fin de propiciar una observación minuciosa de estos movimientos, se configura una representación tridimensional (Figura 31), complementada con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 32). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

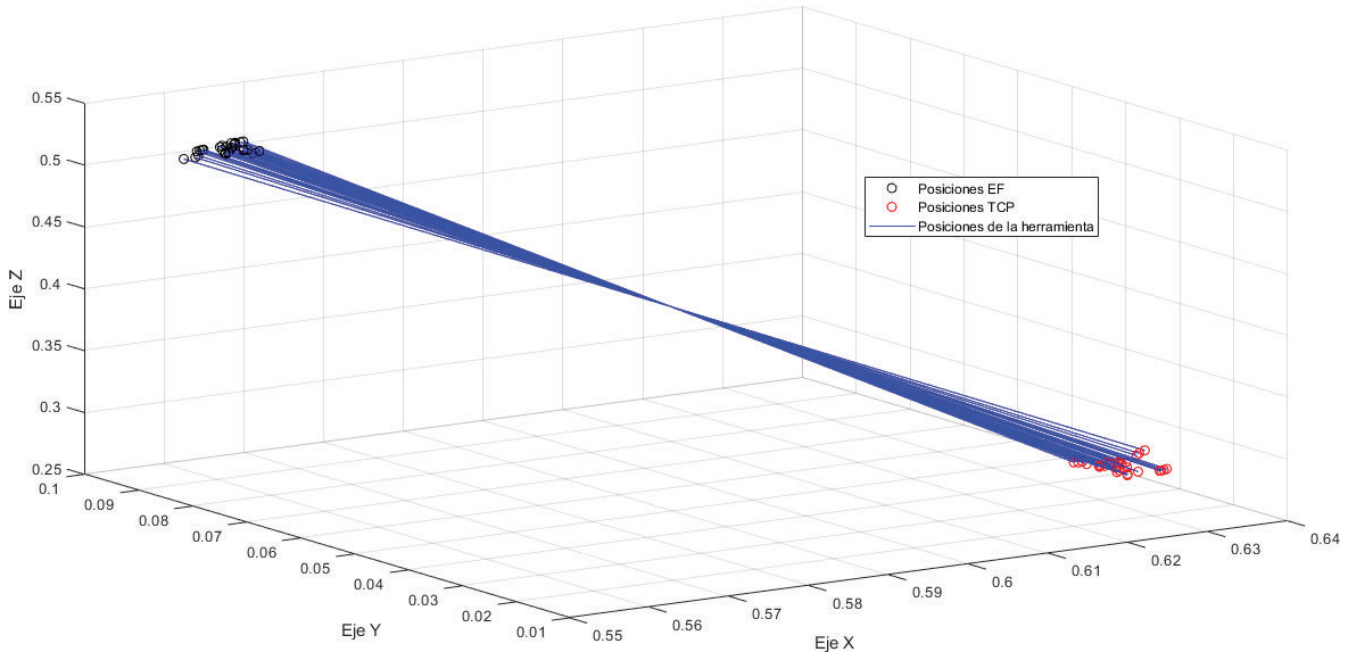
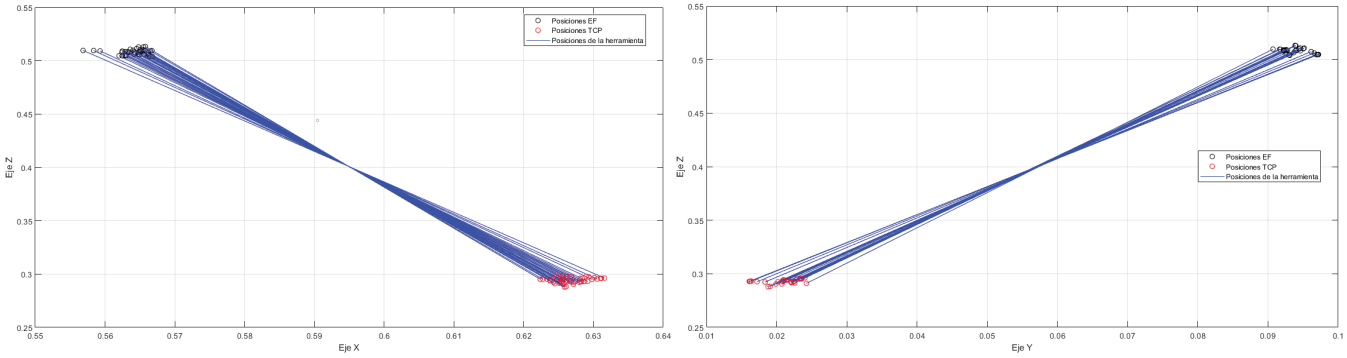


Figura 31: Gráfica 3D con herramienta de 0.236 m e incrementos de 0.005 m en modo simulación.



(a) Plano XZ (b) Plano YZ

Figura 32: Gráficas 2D con herramienta de 0.236 m e incrementos de 0.005 m en modo simulación.

Como anteriormente, se procede a la evaluación de los errores asociados al punto de fulcro en los movimientos realizados. Para ello, se ha creado una gráfica específica que presenta los errores en cada una de las 10 posiciones comandadas, así como en las transiciones entre ellas. La representación visual de estos errores, visualizable en la Figura 33, ofrece una clara identificación de las discrepancias entre la posición efectiva y la posición deseada, resaltando las desviaciones con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido es aproximadamente de 0.35 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

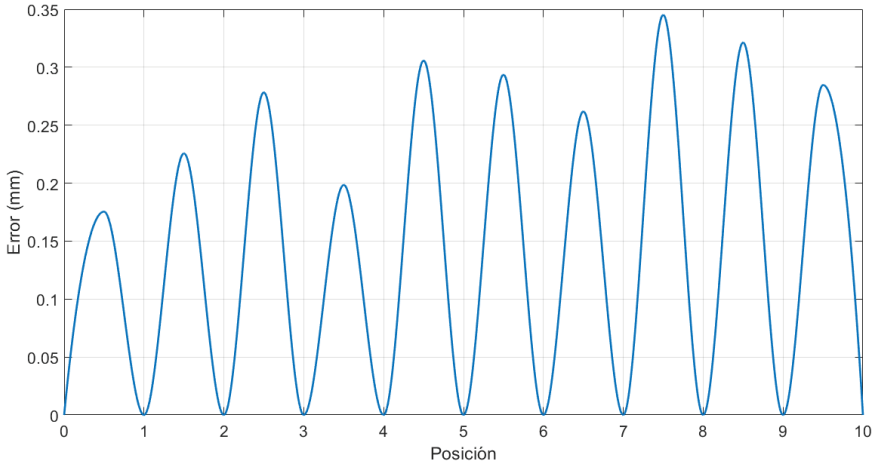


Figura 33: Errores por posición con herramienta de 0.236 m e incrementos de 0.005 m en modo simulación.

En una etapa posterior, se ha generado una tabla exhaustiva que resume el error medio en cada desplazamiento entre posiciones consecutivas, brindando una perspectiva cuantitativa más detallada de las discrepancias y proporcionando una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa ahora en 0.215755 mm.

Posición	Error medio (mm)
$0 \rightarrow 1$	0.109676
$1 \rightarrow 2$	0.141087
$2 \rightarrow 3$	0.173946
$3 \rightarrow 4$	0.124079
$4 \rightarrow 5$	0.191114
$5 \rightarrow 6$	0.183512
$6 \rightarrow 7$	0.163671
$7 \rightarrow 8$	0.215755
$8 \rightarrow 9$	0.200911
$9 \rightarrow 10$	0.177933

Cuadro 2: Errores medios con herramienta de 0.236 m e incrementos de 0.005 m en modo simulación.

Gráficas para la herramienta de 0.236 m con incrementos de 0.001 m:

Habiendo realizado la prueba para incrementos de 0.1 m y 0.005 m, se repite una última vez la prueba para la herramienta de 0.236 m en este caso con incrementos de posición de 0.001 m.

Se generan las gráficas que registran los desplazamientos del robot, considerando incrementos de posición ahora establecidos en 0.001 m, manteniendo una longitud de herramienta de 0.236 m. Con el fin de propiciar una observación minuciosa de estos movimientos, se configura una representación tridimensional (Figura 34), complementada con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 35). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

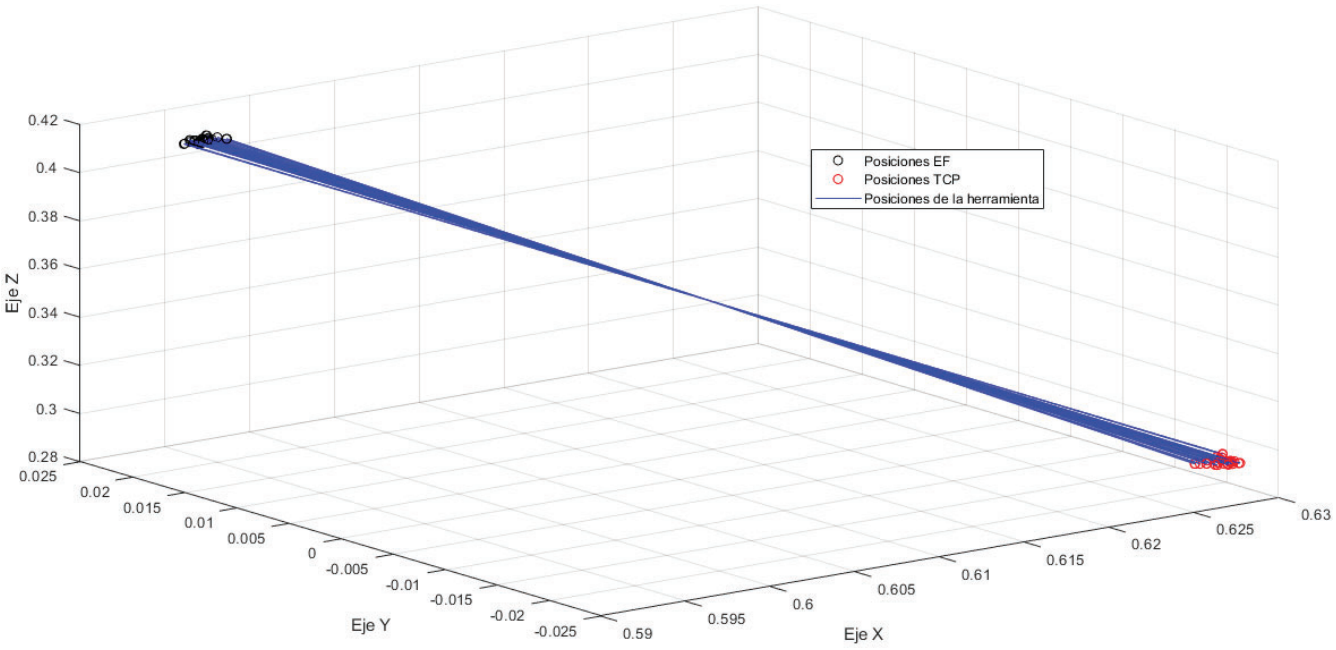
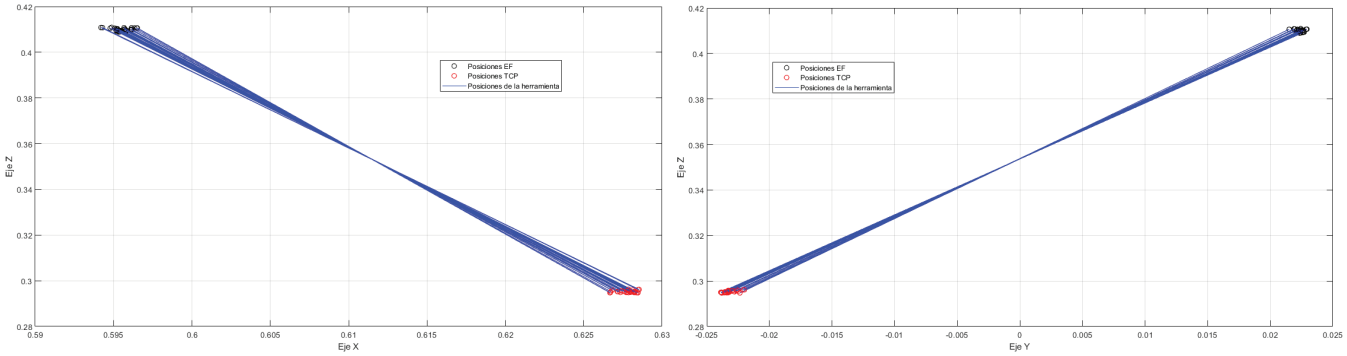


Figura 34: Gráfica 3D con herramienta de 0.236 m e incrementos de 0.001 m en modo simulación.



(a) Plano XZ (b) Plano YZ

Figura 35: Gráficas 2D con herramienta de 0.236 m e incrementos de 0.001 m en modo simulación.

Como anteriormente, se procede a la evaluación de los errores asociados al punto de fulcro en los movimientos realizados. Para ello, se ha creado una gráfica específica que presenta los errores en cada una de las 10 posiciones comandadas, así como en las transiciones entre ellas. La representación visual de estos errores, visualizable en la Figura 36, ofrece una clara identificación de las discrepancias entre la posición efectiva y la posición deseada, resaltando las desviaciones con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido es aproximadamente de 0.2 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

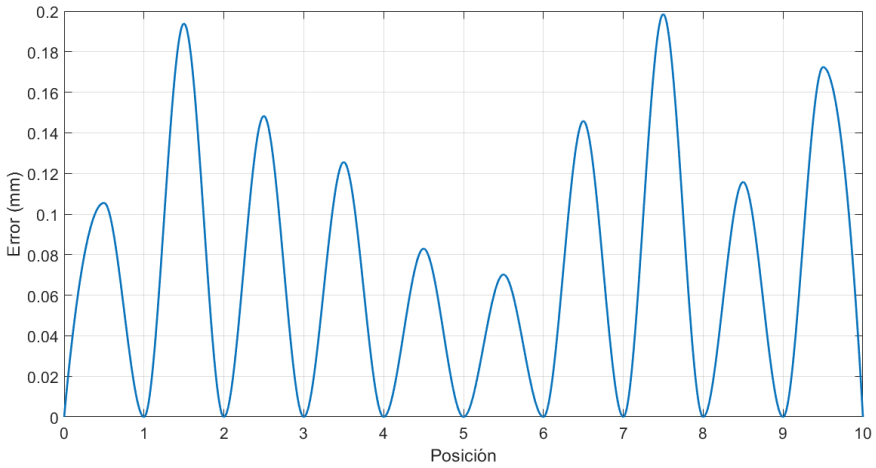


Figura 36: Errores por posición con herramienta de 0.236 m e incrementos de 0.001 m en modo simulación.

Posteriormente, se ha generado una tabla exhaustiva que resume el error medio en cada desplazamiento entre posiciones consecutivas, brindando una perspectiva cuantitativa más detallada de las discrepancias y proporcionando una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa en 0.124039 mm.

Posición	Error medio (mm)
$0 \rightarrow 1$	0.065926
$1 \rightarrow 2$	0.121173
$2 \rightarrow 3$	0.092682
$3 \rightarrow 4$	0.078504
$4 \rightarrow 5$	0.051839
$5 \rightarrow 6$	0.043887
$6 \rightarrow 7$	0.091142
$7 \rightarrow 8$	0.124039
$8 \rightarrow 9$	0.072343
$9 \rightarrow 10$	0.107749

Cuadro 3: Errores medios con herramienta de 0.236 m e incrementos de 0.001 m en modo simulación.

Gráficas para la herramienta de 0.186 m con incrementos de 0.001 m:

Por último, se modifica la longitud de la herramienta a 0.186 m y se repite la prueba una última vez para incrementos de posición de 0.001 m.

Se generan las gráficas que registran los desplazamientos del robot, considerando incrementos de posición ahora establecidos en 0.001 m, con una longitud de herramienta de 0.186 m. Con el fin de propiciar una observación minuciosa de estos movimientos, se configura una representación tridimensional (Figura 37), complementada con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 38). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

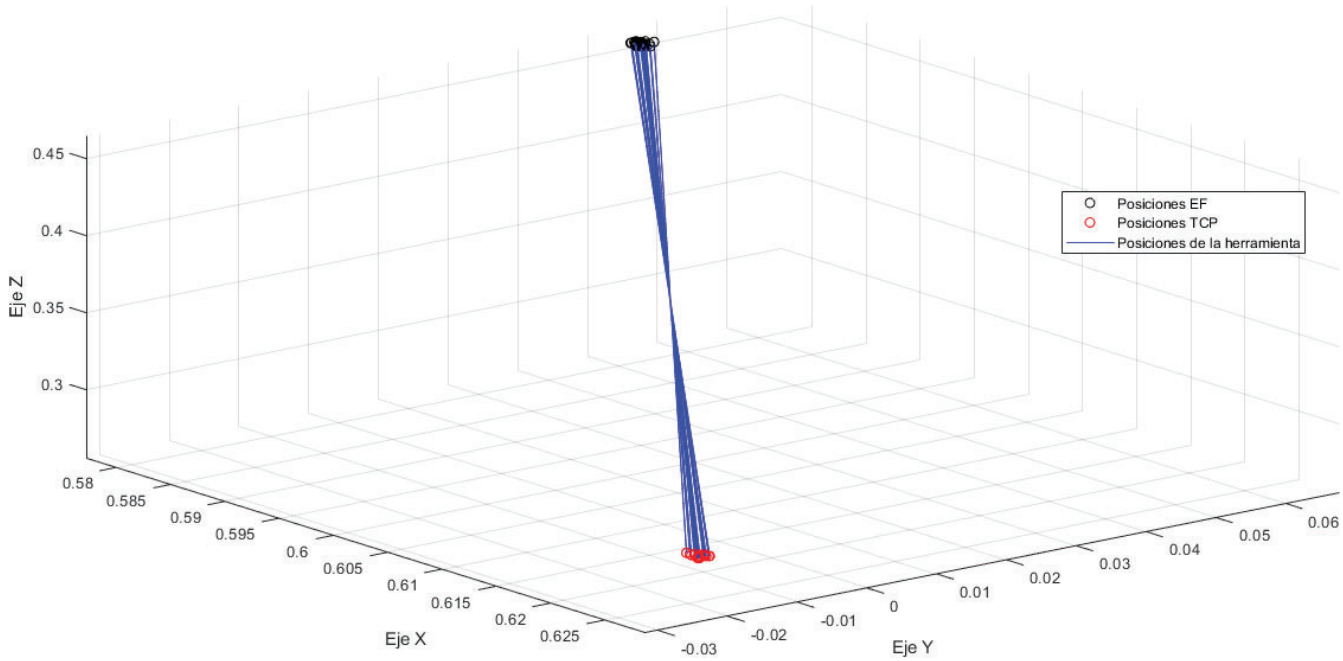
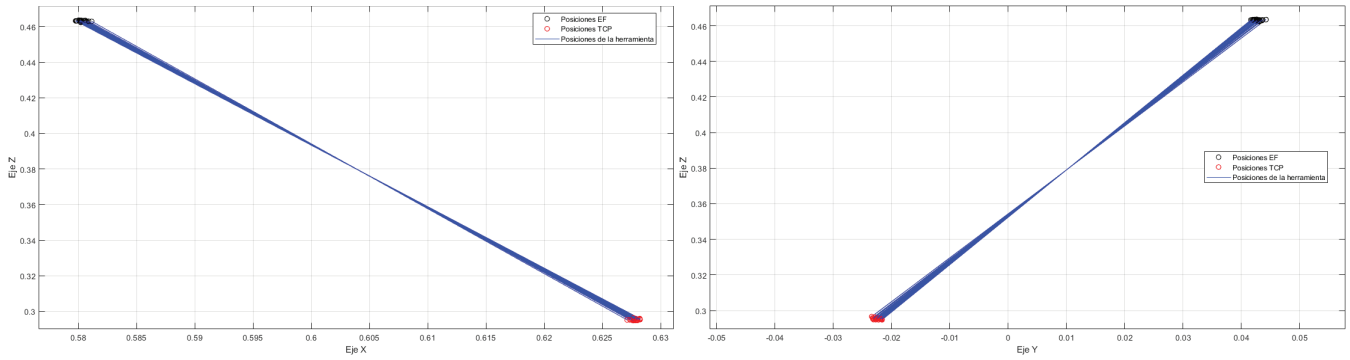


Figura 37: Gráfica 3D con herramienta de 0.186 m e incrementos de 0.001 m en modo simulación.



(a) Plano XZ (b) Plano YZ

Figura 38: Gráficas 2D con herramienta de 0.186 m e incrementos de 0.001 m en modo simulación.

Como ya se hizo para el anterior caso, se procede a la evaluación de los errores asociados al punto de fulcro en los movimientos realizados. Para ello, se ha creado una gráfica específica que presenta los errores en cada una de las 10 posiciones comandadas, así como en las transiciones entre ellas. La representación visual de estos errores, visualizable en la Figura 39, ofrece una clara identificación de las discrepancias entre la posición efectiva y la posición deseada, resaltando las desviaciones con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido está en torno a 0.21 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

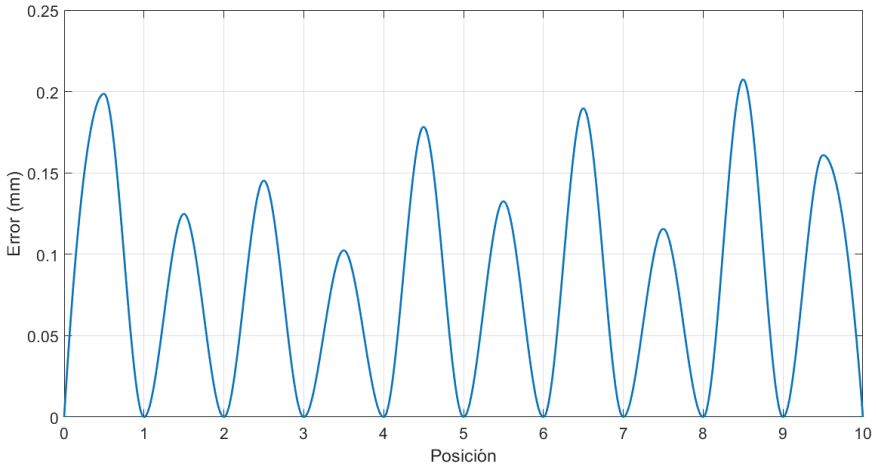


Figura 39: Errores por posición con herramienta de 0.186 m e incrementos de 0.001 m en modo simulación.

Posteriormente, se ha generado una tabla exhaustiva que resume el error medio en cada desplazamiento entre posiciones consecutivas, brindando una perspectiva cuantitativa más detallada de las discrepancias y proporcionando una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa en 0.129636 mm.

Posición	**Error medio (mm)**
$0 \rightarrow 1$	0.124138
$1 \rightarrow 2$	0.078023
$2 \rightarrow 3$	0.090793
$3 \rightarrow 4$	0.064047
$4 \rightarrow 5$	0.111441
$5 \rightarrow 6$	0.082874
$6 \rightarrow 7$	0.118589
$7 \rightarrow 8$	0.072301
$8 \rightarrow 9$	0.129636
$9 \rightarrow 10$	0.100586

Cuadro 4: Errores medios con herramienta de 0.186 m e incrementos de 0.001 m en modo simulación.

Resultados

1. La interfaz es capaz de generar movimientos del robot alrededor de un punto de fulcro independientemente de la longitud de la herramienta y de los incrementos de posición.
2. Se observa que a medida que se aumentan los incrementos de posición en las pruebas, el área (error) alrededor del punto de fulcro aumenta. Esto significa que el robot se desvía ligeramente de la trayectoria deseada cuando los incrementos de posición son más grandes. Esta desviación es una consecuencia de no generar trayectorias intermedias entre los puntos, lo que resulta en una trayectoria más recta y menos curva alrededor del punto de fulcro.

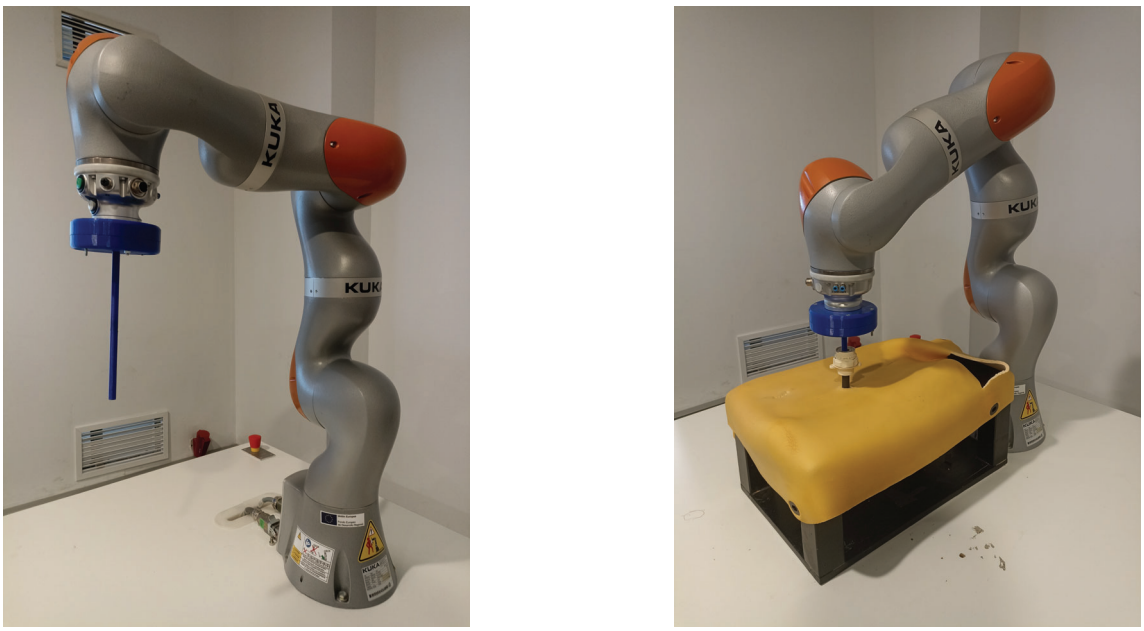
Análisis de resultados

En conclusión, la prueba de movimiento cartesiano alrededor de un punto de fulcro ha sido exitosa, demostrando así la eficacia de la interfaz multimodal para controlar el movimiento del robot simulado alrededor de un punto de fulcro, sea cual sea la longitud de la herramienta.

Por otro lado, la variación en la precisión del movimiento con respecto al tamaño de los incrementos de posición es un hallazgo importante. Esto destaca la importancia de generar trayectorias más completas entre los puntos en lugar de simplemente enviar puntos sueltos. En concreto, estas trayectorias deben descomponerse en un conjunto de puntos con una distancia máxima de 0.001 m, lo que asegura un error lo suficientemente pequeño. Esta meticulosa descomposición garantiza que el robot respete el punto de fulcro durante todo su recorrido, asegurando así la minuciosidad y estabilidad necesarias en los movimientos.

5.2. Experimentación y resultados en el robot real

Tras la validación de la interfaz multimodal en el entorno simulado, se va a explorar ahora la fase de experimentación y obtención de resultados en el entorno del robot real mostrado en la Figura 40. En dicha figura se observa a la izquierda el robot real con la herramienta y a la derecha, el mismo sistema pero añadiendo un pelvitrainer que simula la cavidad abdominal de un paciente y la herramienta introducida a través de un punto de fulcro.



(a) Sin pelvitrainer (b) Con pelvitrainer

Figura 40: Robot físico con herramienta.

5.2.1. Prueba de movimiento articular

Objetivo

Verificar que la interfaz es capaz de recibir comandos de posición articular, procesarlos y transmitirlos al robot. Luego, confirmar que el robot se ha movido a la configuración articular deseada y que la información de configuración articular proporcionada por la interfaz coincide con la configuración enviada.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté en el modo real (`simulation_mode` establecido en «false») y que el manipulador físico esté listo para su uso. Verificar que la herramienta esté adjunta al robot y su longitud coincida con el valor especificado en el archivo de configuración. Por último, comprobar que la interfaz se haya ejecutado mediante el comando → `roslaunch iiwa_surgery iiwa_surgery.launch simulation_mode:=false`.

2. **Envío de mensaje de posición articular:** Mediante el topic `/iiwa_surgery/command/joints` creado por la interfaz y a través de la consola de comandos, publicar un mensaje de tipo `iiwa_msgs/JointPosition` con la siguiente configuración articular: $a_1 = 0.0$, $a_2 = 0.9$, $a_3 = 0.5$, $a_4 = 0.8$, $a_5 = 0.7$, $a_6 = -0.5$, $a_7 = 0.0$

3. **Monitoreo de la configuración articular:** Para observar la información de configuración articular del robot en tiempo real, utilizar el comando → `rostopic echo /iiwa_surgery/output/joints`. Esta información proporcionada por la interfaz después de que el robot haya completado su movimiento debe ser igual a la configuración articular enviada por el operador al comienzo del proceso.

4. **Verificación de resultados:** Verificar que la información de configuración articular reportada en el topic `/iiwa_surgery/output/joints` coincida con la configuración articular enviada a la interfaz. Comprobar

además que el robot haya alcanzado la configuración articular deseada usando el comando → `rostopic echo /iiwa/state/JointPosition`. En la información generada por este comando se encuentra la información articular del manipulador en tiempo real proporcionada por el robot.

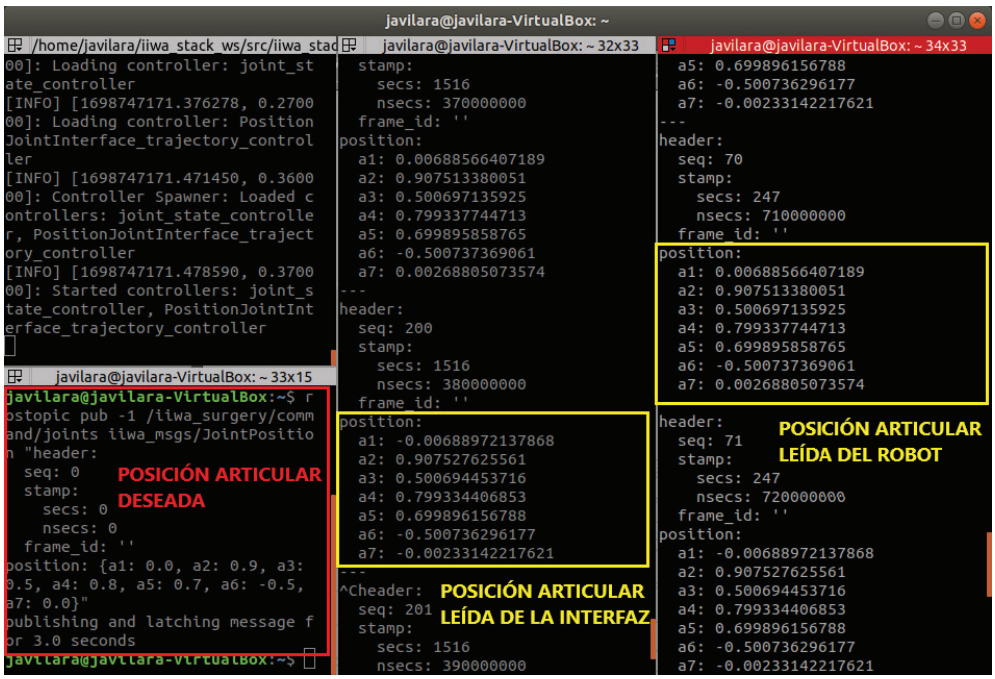


Figura 41: Prueba de movimiento articular en modo real.

Resultados

1. La información de configuración articular publicada en `/iiwa_surgery/output/joints` una vez que el manipulador ha terminado su movimiento, coincide con la configuración articular enviada a la interfaz a través del topic `/iiwa_surgery/command/joints`. Esto se observa en la Figura 41 en el centro de la misma, al emplear el comando → `rostopic echo /iiwa_surgery/output/joints` una vez que el brazo robótico ha terminado su movimiento.

2. El robot ha alcanzado la configuración articular deseada. Esto se observa en la Figura 41 a la derecha de la misma, al emplear el comando → `rostopic echo /iiwa/state/JointPosition` una vez que el brazo robótico ha terminado su movimiento.

Análisis de resultados

Al completarse la prueba con éxito y al coincidir los resultados con las expectativas, esto indica que la interfaz es capaz de recibir comandos de posición articular, transmitirlos al robot real de manera correcta y que el robot puede moverse a la configuración articular deseada.

5.2.2. Prueba de movimiento cartesiano libre

Objetivo

Verificar que la interfaz es capaz de recibir comandos de posición cartesiana relacionados con el Tool Center Point, procesarlos y transmitirlos al robot real. Luego, confirmar que el robot se ha movido a la posición cartesiana deseada y que la información de posición y orientación proporcionada por la interfaz coincide con la información enviada.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté en el modo real (`simulation_mode` establecido en «false») y que el manipulador físico esté listo para su uso. Verificar que la herramienta esté adjunta al robot y su longitud (0.186 m) coincida con el valor especificado en el archivo de configuración. Por último, comprobar que el robot esté trabajando en modo libre (`work_mode` establecido en «free»).

2. **Envío de mensaje de posición:** A través del topic `/iiwa_surgery/command/pose` creado por la interfaz y mediante la consola de comandos, publicar un mensaje de tipo `geometry_msgs/PoseStamped` con la siguiente información:

   - **Posición:** x =0.4, y =0.0, z =1.0
   - **Orientación (cuaternio):** x =0.0, y =0.0, z =0.0, w =1.0

   Este mensaje tiene como objetivo desplazar el TCP del robot hasta la posición y orientación especificadas.

3. **Monitoreo de posición del TCP:** Para observar la posición y orientación del TCP del robot en tiempo real, usar el comando → `rostopic echo /iiwa_surgery/output/tcp_pose`. Esta información proporcionada por la interfaz después de que el robot haya completado su movimiento debe ser igual a la configuración articular enviada por el operador al comienzo del proceso.

4. **Monitoreo de posición del efector final (EF):** También es posible observar la posición y orientación del EF del robot en tiempo real usando el comando → `rostopic echo /iiwa_surgery/output/ef_pose`.

5. **Verificación de resultados:** Verificar que la información de posición y orientación reportada en el topic `/iiwa_surgery/output/tcp_pose` coincida con la configuración enviada a la interfaz. Comprobar además que el robot haya alcanzado la configuración deseada. Para ello, y puesto que el nodo del robot no genera ningún topic que reporte el estado en tiempo real del TCP, se compara la información de los topics `/iiwa_surgery/output/ef_pose` (creado por la interfaz) y `/iiwa/state/CartesianPose` (creado por el robot) una vez el manipulador haya terminado su movimiento. Estos topics reportan la información relativa a la posición y orientación en tiempo real del EF, debiendo ser la información de ambos iguales.

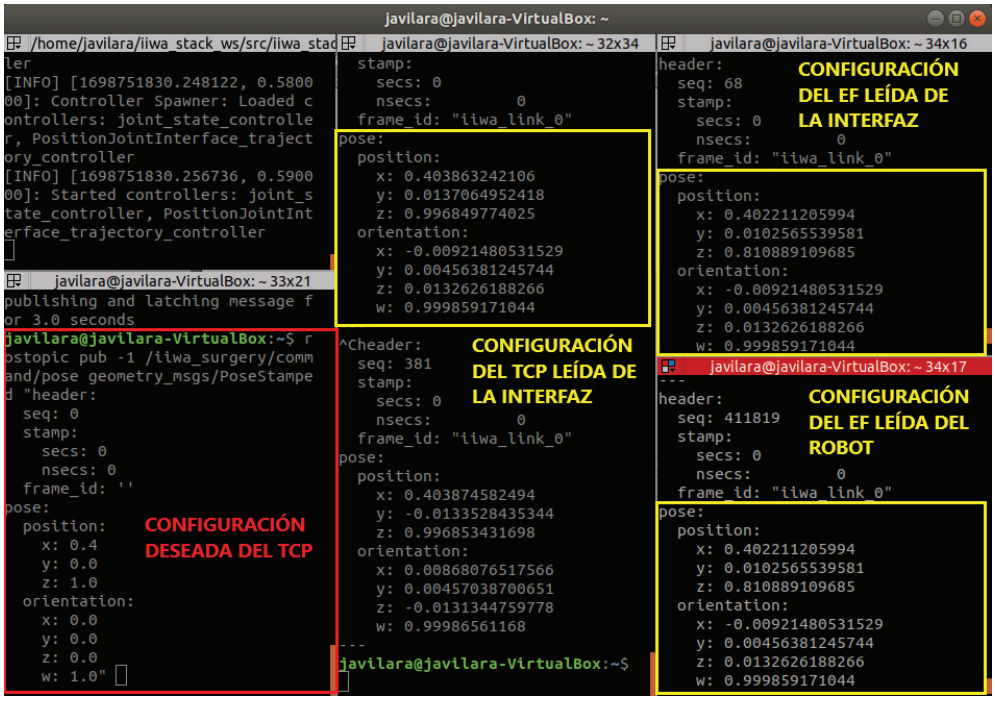


Figura 42: Prueba de movimiento cartesiano libre en modo real.

Resultados

1. La posición y orientación del TCP reportada en `/iiwa_surgery/output/tcp_pose` coincide con la posición y orientación enviada en el mensaje de posición cartesiana al robot a través de topic `/iiwa_surgery/command/pose` una vez que el manipulador haya terminado su desplazamiento. Esto se observa en la Figura 42 en el centro de la misma, al emplear el comando → `rostopic echo /iiwa_surgery/output/tcp_pose` una vez que el brazo robótico ha terminado su movimiento.
2. El robot ha alcanzado la posición deseada. Esto se observa en la Figura 42 a la derecha de la misma, al emplear el comando → `rostopic echo /iiwa_surgery/output/ef_pose` (arriba a la derecha) y el comando → `rostopic echo /iiwa/state/CartesianPose` (abajo a la derecha) una vez que el brazo robótico ha terminado su movimiento. Las posiciones y orientaciones del EF que reportan ambos topics son iguales.

Análisis de resultados

Al completarse la prueba con éxito y al coincidir los resultados con las expectativas, esto indica que la interfaz es capaz de recibir comandos de posición cartesiana, transmitirlos al robot real y que el robot puede moverse a la posición y orientación deseada en el modo de trabajo libre.

5.2.3. Prueba de movimiento cartesiano alrededor de un punto de fulcro

Objetivo

Verificar que la interfaz es capaz de recibir comandos de posición cartesiana relacionados con el Tool Center Point, procesarlos y transmitirlos al robot simulado. Luego, confirmar que el robot se ha movido adecuadamente siguiendo las restricciones alrededor del punto de fulcro independientemente de la longitud de la herramienta.

Procedimiento

1. **Configuración inicial:** Comprobación de que la interfaz esté en el modo de simulación (`simulation_mode` establecido en «true») y que el robot se haya cargado correctamente en Gazebo. Verificar que la herramienta esté adjunta al robot y que su longitud coincida con el valor especificado en el archivo de configuración (0.236 m o 0.186 m para esta prueba según corresponda). Definir el punto de fulcro en el archivo de configuración (`fulcrum_fi` igual a 0.5 para esta prueba, situando así el punto de fulcro en mitad de la herramienta). Confirmar que el robot esté en trabajando en modo de pivote (`work_mode` establecido en «pivot»).
2. **Generación de puntos:** A través del topic `/iiwa_surgery/command/pose` creado por la interfaz y mediante la consola de comandos, publicar mensajes de tipo `geometry_msgs/PoseStamped` con la información de posición y orientación deseadas del TCP. Este mensaje tiene como objetivo desplazar el TCP del robot hasta la configuración especificada. El proceso comienza con el primer mensaje recibido por la interfaz a través de este topic, el cual establece el punto de fulcro. Para los puntos subsiguientes que se quieran trasmitir a la interfaz para generar el movimiento de pivote, solo es necesario especificar la posición del TCP, ya que la orientación será calculada por la interfaz.
3. **Ejecución de pruebas:** Generar una secuencia de puntos del TCP para el movimiento alrededor del punto de fulcro. Realizar la prueba para varios conjuntos de incrementos de posición (fijándose para esta caso en incrementos de 0.1 m, 0.005 m y 0.001 m para la herramienta larga y de 0.001 m para la herramienta corta). Durante la ejecución de la prueba, observar el movimiento del robot en el simulador y registrar los datos publicados en tiempo real por la interfaz en los topics `/iiwa_surgery/output/ef_pose` y `/iiwa_surgery/output/tcp_pose` en un archivo de registro ROS Bag.
4. **Análisis de datos:** Utilizar los datos recopilados para generar gráficas 3D que representen la trayectoria del robot alrededor del punto de fulcro. Además, generar gráficas 2D en ejes XZ e YZ para un análisis más detallado.
5. **Verificación de resultados:** Observar las gráficas generadas y verificar si el robot realiza adecuadamente el movimiento alrededor del punto de fulcro. Prestar atención a la separación de la herramienta con respecto al punto de fulcro y cómo varía con diferentes incrementos de posición.

Gráficas para la herramienta de 0.236 m con incrementos de 0.1 m:

Inicialmente, se generan representaciones gráficas que reflejan los desplazamientos ejecutados por el robot, considerando incrementos de posición de 0.1 m y una longitud de herramienta de 0.236 m. Con el propósito de facilitar la observación detallada de estos movimientos, se configura una representación tridimensional (Figura 43) junto con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 44). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

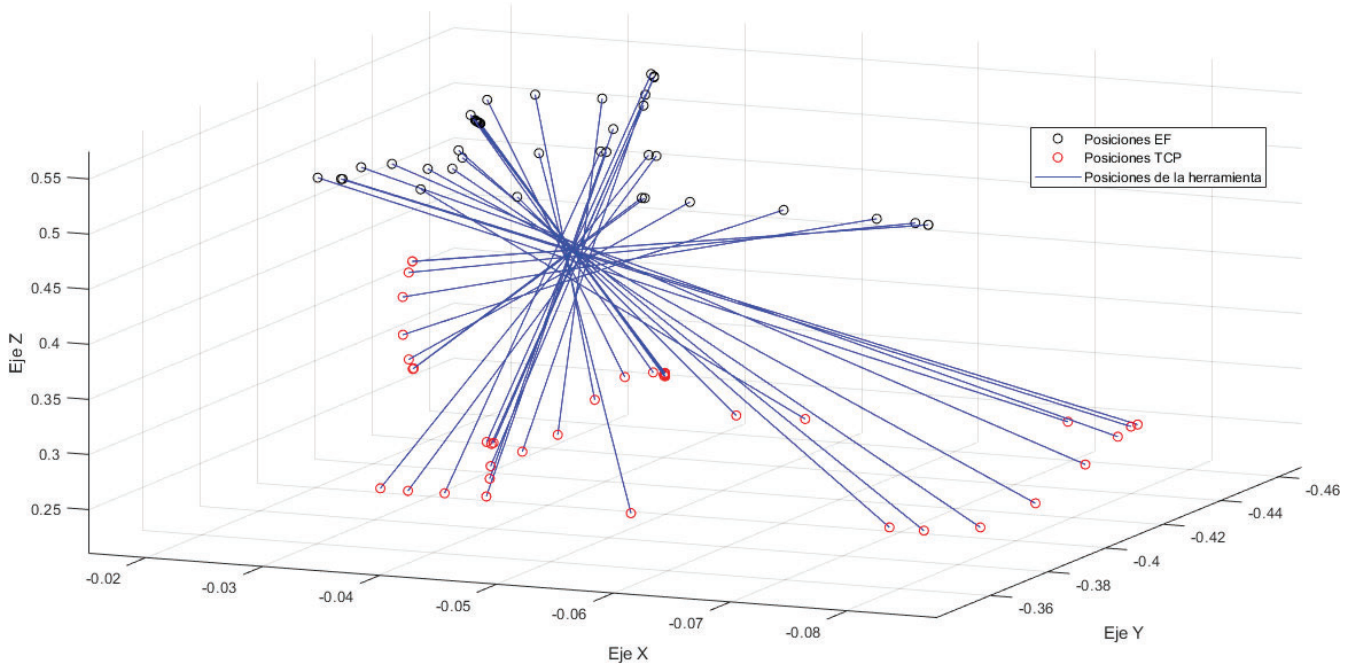
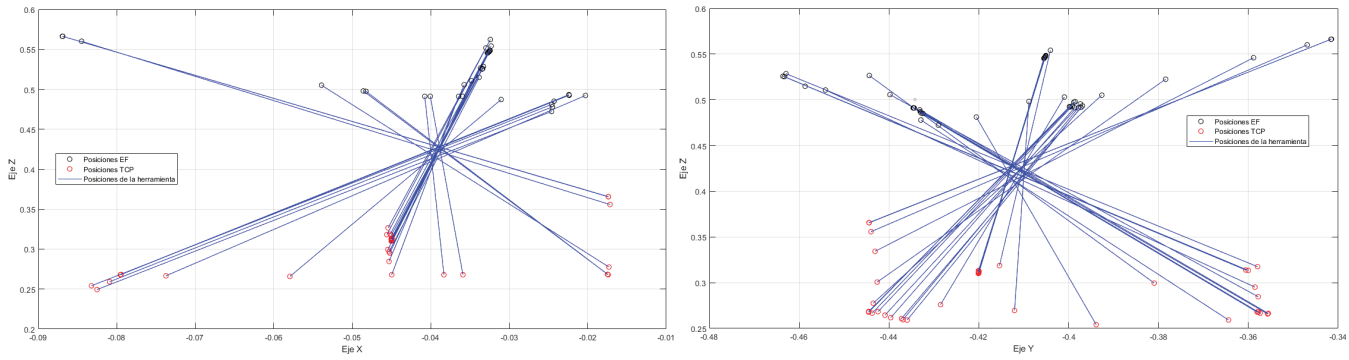


Figura 43: Gráfica 3D con herramienta de 0.236 m e incrementos de 0.1 m en modo real.



(a) Plano XZ (b) Plano YZ

Figura 44: Gráficas 2D con herramienta de 0.236 m e incrementos de 0.1 m en modo real.

A continuación, se aborda la evaluación de los errores cometidos con respecto al punto de fulcro de los movimientos anteriores, donde se ha generado para ello una gráfica que visualiza los errores para cada una de las 10 posiciones comandadas y la transición entre ellas. Esta representación, ilustrada en la Figura 45, permite identificar de manera clara las discrepancias entre la posición real y la posición deseada, destacando cualquier desviación con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido es aproximadamente de 4 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

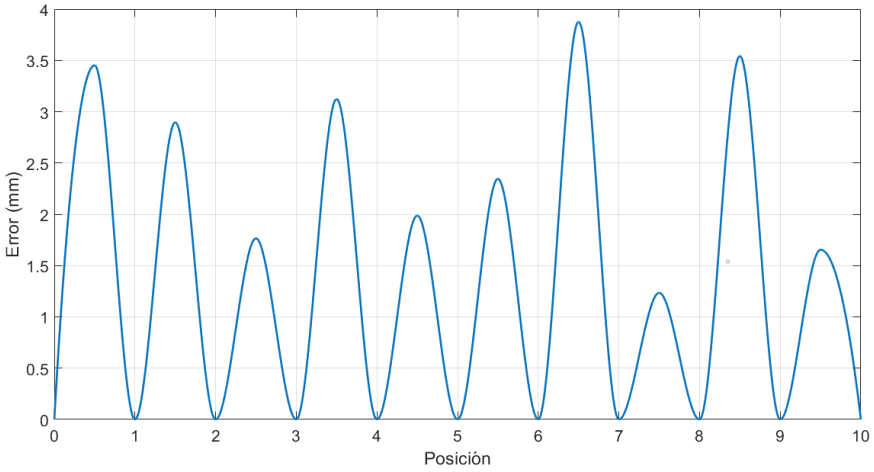


Figura 45: Errores por posición con herramienta de 0.236 m e incrementos de 0.1 m en modo real.

Posteriormente, se ha elaborado una tabla detallada que presenta el error medio cometido en cada desplazamiento entre posiciones consecutivas, proporcionando una visión más cuantitativa de las discrepancias. Esta tabla ofrece una perspectiva detallada del rendimiento del robot en términos de precisión, permitiendo una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa en 2.422839 mm.

Posición	**Error medio (mm)**
$0 \rightarrow 1$	2.157618
$1 \rightarrow 2$	1.811021
$2 \rightarrow 3$	1.103395
$3 \rightarrow 4$	1.952164
$4 \rightarrow 5$	1.242284
$5 \rightarrow 6$	1.466049
$6 \rightarrow 7$	2.422839
$7 \rightarrow 8$	0.771604
$8 \rightarrow 9$	2.214506
$9 \rightarrow 10$	1.033951

Cuadro 5: Errores medios con herramienta de 0.236 m e incrementos de 0.1 m en modo real.

Gráficas para la herramienta de 0.236 m con incrementos de 0.005 m:

Habiendo realizado la prueba para incrementos de 0.1 m, ahora se repite para incrementos de posición de 0.005 m.

Se generan representaciones gráficas que capturan los desplazamientos del robot, considerando incrementos de posición ahora establecidos en 0.005 m, manteniendo una longitud de herramienta de 0.236 m. Con el fin de propiciar una observación minuciosa de estos movimientos, se configura una representación tridimensional (Figura 46), complementada con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 47). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

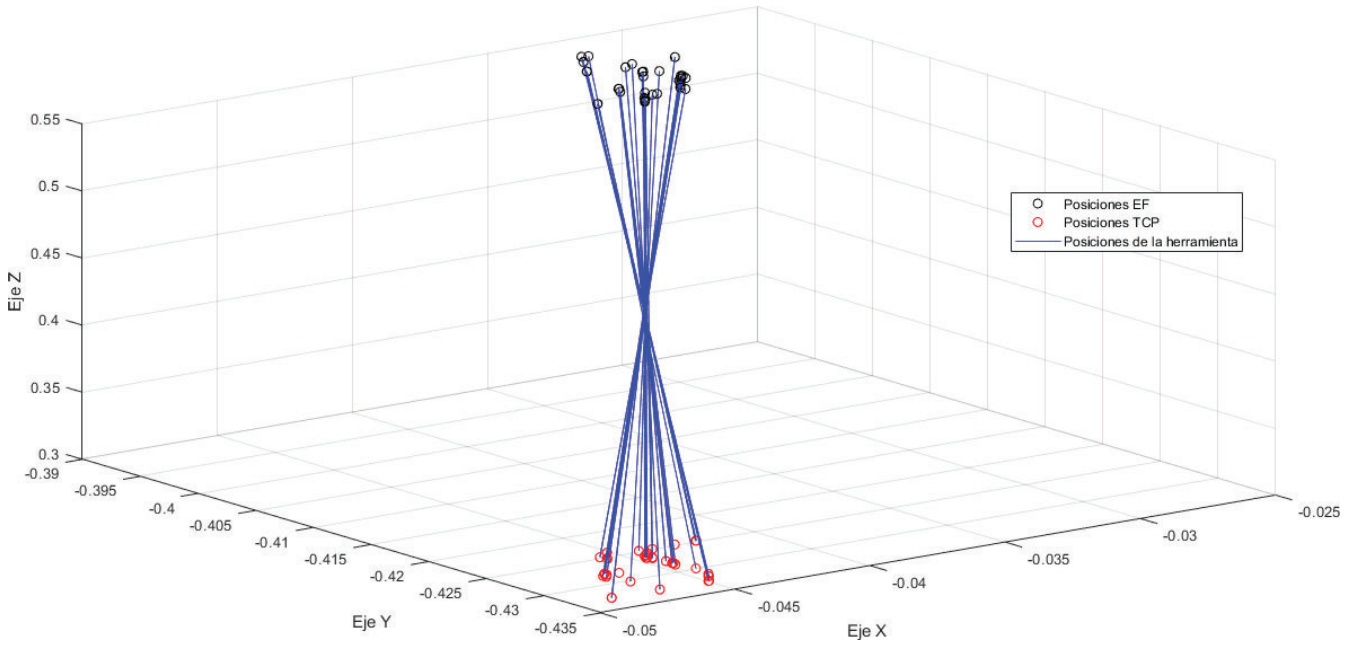
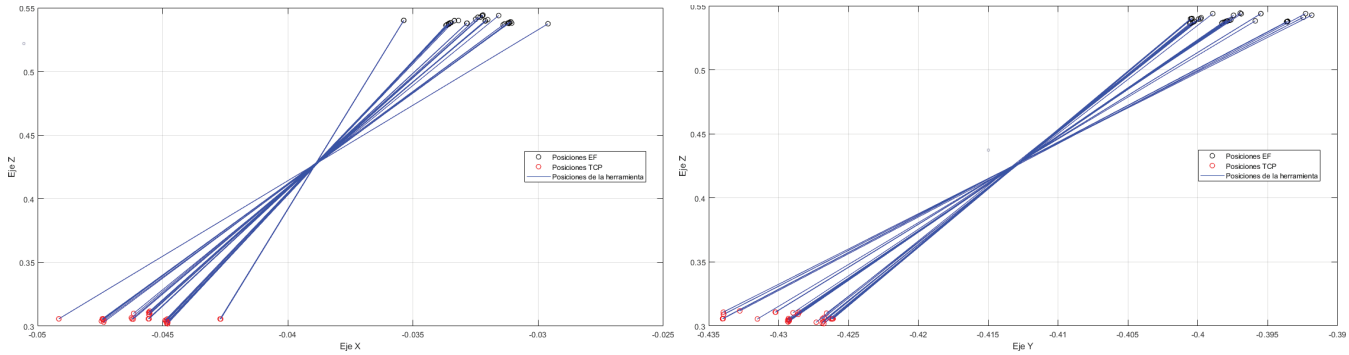


Figura 46: Gráfica 3D con herramienta de 0.236 m e incrementos de 0.005 m en modo real.



(a) Plano XZ

(b) Plano YZ

Figura 47: Gráficas 2D con herramienta de 0.236 m e incrementos de 0.005 m en modo real.

Como anteriormente, se procede a la evaluación de los errores asociados al punto de fulcro en los movimientos realizados. Para ello, se ha creado una gráfica específica que presenta los errores en cada una de las 10 posiciones comandadas, así como en las transiciones entre ellas. La representación visual de estos errores, visualizable en la Figura 48, ofrece una clara identificación de las discrepancias entre la posición efectiva y la posición deseada, resaltando las desviaciones con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido es aproximadamente de 0.35 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

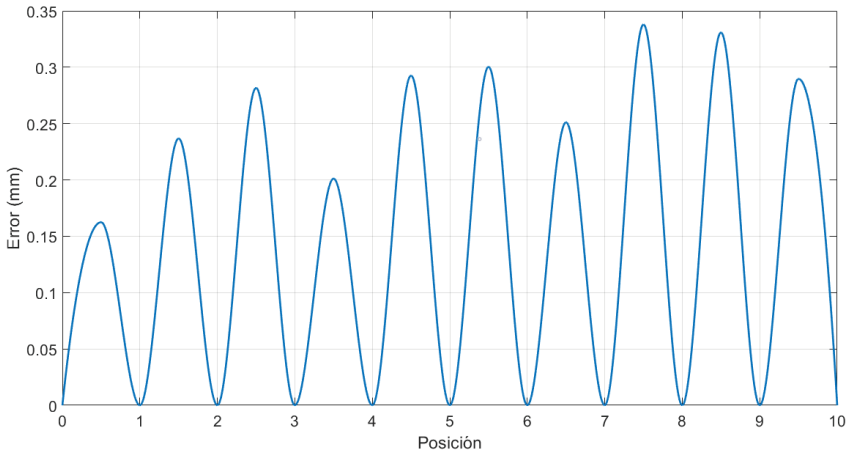


Figura 48: Errores por posición con herramienta de 0.236 m e incrementos de 0.005 m en modo real.

En una etapa posterior, se ha generado una tabla exhaustiva que resume el error medio en cada desplazamiento entre posiciones consecutivas, brindando una perspectiva cuantitativa más detallada de las discrepancias y proporcionando una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa ahora en 0.211217 mm.

Posición	**Error medio (mm)**
$0 \rightarrow 1$	0.101558
$1 \rightarrow 2$	0.148015
$2 \rightarrow 3$	0.176036
$3 \rightarrow 4$	0.125743
$4 \rightarrow 5$	0.182899
$5 \rightarrow 6$	0.187761
$6 \rightarrow 7$	0.156973
$7 \rightarrow 8$	0.211217
$8 \rightarrow 9$	0.206827
$9 \rightarrow 10$	0.181057

Cuadro 6: Errores medios con herramienta de 0.236 m e incrementos de 0.005 m en modo real.

Gráficas para la herramienta de 0.236 m con incrementos de 0.001 m:

Habiendo realizado la prueba para incrementos de 0.1 m y 0.005 m, se repite una última vez la prueba para la herramienta de 0.236 m en este caso con incrementos de posición de 0.001 m.

Se generan las gráficas que registran los desplazamientos del robot, considerando incrementos de posición ahora establecidos en 0.001 m, manteniendo una longitud de herramienta de 0.236 m. Con el fin de propiciar una observación minuciosa de estos movimientos, se configura una representación tridimensional (Figura 49), complementada con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 50). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

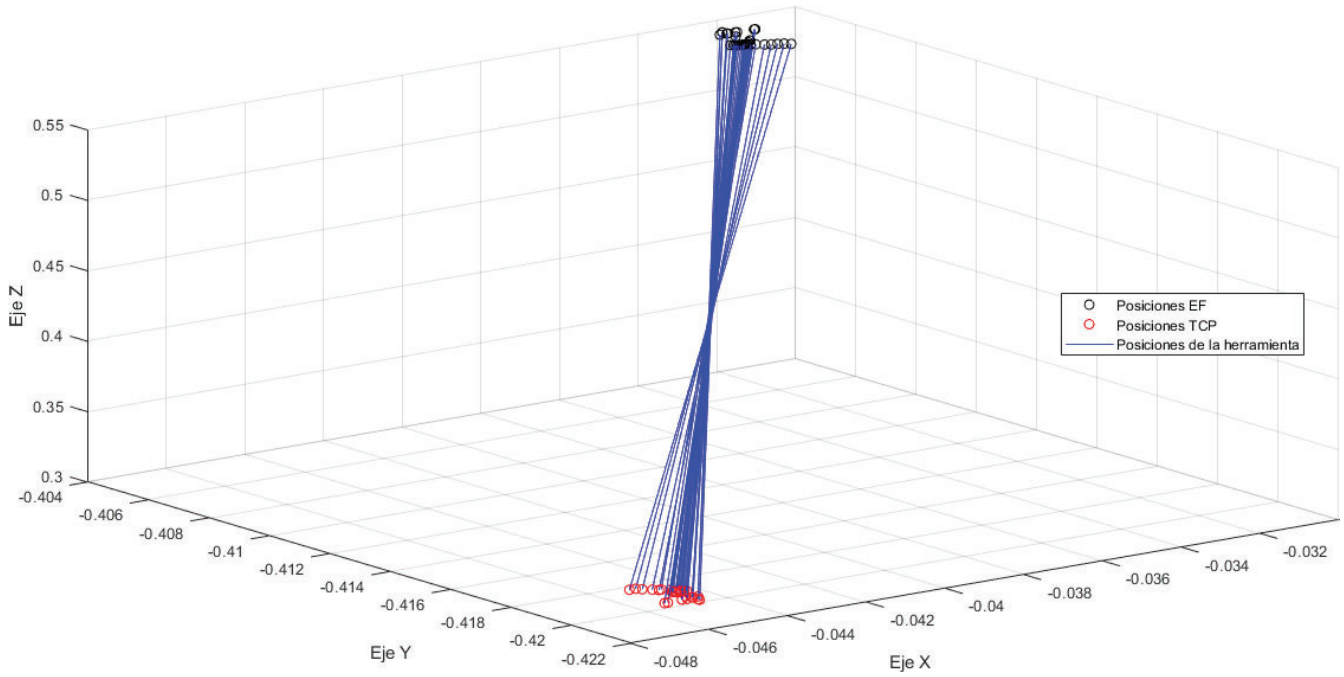
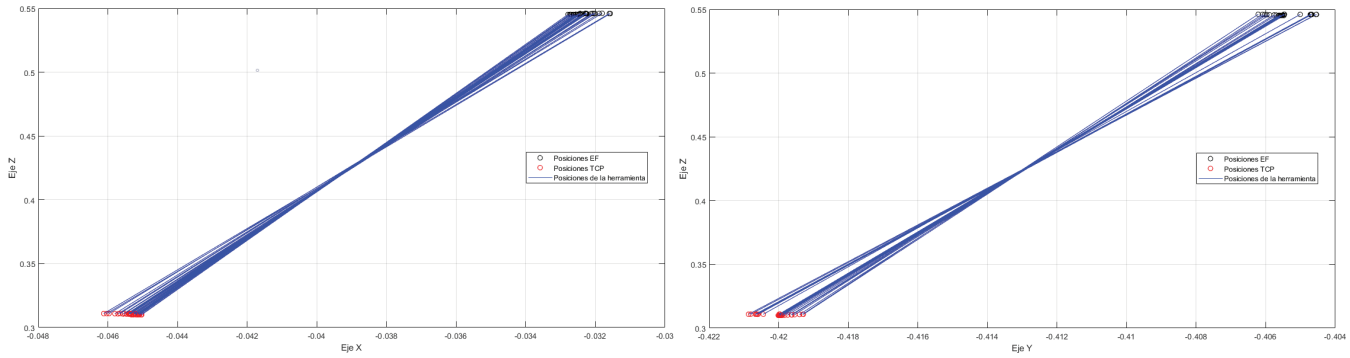


Figura 49: Gráfica 3D con herramienta de 0.236 m e incrementos de 0.001 m en modo real.



(a) Plano XZ (b) Plano YZ

Figura 50: Gráficas 2D con herramienta de 0.236 m e incrementos de 0.001 m en modo real.

Como anteriormente, se procede a la evaluación de los errores asociados al punto de fulcro en los movimientos realizados. Para ello, se ha creado una gráfica específica que presenta los errores en cada una de las 10 posiciones comandadas, así como en las transiciones entre ellas. La representación visual de estos errores, visualizable en la Figura 51, ofrece una clara identificación de las discrepancias entre la posición efectiva y la posición deseada, resaltando las desviaciones con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido es aproximadamente de 0.21 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

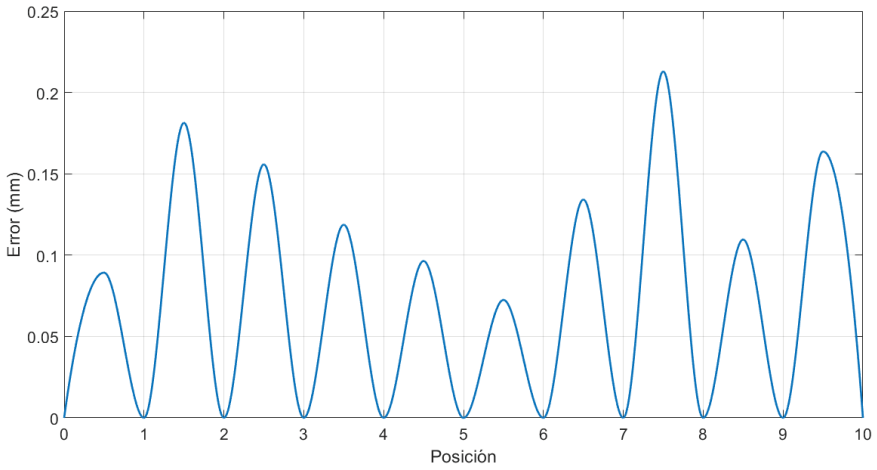


Figura 51: Errores por posición con herramienta de 0.236 m e incrementos de 0.001 m en modo real.

Posteriormente, se ha generado una tabla exhaustiva que resume el error medio en cada desplazamiento entre posiciones consecutivas, brindando una perspectiva cuantitativa más detallada de las discrepancias y proporcionando una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa en 0.134086 mm.

Posición	**Error medio (mm)**
$0 \rightarrow 1$	0.057421
$1 \rightarrow 2$	0.112372
$2 \rightarrow 3$	0.097492
$3 \rightarrow 4$	0.074328
$4 \rightarrow 5$	0.061255
$5 \rightarrow 6$	0.045514
$6 \rightarrow 7$	0.083981
$7 \rightarrow 8$	0.134086
$8 \rightarrow 9$	0.068614
$9 \rightarrow 10$	0.103346

Cuadro 7: Errores medios con herramienta de 0.236 m e incrementos de 0.001 m en modo real.

Gráficas para la herramienta de 0.186 m con incrementos de 0.001 m:

Por último, se modifica la longitud de la herramienta a 0.186 m y se repite la prueba una última vez para incrementos de posición de 0.001 m.

Se generan las gráficas que registran los desplazamientos del robot, considerando incrementos de posición ahora establecidos en 0.001 m, con una longitud de herramienta de 0.186 m. Con el fin de propiciar una observación minuciosa de estos movimientos, se configura una representación tridimensional (Figura 52), complementada con dos representaciones bidimensionales correspondientes a los planos XZ e YZ (Figura 53). En dichas representaciones, se han trazado líneas que conectan el EF con el TCP durante la ejecución de los movimientos del robot. Como se puede observar, estas líneas convergen de manera evidente en un único punto, que corresponde al punto de fulcro.

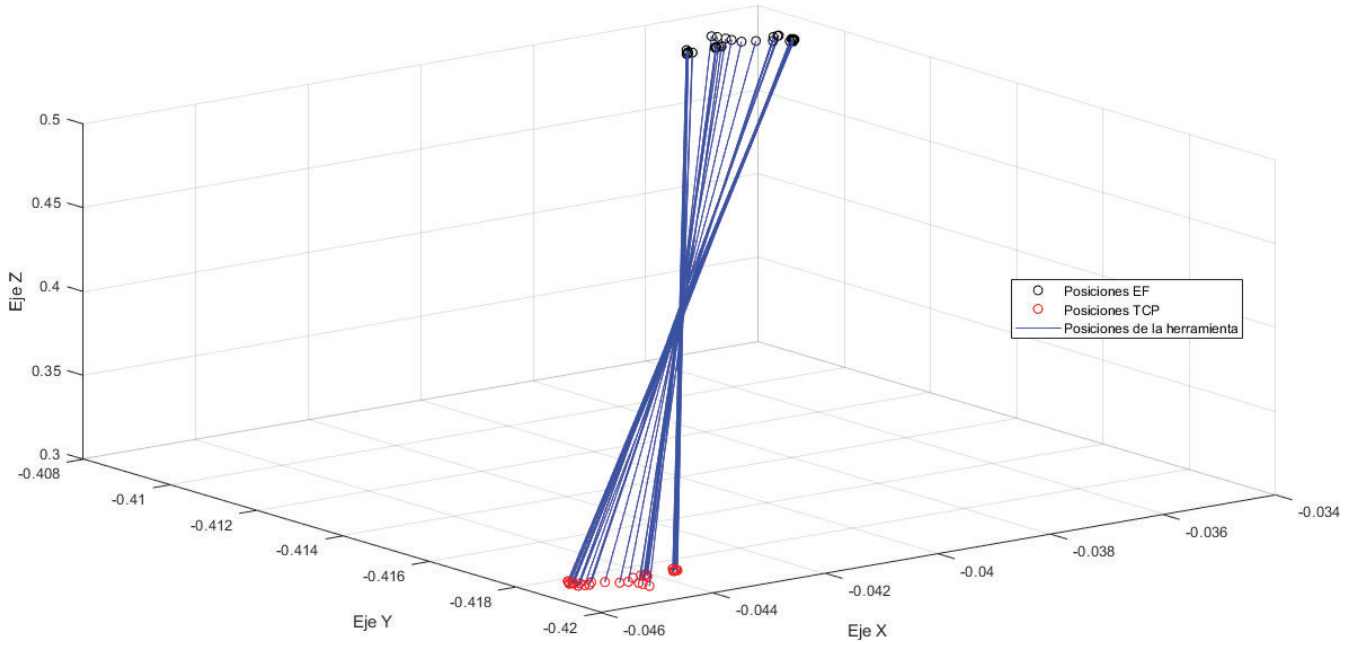
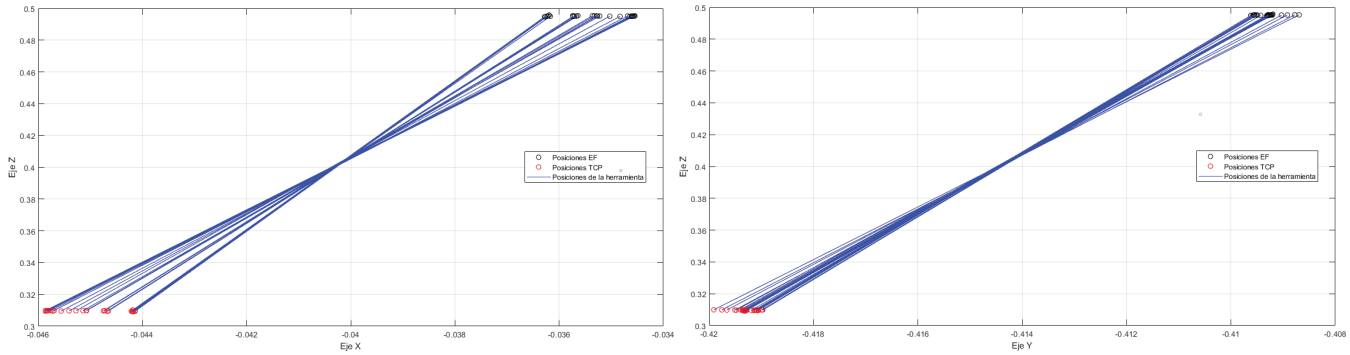


Figura 52: Gráfica 3D con herramienta de 0.186 m e incrementos de 0.001 m en modo real.



(a) Plano XZ (b) Plano YZ

Figura 53: Gráficas 2D con herramienta de 0.186 m e incrementos de 0.001 m en modo real.

Como ya se hizo para el anterior caso, se procede a la evaluación de los errores asociados al punto de fulcro en los movimientos realizados. Para ello, se ha creado una gráfica específica que presenta los errores en cada una de las 10 posiciones comandadas, así como en las transiciones entre ellas. La representación visual de estos errores, visualizable en la Figura 54, ofrece una clara identificación de las discrepancias entre la posición efectiva y la posición deseada, resaltando las desviaciones con respecto al punto de fulcro. La gráfica de errores revela que el máximo error cometido está en torno a 0.22 mm. No obstante, se evidencia de manera concluyente que, para cada una de las posiciones comandadas, el error se mantiene en 0 mm.

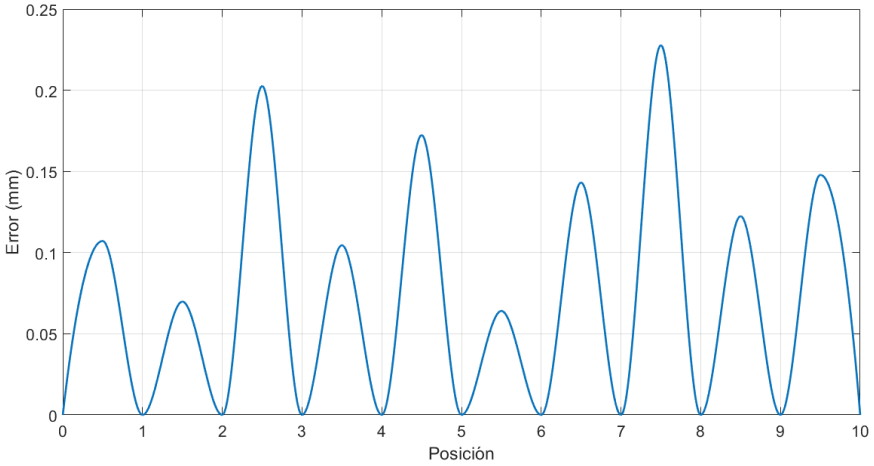


Figura 54: Errores por posición con herramienta de 0.186 m e incrementos de 0.001 m en modo real.

Posteriormente, se ha generado una tabla exhaustiva que resume el error medio en cada desplazamiento entre posiciones consecutivas, brindando una perspectiva cuantitativa más detallada de las discrepancias y proporcionando una evaluación precisa de los errores a lo largo de la trayectoria. En dicha tabla se revela que el error medio más alto cometido se sitúa en 0.142313 mm.

Posición	**Error medio (mm)**
$0 \rightarrow 1$	0.057321
$1 \rightarrow 2$	0.047875
$2 \rightarrow 3$	0.126626
$3 \rightarrow 4$	0.064118
$4 \rightarrow 5$	0.127762
$5 \rightarrow 6$	0.048687
$6 \rightarrow 7$	0.066533
$7 \rightarrow 8$	0.142313
$8 \rightarrow 9$	0.076543
$9 \rightarrow 10$	0.087123

Cuadro 8: Errores medios con herramienta de 0.186 m e incrementos de 0.001 m en modo real.

Resultados

1. La interfaz es capaz de generar movimientos del robot real alrededor de un punto de fulcro independientemente de la longitud de la herramienta y de los incrementos de posición. Estos movimientos se pueden visualizar en el siguiente video: Movimiento de pivote.

2. Se observa que a medida que se aumentan los incrementos de posición en las pruebas, el área (error) alrededor del punto de fulcro aumenta. Esto significa que el robot se desvía ligeramente de la trayectoria deseada cuando los incrementos de posición son más grandes. Esta desviación es una consecuencia de no generar trayectorias intermedias entre los puntos, lo que resulta en una trayectoria más recta y menos curva alrededor del punto de fulcro.

Análisis de resultados

En conclusión, la prueba de movimiento cartesiano alrededor de un punto de fulcro ha sido exitosa, demostrando así la eficacia de la interfaz multimodal para controlar el movimiento del robot real alrededor de un punto de fulcro, sea cual sea la longitud de la herramienta.

Por otro lado, como ya sucedía para el robot simulado, la variación en la precisión del movimiento con respecto al tamaño de los incrementos de posición resulta un aspecto a tener en cuenta. Esto destaca la importancia de generar trayectorias más completas entre los puntos en lugar de simplemente enviar puntos sueltos. En concreto, estas trayectorias deben descomponerse en un conjunto de puntos con una distancia máxima de 0.001 m, lo que asegura un error lo suficientemente pequeño. Esta meticulosa descomposición garantiza que el robot respete el punto de fulcro durante todo su recorrido, asegurando así la minuciosidad y estabilidad necesarias en los movimientos.

6. Conclusiones y futuros trabajos

En este Trabajo Fin de Máster se ha desarrollado una interfaz multimodal para el control de un robot KUKA LBR iiwa con fines quirúrgicos a través del lenguaje Python y sobre el sistema operativo de ROS. A lo largo de este trabajo, se han logrado avances sustanciales y se han obtenido valiosas conclusiones que merecen ser destacadas:

- Se ha concebido y desarrollado con éxito una interfaz multimodal que permite un control preciso y versátil del robot KUKA LBR iiwa 7 R800 como robot quirúrgico. Esto incluye la capacidad de realizar movimientos articulares y movimientos cartesianos libres para lograr una posición inicial concreta y adaptable del robot antes de la cirugía, la capacidad de realizar movimientos de pivote alrededor de un punto de fulcro para llevar a cabo los diversos procedimientos quirúrgicos y la recepción del estado del robot y de la posición deseada del TCP para el control y la planificación de los movimientos. Además de todo ello, también dispone de la capacidad de configuración flexible de todos los parámetros relevantes del robot, así como de un modo de funcionamiento simulado del sistema, que permite a los usuarios practicar y simular procedimientos quirúrgicos antes de llevarlos a cabo en el quirófano.
- La integración de esta interfaz multimodal en el entorno de ROS ha sido exitosa, permitiendo cambiar entre diferentes modos de funcionamiento del robot y ajustar los parámetros de configuración de manera eficiente y sencilla. La validación de su funcionamiento se ha llevado a cabo mediante una serie de experimentos tanto en el modo simulado como real, confirmando así la efectividad de todas sus funcionalidades.
- El robot KUKA LBR iiwa 7 R800 se caracteriza por ser una plataforma ideal para la colaboración entre humanos y robots, siendolo ahora además en el contexto quirúrgico. Sus capacidades sensoriales y su diseño modular lo hacen más que adecuado para una amplia gama de aplicaciones médicas y tecnológicas. La implementación del enfoque multimodal tratado en este trabajo, no ha hecho más que mejorar aún más su versatilidad y eficacia en entornos quirúrgicos.
- Todos los aspectos del código desarrollado en este proyecto se han documentado exhaustivamente, proporcionando una base sólida y comprensible para futuros trabajos en este campo. Esta documentación detallada no solo respalda la replicabilidad y la mejora continua del presente trabajo, sino que también sirve como referencia valiosa para investigadores y desarrolladores que buscan ampliar y perfeccionar esta interfaz multimodal en el futuro.

Aunque gracias a este proyecto se han logrado avances notables, existen varias oportunidades para futuras investigaciones y desarrollos en el campo de la robótica quirúrgica:

- **Generación dinámica de trayectorias:** Una vía potencial para la mejora de la interfaz multimodal es la implementación de una generación dinámica de trayectorias. Esto implica desarrollar un control de trayectoria que ajuste la ruta del robot en función de la distancia entre el punto comandado y su posición actual. Cuando la distancia excede un umbral definido, se generarían puntos intermedios para garantizar movimientos más suaves y estables alrededor del punto de fulcro. Esta funcionalidad no solo optimizaría el rendimiento del robot en términos de precisión, sino que también podría mejorar la seguridad durante las cirugías al minimizar las oscilaciones indeseadas.
- **Detección de errores durante el movimiento:** Integrar capacidades avanzadas de detección de errores podría ser crucial para la seguridad en entornos quirúrgicos. El control de colisiones, especialmente al operar con más de un brazo robótico simultáneamente, es una área clave de mejora. La interfaz multimodal podría ser mejorada para detectar posibles colisiones y realizar ajustes automáticos en tiempo real para evitar situaciones peligrosas. Asimismo, aprovechar la séptima articulación del robot para detectar y reaccionar ante colisiones externas podría ser otra función valiosa, añadiendo así un nivel extra de seguridad.
- **Uso de herramientas no alineadas con el eje Z del efector final:** Explorar la integración de herramientas no alineadas con el eje Z del efector final abriría nuevas posibilidades en cirugías especializadas. Al adaptar la interfaz multimodal para trabajar de manera eficiente con herramientas que no siguen la orientación estándar, se podrían abordar procedimientos quirúrgicos específicos que requieran un enfoque más flexible. Esto implicaría el desarrollo de algoritmos de control y lógicas de operación específicas para estas herramientas, mejorando así la versatilidad y aplicabilidad de la interfaz en una variedad de contextos clínicos.

Referencias

[1] Surgical Robotics Industry and Technology Overview. https://bisresearch.com/industry-report/surgical-robotics-market-report.html

[2] Rivas-Blanco, I., Pérez-Del-Pulgar, C. J., García-Morales, I., & Muñoz, V. F. (2021). A Review on Deep Learning in Minimally Invasive Surgery. IEEE Access, 9, 48658-48678. Published: March 24, 2021. Electronic ISSN: 2169-3536. INSPEC Accession Number: 20965690. DOI: 10.1109/ACCESS.2021.3068852. Publisher: IEEE.

[3] Vilchis - González, A. H., Ávila - Vilchis, J. C., Estrada - Flores, R. G., Martínez - Méndez, R., Portillo - Rodríguez, O., & Romero - Huertas, M. (2014). Modular Robots for Minimally Invasive Surgery. Revista Mexicana De Ingenieria Biomedica, 35(1), 63–79. Retrieved from https://mail.rmib.mx/index.php/rmib/article/view/215

[4] Romanelli, J.R., L. Mark, and P.A. Omotosho. 2008. "Single Port Laparoscopic Cholecystectomy with the TriPort System: A Case Report." Surgical innovation 15(3): 223–28.

[5] Gascón Hove, M. et al. 2014. "Single-Port Access Surgery Resection of a Presacral Schwannoma | SpringerLink." Techniques in Coloproctology 18(4): 407–8.

[6] Wang, C.J. et al. 2016. "Natural Orifice Transluminal Endoscopic Surgery-Assisted versus Laparoscopic Ovarian Cystectomy (NAOC vs. LOC): A Case-Matched Study." Surgical Endoscopy 30(3): 1227–34.

[7] Lanfranco AR, Castellanos AE, Desai JP, Meyers WC. Robotic surgery: a current perspective. Ann Surg. 2004 Jan;239(1):14-21. doi: 10.1097/01.sla.0000103020.19595.7d. PMID: 14685095; PMCID: PMC1356187.

[8] Ranev D, Teixeira J. History of Computer-Assisted Surgery. Surg Clin North Am. 2020 Apr;100(2):209-218. doi: 10.1016/j.suc.2019.11.001. Epub 2020 Jan 7. PMID: 32169176.

[9] Sommer GM, Broschewitz J, Huppert S, Sommer CG, Jahn N, Jansen-Winkeln B, Gockel I, Hau HM. The role of virtual reality simulation in surgical training in the light of COVID-19 pandemic: Visual spatial ability as a predictor for improved surgical performance: a randomized trial. Medicine (Baltimore). 2021 Dec 17;100(50):e27844. doi: 10.1097/MD.0000000000027844. PMID: 34918632; PMCID: PMC8677906.

[10] Finley, D. S., & Nguyen, N. T. (2005). Surgical robotics. Current Surgery, 62(2), 262-272. ISSN 0149-7944. doi: https://doi.org/10.1016/j.cursur.2004.11.005.

[11] CORDOVA DUPEYRAT, Alfredo y H. BALLANTYNE, Garth. Sistemas Quirúrgicos Robóticos y Telerobóticos para cirugía abdominal. Rev. gastroenterol. Perú [online]. 2003, vol.23, n.1 [citado 2023-10-26], pp.58-66. Disponible en: http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S1022-51292003000100008&lng=es&nrm=iso. ISSN 1022-5129.

[12] Morrell ALG, Morrell-Junior AC, Morrell AG, Mendes JMF, Tustumi F, DE-Oliveira-E-Silva LG, Morrell A. The history of robotic surgery and its evolution: when illusion becomes reality. Rev Col Bras Cir. 2021 Jan 13;48:e20202798. English, Portuguese. doi: 10.1590/0100-6991e-20202798. PMID: 33470371.

[13] P Ballester, Y Jain, K.R Haylett, R.F McCloy, Comparison of task performance of robotic camera holders EndoAssist and Aesop, International Congress Series, Volume 1230, 2001, Pages 1100-1103, ISSN 0531-5131, https://doi.org/10.1016/S0531-5131(01)00231-X.

[14] Gilbert JM. The EndoAssist robotic camera holder as an aid to the introduction of laparoscopic colorectal surgery. Ann R Coll Surg Engl. 2009 Jul;91(5):389-93. doi: 10.1308/003588409X392162. Epub 2009 Apr 30. PMID: 19409150; PMCID: PMC2758433.

[15] Kommu, S.S., Rimington, P., Anderson, C. et al. Initial experience with the EndoAssist camera-holding robot in laparoscopic urological surgery. J Robotic Surg 1, 133–137 (2007). https://doi.org/10.1007/s11701-007-0010-5

[16] Lanfranco AR, Castellanos AE, Desai JP, Meyers WC. Robotic surgery: a current perspective. Ann Surg. 2004 Jan;239(1):14-21. doi: 10.1097/01.sla.0000103020.19595.7d. PMID: 14685095; PMCID: PMC1356187.

[17] ZEUS y Da Vinci: Presente y futuro de la cirugía. https://robotics2017site.wordpress.com/tag/cirugia-robotica/

[18] Soriano-Baron, H., Martinez-del-Campo, E., Crawford, N. R., & Theodore, N. Robotics in Spinal Surgery: The Future is Here.

[19] George EI, Brand TC, LaPorta A, Marescaux J, Satava RM. Origins of Robotic Surgery: From Skepticism to Standard of Care. JSLS. 2018 Oct-Dec;22(4):e2018.00039. doi: 10.4293/JSLS.2018.00039. PMID: 30524184; PMCID: PMC6261744.

[20] R. E. Perez and S. D. Schwaitzberg, "Robotic surgery: finding value in 2019 and beyond," Ann. Laparosc. Endosc. Surg., vol. 4, no. 0, pp. 51–51, May 2019. https://ales.amegroups.org/article/view/5205/html

[21] De Backer, I. (2020, April 24). Robots Provide Surgeons With Superhuman Stability and Precision. https://www.idtechex.com/en/research-article/robots-provide-surgeons-with-superhuman-stability-and-precision/20532

[22] Muñoz VF. Sensors Technology for Medical Robotics. Sensors (Basel). 2022 Nov 29;22(23):9290. doi: 10.3390/s22239290. PMID: 36501991; PMCID: PMC9736968.

[23] Wang, J., Zhang, X., Chen, X., & Song, Z. (2023). A touch-free human-robot collaborative surgical navigation robotic system based on hand gesture recognition. Frontiers in Neuroscience, 17. https://www.frontiersin.org/articles/10.3389/fnins.2023.1200576. DOI: 10.3389/fnins.2023.1200576

[24] KUKA. LBR iiwa 7 R800. https://www.kuka.com/es-es/productos-servicios/sistemas-de-robot/robot-industrial/lbr-iiwa

[25] ¿Qué es un COBOT o Robot Colaborativo? https://innovacion-tecnologia.com/robotica/que-es-un-cobot-aplicaciones-tipos-precios/#google_vignette

[26] Programming and Configuration of LBR iiwa: KUKA Sunrise.Operating System 1.x. Workshop LBR iiwa V3. Ausburg: KUKA Roboter, 2015.

[27] IIWA STACK WIKI. https://github.com/IFL-CAMP/iiwa_stack/wiki

[28] Hello World Project in ROS https://automaticaddison.com/create-a-hello-world-project-in-ros/

[29] Pytransform3d https://dfki-ric.github.io/pytransform3d/

[30] DOXYGEN https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/110

[31] DOXYGEN Documentation https://www.doxygen.nl/index.html

ANEXOS

A. Hoja de especificaciones del KUKA LBR iiwa 7 R800

KUKA

LBR iiwa 7 R800

Datos básicos

Tipo de cinemática	Brazo articulado
Cobot	Sí
Requisitos de seguridad	Categoría 3 y Performance Level d según EN ISO 13849-1

Datos técnicos

Alcance máximo	800 mm
Carga nominal	7 kg
Carga máxima	-
Repetibilidad de posición (ISO 9283)	± 0,1 mm
Número ejes	7
Posición de montaje	Suelo; Techo; Pared
Superficie de colocación	-
Peso	aprox. 23,9 kg

Datos de los ejes

Rango de desplazamiento	
A1	±170 °
A2	±120 °
A3	±170 °
A4	±120 °
A5	±170 °
A6	±120 °
Rango de desplazamiento eje 7	±175 °
Velocidad con carga nominal	
A1	98 °/s
A2	98 °/s
A3	100 °/s
A4	130 °/s
A5	140 °/s
A6	180 °/s
Velocidad del eje 7	180 °/s

Condiciones de servicio

Temperatura ambiente durante el servicio	5 °C hasta 45 °C (278 K hasta 318 K)

Tipo de protección

Tipo de protección (IEC 60529)	IP54

Unidad de control

Unidad de control	KUKA Sunrise Cabinet

Gráfica del campo de trabajo

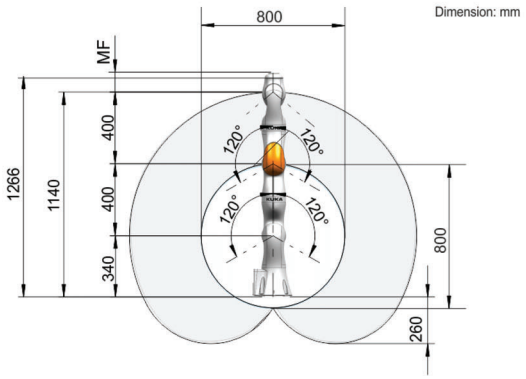
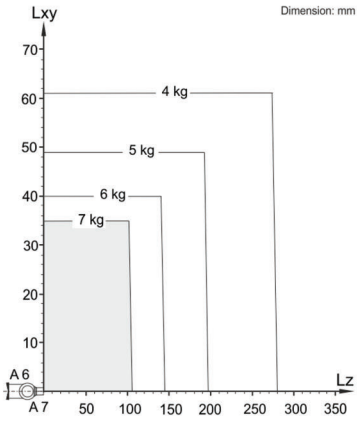


Diagrama de cargas



B. Archivo iiwa_control_node.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import numpy as np
from geometry_msgs.msg import PoseStamped
from iiwa_msgs.msg import CartesianPose
from iiwa_msgs.msg import JointPosition
from iiwa_control_class import iiwa_surgery_class
from sensor_msgs.msg import JointState
from conversions import matrix_from_quaternion

class iiwa_surgery_node_class:

    def __init__(self):

        rospy.init_node('iiwa_surgery_node')
        self.iiwa = iiwa_surgery_class()
        self.first_position = None

        # Obtener los valores de los parametros del sistema de parametros de ROS
        self.simulation_mode = rospy.get_param("simulation_mode", True)
        self.tool_length = rospy.get_param("tool_length", 0.0)
        self.tool_orientation = rospy.get_param("tool_orientation", [0.0, 0.0, 0.0])
        self.fulcrum_fi = rospy.get_param("fulcrum_fi", 0.0)
        self.robot_ip = rospy.get_param("robot_ip", "")
        self.work_mode = rospy.get_param("work_mode", "free")

        # Pasar los valores de los parametros a la instancia de la clase iiwa_surgery_class
        self.iiwa.simulation_mode = self.simulation_mode
        self.iiwa.tool_length = self.tool_length
        self.iiwa.set_tool_data(self.tool_length, self.tool_orientation)
        self.iiwa.set_fulcrum_fi(self.fulcrum_fi)
        self.iiwa.set_robot_ip(self.robot_ip)
        self.iiwa.set_work_mode(self.work_mode)

        # Configurar suscriptores y publicadores de ROS
        self.joint_sub = rospy.Subscriber('iiwa_surgery/command/joints', JointPosition, self.
    joint_command_callback)
        self.pose_sub = rospy.Subscriber('iiwa_surgery/command/pose', PoseStamped, self.
    pose_command_callback)

        self.joint_pub = rospy.Publisher('iiwa_surgery/output/joints', JointPosition, queue_size
    =10)
        self.ef_pose_pub = rospy.Publisher('iiwa_surgery/output/ef_pose', PoseStamped, queue_size
    =10)
        self.tcp_pose_pub = rospy.Publisher('iiwa_surgery/output/tcp_pose', PoseStamped,
    queue_size=10)

        if self.simulation_mode:
            self.joint_state_sub = rospy.Subscriber('/iiwa/joint_states', JointState, self.
    joint_state_callback)
            self.cartesian_pose_sub = rospy.Subscriber('/iiwa/state/CartesianPose', PoseStamped,
    self.cartesian_pose_callback)
        else:
            self.joint_position_sub = rospy.Subscriber('/iiwa/state/JointPosition', JointPosition,
     self.joint_position_callback)
            self.cartesian_pose_sub = rospy.Subscriber('/iiwa/state/CartesianPose', CartesianPose,
     self.cartesian_pose_callback)



    def joint_command_callback(self, msg):

        # Mover el robot a una posicion articular especifica
        joint_config = JointPosition()
        joint_config = msg
        # Enviar mensaje de posicion articular al metodo
```

```
60          self.iiwa.move_joint(joint_config)
61
62      def pose_command_callback(self, msg):
63
64          if self.work_mode == "free":
65              # Mover el robot a una posicion cartesiana especifica en modo "free"
66              pose = PoseStamped()
67              pose = msg
68              self.iiwa.move_cartesian(pose)
69          elif self.work_mode == "pivot":
70              # Mover el robot en modo "pivot" alrededor de un punto de fulcro
71              pose = PoseStamped()
72              pose = msg
73
74              # Obtener la posicion x, y, z del mensaje de entrada
75              position = np.array([msg.pose.position.x, msg.pose.position.y, msg.pose.position.z])
76
77              if self.first_position is None:
78                  # Primer mensaje recibido, enviar directamente la posicion al metodo
79                  self.first_position = position
80                  increment_vector = np.zeros(3)  # Crear un vector de ceros
81                  j = 0
82                  self.iiwa.move_cartesian_fulcrum(pose, increment_vector, j)
83              else:
84                  # Calcular vector de incrementos
85                  increment_vector = position - self.first_position
86                  j = 1
87                  # Enviar vector de incrementos y pose al metodo
88                  self.iiwa.move_cartesian_fulcrum(pose, increment_vector, j)
89          else:
90              rospy.logwarn("Modo de trabajo no v lido.")
91
92      def joint_state_callback(self, msg):
93
94          joint_position_msg = JointPosition()
95          joint_position_msg.header = msg.header
96
97          # Asignar los valores de position a los campos a1 a a7
98          joint_position_msg.position.a1 = msg.position[0]
99          joint_position_msg.position.a2 = msg.position[1]
100         joint_position_msg.position.a3 = msg.position[2]
101         joint_position_msg.position.a4 = msg.position[3]
102         joint_position_msg.position.a5 = msg.position[4]
103         joint_position_msg.position.a6 = msg.position[5]
104         joint_position_msg.position.a7 = msg.position[6]
105
106         # Publicar el estado de posicion articular
107         self.joint_pub.publish(joint_position_msg)
108
109     def joint_position_callback(self, msg):
110
111         # Publicar el estado de posicion articular recibido
112         self.joint_pub.publish(msg)
113
114     def cartesian_pose_callback(self, msg):
115
116         cartesian_pose = PoseStamped()
117
118         if self.simulation_mode:
119             # Cuando se trabaja con el simulador, usar el mensaje directamente
120             cartesian_pose = msg
121         else:
122             # Cuando se trabaja con el robot real, mapear a geometry_msgs.PoseStamped
123             cartesian_pose.header = msg.poseStamped.header
124             cartesian_pose.pose.position = msg.poseStamped.pose.position
125             cartesian_pose.pose.orientation = msg.poseStamped.pose.orientation
126
127
128         # Publicar la informaci n recibida
129         self.ef_pose_pub.publish(cartesian_pose)
130
```

```
131         # Calcular la posicion del TCP a partir del EF
132 
133         # Obtener la posicion y orientacion del mensaje
134         position = cartesian_pose.pose.position
135         orientation = cartesian_pose.pose.orientation
136 
137         # Reordenar el cuaternio en el vector q (w, x, y, z)
138         q = [orientation.w, orientation.x, orientation.y, orientation.z]
139 
140         # Crear una matriz de rotacion a partir del cuaternio recibido
141         Rm = matrix_from_quaternion(q)
142 
143         # Calcular el vector de direccion relativo al EF
144         direction = np.array([0,0,self.tool_length])
145         direction_transformed = np.dot(Rm, direction)
146 
147         # Guardar la posicion del EF del robot en la variable Pef
148         Pef = np.array([position.x,position.y,position.z])
149 
150         # Calcular la posicion del TCP
151         Ptcp = Pef + direction_transformed
152 
153         # Actualizar valores de posicion del TCP
154         position.x = Ptcp[0]
155         position.y = Ptcp[1]
156         position.z = Ptcp[2]
157 
158         # Crear un mensaje del tipo PoseStamped
159         tcp_pose = PoseStamped()
160 
161         # Actualizar el mensaje de tcp_pose con los valores ajustados
162         tcp_pose.header = cartesian_pose.header
163         tcp_pose.pose.position = position
164         tcp_pose.pose.orientation = orientation # La herramienta tendra la misma orientacion que
    el EF
165 
166         # Publicar el mensaje
167         self.tcp_pose_pub.publish(tcp_pose)
168 
169 
170     def run(self):
171         rate = rospy.Rate(10)  # Frecuencia de bucle de ejecucion
172 
173         while not rospy.is_shutdown():
174             # Realizar cualquier procesamiento adicional si es necesario
175 
176             rate.sleep()
177 
178 if __name__ == '__main__':
179     node = iiwa_surgery_node_class()
180     node.run()
```

C. Archivo iiwa_control_class.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import numpy as np
import math
import csv
from conversions import matrix_from_quaternion
from conversions import quaternion_from_matrix
from geometry_msgs.msg import PoseStamped
from iiwa_msgs.msg import JointPosition

class iiwa_surgery_class:

    def __init__(self, simulation_mode=True):

        # Variable miembro para configurar el modo de trabajo (simulacion o robot real)
        self.simulation_mode = simulation_mode
        # Variables miembro para los datos de la herramienta
        self.tool_length = 0.0
        self.tool_orientation = [0.0, 0.0, 0.0]
        # Variable miembro para el punto de fulcro
        self.fulcrum_fi = 0.0
        # Variables miembro con el valor de la IP del robot
        self.robot_ip = ""
        # Variables miembro con el modo de trabajo del robot
        self.work_mode = "free"  # Valor predeterminado

        # Otras variables miembro de configuracion del robot, si es necesario

        self.cartesian_pub = rospy.Publisher('/iiwa/command/CartesianPose', PoseStamped, queue_size=10)
        self.joint_pub = rospy.Publisher('/iiwa/command/JointPosition', JointPosition, queue_size=10)
        # ... otros publicadores y suscriptores

    def set_tool_data(self, tool_length, tool_orientation):

        # Metodo para configurar los datos de la herramienta
        # Variables miembro para los datos de la herramienta
        self.tool_length = tool_length
        self.tool_orientation = tool_orientation

    def set_fulcrum_fi(self, fulcrum_fi):

        # Metodo para configurar el punto de fulcro
        if 0 <= fulcrum_fi <= 1:
            # Variable miembro para el punto de fulcro
            self.fulcrum_fi = fulcrum_fi
        else:
            rospy.logwarn("El punto de fulcro debe ser un valor decimal entre 0 y 1.")

    def set_robot_ip(self, robot_ip):

        # Variables miembro con el valor de la IP del robot
        self.robot_ip = robot_ip

    def set_work_mode(self, work_mode):

        if work_mode in ["free", "pivot"]:
            self.work_mode = work_mode
        else:
            rospy.logwarn("Modo de trabajo no v lido.")


    def move_joint(self, joint_config):

        joint_msg = JointPosition()
```

```
67          joint_msg = joint_config
68          self.joint_pub.publish(joint_msg)
69
70      def move_cartesian(self, pose):
71
72          cartesian_msg = PoseStamped()
73
74          # Obtener el EF a partir del TCP:
75
76          # Obtener la posicion y orientacion del mensaje
77          position = pose.pose.position
78          orientation = pose.pose.orientation
79
80          # Reordenar el cuaternio en el vector q (w, x, y, z)
81          q = [orientation.w, orientation.x, orientation.y, orientation.z]
82
83          # Crear una matriz de rotacion a partir del quaternio recibido
84          Rm = matrix_from_quaternion(q)
85
86          # Calcular el vector de direccion relativo al TCP
87          direction = np.array([0,0,-self.tool_length])
88          direction_transformed = np.dot(Rm, direction)
89
90          # Guardar la posicion del TCP
91          Ptcp = np.array([position.x,position.y,position.z])
92
93          # Calcular la posicion del EF
94          Pef = Ptcp + direction_transformed
95
96          # Actualizar los valores de la posicion del EF
97          position.x = Pef[0]
98          position.y = Pef[1]
99          position.z = Pef[2]
100
101         # Actualizar el mensaje de cartesian_msg con los valores ajustados
102         cartesian_msg.header = pose.header
103         cartesian_msg.header.frame_id = "iiwa_link_0"
104         cartesian_msg.pose.position = position
105         cartesian_msg.pose.orientation = orientation #EF tendra la misma orientacion que el TCP
106
107         # Publicar el mensaje
108         self.cartesian_pub.publish(cartesian_msg)
109
110     def move_cartesian_fulcrum(self, pose, increment_vector, j):
111
112         cartesian_msg = PoseStamped() # En este mensaje se almacenara la informaci n del EF
113
114         # nombre_archivo = "/home/javilara/iiwa_stack_ws/src/iiwa_stack/iiwa_surgery/src/
    datos_robot.csv"
115
116         # def guardar_datos_csv(datos, nombre_archivo):
117
118         #     with open(nombre_archivo, mode='a') as archivo_csv:
119         #         escritor_csv = csv.writer(archivo_csv)
120         #         escritor_csv.writerow(datos)
121
122         # Asignar los valores de increment_vector a Ph1, Ph2 y Ph3. Estos ser n los incrementos
    para mover la herramienta
123         Ph1 = increment_vector[0]
124         Ph2 = increment_vector[1]
125         Ph3 = increment_vector[2]
126
127         if(j == 0): #Si es la primera posicion que se recibe:
128
129             # Obtener la posicion y orientacion del mensaje de msg
130             position = pose.pose.position
131             orientation = pose.pose.orientation
132
133             # Reordenar el cuaternio en el vector q (w, x, y, z)
134             q = [orientation.w, orientation.x, orientation.y, orientation.z]
135
```

```
136             # Creamos una matriz de rotacion a partir del quaternio recibido en pose
137             Rm = matrix_from_quaternion(q)
138 
139             # Calculamos el vector de direccion relativo al TCP inicial
140             direction = np.array([0,0,-self.tool_length])
141             direction_transformed = np.dot(Rm, direction)
142 
143             # Guardamos la posicion del TCP inicial
144             self.Ptcp = np.array([position.x,position.y,position.z])
145 
146             # Calcular la posicion del EF inicial
147             Pef = self.Ptcp + direction_transformed
148 
149             # Calcular el vector de direccion relativo al EF
150             direction = np.array([0,0, self.fulcrum_fi*self.tool_length])
151             direction_transformed = np.dot(Rm, direction)
152 
153             # Calcular la posicion del punto de fulcro
154             self.Pf = Pef + direction_transformed # El punto de fulcro se calcula al principio y
    no cambia
155 
156             # Se muestran por pantalla los datos
157             print("Punto de fulcro: ", self.Pf)
158             print("Posicion EF inical: ", Pef)
159             print("Posicion TCP inicial: ", self.Ptcp)
160             print(" ")
161 
162             # Se guardan los datos en el archivo .csv
163             # guardar_datos_csv([self.Pf[0], self.Pf[1], self.Pf[2]], nombre_archivo)
164             # guardar_datos_csv([Pef[0], Pef[1], Pef[2]], nombre_archivo)
165             # guardar_datos_csv([self.Ptcp[0], self.Ptcp[1], self.Ptcp[2]], nombre_archivo)
166 
167 
168             # Actualizar la informaci n del cartesian_msg (EF)
169             cartesian_msg.header = pose.header
170             cartesian_msg.header.frame_id = "iiwa_link_0"
171             cartesian_msg.pose.position.x = Pef[0]
172             cartesian_msg.pose.position.y = Pef[1]
173             cartesian_msg.pose.position.z = Pef[2]
174             cartesian_msg.pose.orientation.w = q[0]
175             cartesian_msg.pose.orientation.x = q[1]
176             cartesian_msg.pose.orientation.y = q[2]
177             cartesian_msg.pose.orientation.z = q[3]
178 
179         if(j == 1): # Si no es la primera posicion que se recibe:
180 
181             # Nueva posicion de la punta de la herramienta con el incremento
182             Ptn = [self.Ptcp[0]+Ph1, self.Ptcp[1]+Ph2, self.Ptcp[2]+Ph3]
183 
184             # Nueva direccion en el eje z de la herramienta
185             zn = self.Pf - Ptn
186 
187             # Distancia del punto de fulcro a la nueva posicion de la pinza
188             Mzn = math.sqrt(zn[0]*zn[0] + zn[1]*zn[1] + zn[2]*zn[2])
189             ro = self.tool_length - Mzn
190 
191             # Nueva posicion del efector final del robot
192             Pn = self.Pf + ro*zn/Mzn
193 
194             # El eje z de la herramienta ya lo tenemos, pero se hace unitario.
195             znn = [-zn[0]/Mzn, -zn[1]/Mzn, -zn[2]/Mzn]
196 
197             # Calcular un vector perpendicular al eje Z est ndar y a znn
198             xnn = np.cross([0, 0, 1], znn)
199             xnn /= np.linalg.norm(xnn)  # Normaliza el vector
200 
201             # Calcula un segundo vector perpendicular a znn y xnn
202             ynn = np.cross(znn, xnn)
203             ynn /= np.linalg.norm(ynn)  # Normaliza el vector
204 
205             # Crea una matriz de rotacion a partir de los vectores unitarios
```

```
206             M = np.column_stack((xnn, ynn, znn))
207 
208             # Calcular la orientacion en cuaternios a partir de matriz de rotacion
209             q = quaternion_from_matrix(M)
210 
211             # Actualizar la informacion del cartesian_msg (nuevo EF)
212             cartesian_msg.header = pose.header
213             cartesian_msg.header.frame_id = "iiwa_link_0"
214             cartesian_msg.pose.position.x = Pn[0]
215             cartesian_msg.pose.position.y = Pn[1]
216             cartesian_msg.pose.position.z = Pn[2]
217             cartesian_msg.pose.orientation.w = q[0]
218             cartesian_msg.pose.orientation.x = q[1]
219             cartesian_msg.pose.orientation.y = q[2]
220             cartesian_msg.pose.orientation.z = q[3]
221 
222             # Se muestran por pantalla los datos
223             print("Posicion EF tras incremento: ", Pn)
224             print("Posicion TCP tras incremento: ", Ptn)
225 
226             # Imprime la distancia en valor absoluto
227             distancia = np.linalg.norm(Pn - Ptn)
228             print("Longitud de la herramienta: ",abs(distancia))
229 
230             # Se guardan los datos en el archivo .csv
231             # guardar_datos_csv([Pn[0], Pn[1], Pn[2]], nombre_archivo)
232             # guardar_datos_csv([Ptn[0], Ptn[1], Ptn[2]], nombre_archivo)
233 
234             #COMPROBACIONES
235 
236             # Para comprobar que se han hecho bien los calculos y el movimiento alrededor del
    punto de fulcro se comprueba
237             # si el punto de fulcro (self.Pf) se encuentra en la herramienta en la nueva posicion,
     es decir,
238             # entre Pn y Ptn o lo que es lo mismo, entre las nuevas posiciones de EF y TCP
239 
240             epsilon = 1e-6  # Valor peque o para manejar errores de punto flotante
241 
242             # Calcula las distancias entre los puntos
243             dist1 = np.linalg.norm(np.array(Pn) - np.array(self.Pf))
244             dist2 = np.linalg.norm(np.array(Ptn) - np.array(self.Pf))
245             total_dist = np.linalg.norm(np.array(Ptn) - np.array(Pn))
246 
247             # Comprueba si el punto fijado como punto de fulcro se encuentra entre Pn y Ptn,
    teniendo en cuenta una peque a tolerancia
248             is_between=abs(dist1 + dist2 - total_dist) < epsilon
249 
250             if is_between:
251                 print("El punto de fulcro (Pf) est  entre EF (Pn) y TCP (Ptn).")
252             else:
253                 print("El punto de fulcro (Pf) no est  entre EF (Pn) y TCP (Ptn)")
254             print(" ")
255 
256 
257         # Publicar el mensaje
258         self.cartesian_pub.publish(cartesian_msg)
```

D. Archivo iiwa_surgery_params.yalm

```
# Establece el modo de simulaci n. 'true' para el modo de simulaci n y 'false' para el modo real.
simulation_mode: true

# Define la longitud de la herramienta respecto del efector final (EF).
tool_length: 0.1275

# Especifica la orientaci n de la herramienta en radianes con formato [roll, pitch, yaw].
tool_orientation: [0.0, 0.0, 0.0]

# Indica en qu  punto de la longitud de la herramienta se encuentra el punto de fulcro. Debe estar entre 0 y 1, donde 0 representa el extremo final (EF) y 1 el Tool Center Point (TCP).
fulcrum_fi: 0.5

# Define la direcci n IP del robot. Ejemplo: "192.168.1.100".
robot_ip: ""

# Establece el modo de trabajo del robot: "pivot" si est  en modo pivotaje alrededor del punto de fulcro y "free" para el modo de funcionamiento libre.
work_mode: "pivot"
```

E. Archivo iiwa_surgery.launch

```
<launch>
    <!-- Define el argumento simulation_mode con un valor predeterminado de false -->
    <arg name="simulation_mode" default="false" />

    <!-- Carga los parametros del archivo iiwa_surgery_params.yaml -->
    <rosparam file="$(find iiwa_surgery)/config/iiwa_surgery_params.yaml" command="load"/>

    <!-- Ejecutar este grupo si simulation_mode es true -->
    <group if="$(arg simulation_mode)">
        <!-- Ejecutar en modo de simulaci n -->

        <!-- Iniciar Gazebo con Sunrise para la simulacion -->
        <include file="$(find iiwa_gazebo)/launch/iiwa_gazebo_with_sunrise_edited.launch" />

        <!-- Ejecutar el nodo iiwa_control_node -->
        <node name="iiwa_control_node" pkg="iiwa_surgery" type="iiwa_control_node.py" output="
    screen"/>
    </group>

    <!-- Ejecutar este grupo si simulation_mode es false -->
    <group unless="$(arg simulation_mode)">
        <!-- Ejecutar en modo real -->

        <!-- Ejecutar el nodo iiwa_control_node -->
        <node name="iiwa_control_node" pkg="iiwa_surgery" type="iiwa_control_node.py" output="
    screen"/>
    </group>
</launch>
```

F. Documentación Doxygen

Interfaz multimodal para KUKA LBR iiwa 7 R800

Generado por Doxygen 1.9.8

Capítulo 1

Índice de clases

1.1. Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

Capítulo 2

Índice de archivos

2.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

Capítulo 3

Documentación de clases

3.1. Referencia de la clase iiwa_surgery_class

Clase para controlar un robot KUKA LBR iiwa con fines quirúrgicos en un entorno de ROS.

Métodos protegidos

- __init__ (self, simulation_mode=True)

  Inicializa una instancia de la clase iiwa_surgery_class.
- move_cartesian (self, pose)

  Mueve el robot en el espacio cartesiano.
- move_cartesian_fulcrum (self, pose, increment_vector, j)

  Mueve el robot en el espacio cartesiano alrededor de un punto de fulcro.
- move_joint (self, joint_config)

  Realiza movimientos de las articulaciones del robot.
- set_fulcrum_fi (self, fulcrum_fi)

  Configura el punto de fulcro.
- set_robot_ip (self, robot_ip)

  Configura la dirección IP del robot.
- set_tool_data (self, tool_length, tool_orientation)

  Configura los datos de la herramienta.
- set_work_mode (self, work_mode)

  Configura el modo de trabajo del robot.

Atributos protegidos

- cartesian_pub

  Publicador ROS utilizado para enviar mensajes de posición cartesiana al robot.
- fulcrum_fi

  Valor que indica en que punto de la longitud de la herramienta se encuentra el punto de fulcro.
- joint_pub

  Publicador ROS utilizado para enviar mensajes de posición articular al robot.
- Pf

  Vector que almacena la posición del punto de fulcro en el espacio cartesiano.

- Ptcp

  Vector que almacena la posición del TCP (Tool Center Point) en el espacio cartesiano.
- robot_ip

  Cadena que representa la dirección IP del robot.
- simulation_mode

  Booleano que indica si el sistema está en modo de simulación.
- tool_length

  Valor que representa la longitud de la herramienta.
- tool_orientation

  Lista que almacena la orientación de la herramienta en radianes con el en formato [roll, pitch, yaw].
- work_mode

  Cadena que indica el modo de trabajo del robot ('free' o 'pivot').

3.1.1. Descripción detallada

Clase para controlar un robot KUKA LBR iiwa con fines quirúrgicos en un entorno de ROS.

Esta clase permite configurar el robot, establecer datos de la herramienta y realizar movimientos articulares y movimientos en el espacio cartesiano.

Configuración del robot:

- Esta clase se utiliza para controlar el robot KUKA LBR iiwa en un entorno de simulación con Gazebo o en un robot real.
- Es posible configurar la dirección IP del robot utilizando el método `set_robot_ip`.
- El modo de trabajo del robot se puede configurar como 'free' o 'pivot' utilizando el método `set_work_mode`.

Configuración de la herramienta:

- Utilizar el método `set_tool_data` para definir la longitud de la herramienta y su orientación. La herramienta a utilizar siempre se encuentra alineada con el eje Z del efector dinal (EF).

Movimiento articular preciso:

- El método `move_joint` permite realizar movimientos articulares proporcionando una configuración de las articulaciones del robot.

Movimiento libre en el espacio cartesiano:

- El método `move_cartesian` permite mover el robot en el espacio cartesiano proporcionando una posición y orientación deseada del TCP.

Movimiento en el espacio cartesiano alrededor de un punto de fulcro:

- El método `move_cartesian_fulcrum` permite mover el robot alrededor de un punto de fulcro. Proporcionar una posición y orientación deseada del TCP, un vector de incrementos y un indicador de posición.

3.1.2. Documentación de constructores y destructores

3.1.2.1. __init__()

```
__init__ (
              self,
              simulation_mode = True )  [protected]
```

Inicializa una instancia de la clase iiwa_surgery_class.

Parámetros

simulation_mode	(bool, opcional): Indica si se está ejecutando en modo simulación. El valor predeterminado es True.

3.1.3. Documentación de funciones miembro

3.1.3.1. move_cartesian()

```
move_cartesian (
              self,
              pose )  [protected]
```

Mueve el robot en el espacio cartesiano.

Parámetros

pose	(geometry_msgs.msg.PoseStamped): Posición y orientación deseada del Tool Center Point (TCP) en espacio cartesiano.

3.1.3.2. move_cartesian_fulcrum()

```
move_cartesian_fulcrum (
              self,
              pose,
              increment_vector,
              j )  [protected]
```

Mueve el robot en el espacio cartesiano alrededor de un punto de fulcro.

Para establecer el punto de fulcro (XYZ), este método realiza los siguientes pasos:

1. Si `j` es igual a 0 (indicando la primera posición), tras establecer las posiciones iniciales del EF y TCP, se calcula el punto de fulcro y guarda su posición.
2. Si `j` es igual a 1 (indicando posiciones sucesivas), calcula las nuevas posiciones del EF y TCP alrededor del punto de fulcro de acuerdo con los incrementos recibidos.

El punto de fulcro se establece inicialmente en la primera llamada a este método y se mantiene constante en posiciones sucesivas.

Parámetros

increment_vector	(list): Vector de incrementos [Ph1, Ph2, Ph3] para mover la herramienta.
j	(int): Indicador de posición, 0 para la posición inicial, 1 para las posiciones sucesivas.

3.1.3.3. move_joint()

```
move_joint (
              self,
              joint_config )  [protected]
```

Realiza movimientos de las articulaciones del robot.

Parámetros

joint_config	(iiwa_msgs.msg.JointPosition): Configuración de las articulaciones del robot.

3.1.3.4. set_fulcrum_fi()

```
set_fulcrum_fi (
              self,
              fulcrum_fi )  [protected]
```

Configura el punto de fulcro.

Parámetros

fulcrum_fi	(float): Posición del punto de fulcro, un valor entre 0 y 1.

3.1.3.5. set_robot_ip()

```
set_robot_ip (
              self,
              robot_ip )  [protected]
```

Configura la dirección IP del robot.

Parámetros

robot_ip	(str): Dirección IP del robot.

3.1.3.6. set_tool_data()

```
set_tool_data (
              self,
              tool_length,
              tool_orientation )  [protected]
```

Configura los datos de la herramienta.

Parámetros

tool_length	(float): Longitud de la herramienta.
tool_orientation	(list): Orientación de la herramienta en formato [roll, pitch, yaw].

3.1.3.7. set_work_mode()

```
set_work_mode (
              self,
              work_mode )  [protected]
```

Configura el modo de trabajo del robot.

Parámetros

work_mode	(str): Modo de trabajo del robot ('free' o 'pivot').

3.1.4. Documentación de datos miembro

3.1.4.1. cartesian_pub

```
cartesian_pub  [protected]
```

Publicador ROS utilizado para enviar mensajes de posición cartesiana al robot.

3.1.4.2. fulcrum_fi

```
fulcrum_fi  [protected]
```

Valor que indica en que punto de la longitud de la herramienta se encuentra el punto de fulcro.

Debe estar entre 0 y 1, siendo 0 el EF y el 1 el TCP.

3.1.4.3. joint_pub

```
joint_pub  [protected]
```

Publicador ROS utilizado para enviar mensajes de posición articular al robot.

3.1.4.4. Pf

```
Pf  [protected]
```

Vector que almacena la posición del punto de fulcro en el espacio cartesiano.

3.1.4.5. Ptcp

`Ptcp [protected]`

Vector que almacena la posición del TCP (Tool Center Point) en el espacio cartesiano.

3.1.4.6. robot_ip

`robot_ip [protected]`

Cadena que representa la dirección IP del robot.

Ej: "192.228.17.57"

3.1.4.7. simulation_mode

`simulation_mode [protected]`

Booleano que indica si el sistema está en modo de simulación.

3.1.4.8. tool_length

`tool_length [protected]`

Valor que representa la longitud de la herramienta.

3.1.4.9. tool_orientation

`tool_orientation [protected]`

Lista que almacena la orientación de la herramienta en radianes con el en formato [roll, pitch, yaw].

3.1.4.10. work_mode

`work_mode [protected]`

Cadena que indica el modo de trabajo del robot ('free' o 'pivot').

La documentación de esta clase está generada del siguiente archivo:

- TFM_JavierLara/iiwa_surgery/src/iiwa_control_class.py

3.2. Referencia de la clase iiwa_surgery_node_class

Clase que define el nodo ROS para el control del robot KUKA LBR iiwa con fines quirúrgicos.

Métodos protegidos

- __init__ (self)

  Constructor de la clase.
- cartesian_pose_callback (self, msg)

  Callback para recibir estados de posición cartesiana y posteriormente publicarlos en el topic iiwa_surgery/output/ef↩ _pose y tras trasnformación en iiwa_surgery/output/tcp_pose.
- joint_command_callback (self, msg)

  Callback para recibir comandos de posición articular y posteriormente enviarlos al método move_joint de la clase iiwa_surgery_class.
- joint_position_callback (self, msg)

  Callback para recibir estados de posición articular y posteriormente publicarlos en el topic iiwa_surgery/output/joints.
- joint_state_callback (self, msg)

  Callback para recibir estados de posición articular y posteriormente publicarlos en el topic iiwa_surgery/output/joints.
- pose_command_callback (self, msg)

  Callback para recibir comandos de posición cartesiana y posteriormente enviarlos a los métodos move_cartesian o move_cartesian_fulcrum de la clase iiwa_surgery_class dependiendo si el movimiento es libre o de pivoteo.
- run (self)

  Ejecuta el bucle principal mientras el nodo está en funcionamiento.

Atributos protegidos

- cartesian_pose_sub

  Suscriptor ROS utilizado para recibir estados de posición cartesiana.
- ef_pose_pub

  Publicador ROS utilizado para enviar estados de posición final del efector (EF) del robot.
- first_position

  Variable que se utiliza para almacenar la primera posición recibida en el callback pose_command_callback.
- fulcrum_fi

  Valor que indica en que punto de la longitud de la herramienta se encuentra el punto de fulcro.
- iiwa

  Es una instancia de la clase iiwa_surgery que se utiliza para controlar el robot iiwa.
- joint_position_sub

  Suscriptor ROS utilizado para recibir estados de posición articular.
- joint_pub

  Publicador ROS utilizado para enviar mensajes de posición articular.
- joint_state_sub

  Suscriptor ROS utilizado para recibir estados de posición articular.
- joint_sub

  Suscriptor ROS utilizado para recibir comandos de posición articular.
- pose_sub

  Suscriptor ROS utilizado para recibir comandos de posición cartesiana.
- robot_ip

  Almacena la dirección IP del robot iiwa.
- simulation_mode

  Indica si el robot está en modo de simulación o no, siendo True para simulación y False para el robot real.
- tcp_pose_pub

  Publicador ROS utilizado para enviar estados de posición del Tool Center Point (TCP).
- tool_length

  Almacena la longitud de la herramienta utilizada en el robot.
- tool_orientation

  Lista que almacena la orientación de la herramienta en radianes con el en formato [roll, pitch, yaw].
- work_mode

  Indica el modo de trabajo predeterminado del robot ("free" para movimiento libre y "pivot" para movimiento alrededor de punto de fulcro).

3.2.1. Descripción detallada

Clase que define el nodo ROS para el control del robot KUKA LBR iiwa con fines quirúrgicos.

Es importante destacar que los parámetros de configuración utilizados en esta clase se obtienen del archivo iiwa_surgery_params.yaml, los cuales son cargados en el sistema de parámetros de ROS a partir del archivo de lanzamiento iiwa_surgery.launch. A continuación, se muestra cómo se enlazan estos parámetros con las variables de la clase:

- `simulation_mode`: Indica si el robot está en modo de simulación o no (True para simulación, False para el robot real).
- `tool_length`: Almacena la longitud de la herramienta utilizada en el robot.
- `tool_orientation`: Lista que almacena la orientación de la herramienta en radianes en el formato [roll, pitch, yaw].
- `fulcrum_fi`: Valor que indica en qué punto de la longitud de la herramienta se encuentra el punto de fulcro, siendo 0 para el EF y 1 para el TCP.
- `robot_ip`: Almacena la dirección IP del robot iiwa.
- `work_mode`: Indica el modo de trabajo predeterminado del robot ("free" para movimiento libre y "pivot" para movimiento alrededor de un punto de fulcro).

3.2.2. Documentación de constructores y destructores

3.2.2.1. __init__()

```
__init__ (
              self )  [protected]
```

Constructor de la clase.

3.2.3. Documentación de funciones miembro

3.2.3.1. cartesian_pose_callback()

```
cartesian_pose_callback (
              self,
              msg )  [protected]
```

Callback para recibir estados de posición cartesiana y posteriormente publicarlos en el topic iiwa_surgery/output/ef_pose y tras trasnformación en iiwa_surgery/output/tcp_pose.

Parámetros

msg	Mensaje de estado de posición cartesiana (PoseStamped).

3.2.3.2. joint_command_callback()

```
joint_command_callback (
              self,
              msg )  [protected]
```

Callback para recibir comandos de posición articular y posteriormente enviarlos al método move_joint de la clase iiwa_surgery_class.

Parámetros

msg	Mensaje de posición articular (JointPosition).

3.2.3.3. joint_position_callback()

```
joint_position_callback (
              self,
              msg )  [protected]
```

Callback para recibir estados de posición articular y posteriormente publicarlos en el topic iiwa_↩ surgery/output/joints.

Parámetros

msg	Mensaje de estado de posición articular (JointPosition).

3.2.3.4. joint_state_callback()

```
joint_state_callback (
              self,
              msg )  [protected]
```

Callback para recibir estados de posición articular y posteriormente publicarlos en el topic iiwa_↩ surgery/output/joints.

Parámetros

msg	Mensaje de estado de posición articular (JointState).

3.2.3.5. pose_command_callback()

```
pose_command_callback (
              self,
              msg )  [protected]
```

Callback para recibir comandos de posición cartesiana y posteriormente enviarlos a los métodos move_cartesian o move_cartesian_fulcrum de la clase iiwa_surgery_class dependiendo si el movimiento es libre o de pivoteo.

Parámetros

msg	Mensaje de posición cartesiana (PoseStamped).

3.2.3.6. run()

```
run (
              self )  [protected]
```

Ejecuta el bucle principal mientras el nodo está en funcionamiento.

3.2.4. Documentación de datos miembro

3.2.4.1. cartesian_pose_sub

```
cartesian_pose_sub  [protected]
```

Suscriptor ROS utilizado para recibir estados de posición cartesiana.

3.2.4.2. ef_pose_pub

```
ef_pose_pub  [protected]
```

Publicador ROS utilizado para enviar estados de posición final del efector (EF) del robot.

3.2.4.3. first_position

```
first_position  [protected]
```

Variable que se utiliza para almacenar la primera posición recibida en el callback pose_command_callback.

3.2.4.4. fulcrum_fi

```
fulcrum_fi  [protected]
```

Valor que indica en que punto de la longitud de la herramienta se encuentra el punto de fulcro.

Debe estar entre 0 y 1, siendo 0 el EF y el 1 el TCP.

3.2.4.5. iiwa

```
iiwa  [protected]
```

Es una instancia de la clase iiwa_surgery que se utiliza para controlar el robot iiwa.

3.2.4.6. joint_position_sub

`joint_position_sub [protected]`

Suscriptor ROS utilizado para recibir estados de posición articular.

3.2.4.7. joint_pub

`joint_pub [protected]`

Publicador ROS utilizado para enviar mensajes de posición articular.

3.2.4.8. joint_state_sub

`joint_state_sub [protected]`

Suscriptor ROS utilizado para recibir estados de posición articular.

3.2.4.9. joint_sub

`joint_sub [protected]`

Suscriptor ROS utilizado para recibir comandos de posición articular.

3.2.4.10. pose_sub

`pose_sub [protected]`

Suscriptor ROS utilizado para recibir comandos de posición cartesiana.

3.2.4.11. robot_ip

`robot_ip [protected]`

Almacena la dirección IP del robot iiwa.

3.2.4.12. simulation_mode

`simulation_mode [protected]`

Indica si el robot está en modo de simulación o no, siendo True para simulación y False para el robot real.

3.2.4.13. tcp_pose_pub

`tcp_pose_pub [protected]`

Publicador ROS utilizado para enviar estados de posición del Tool Center Point (TCP).

3.2.4.14. tool_length

`tool_length [protected]`

Almacena la longitud de la herramienta utilizada en el robot.

3.2.4.15. tool_orientation

`tool_orientation [protected]`

Lista que almacena la orientación de la herramienta en radianes con el en formato [roll, pitch, yaw].

3.2.4.16. work_mode

`work_mode [protected]`

Indica el modo de trabajo predeterminado del robot ("free" para movimiento libre y "pivot" para movimiento alrededor de punto de fulcro).

La documentación de esta clase está generada del siguiente archivo:

- TFM_JavierLara/iiwa_surgery/src/iiwa_control_node.py

Capítulo 4

Documentación de archivos

4.1. Referencia del archivo TFM_JavierLara/iiwa_surgery/src/iiwa_control_class.py

Clases

- class iiwa_surgery_class

  Clase para controlar un robot KUKA LBR iiwa con fines quirúrgicos en un entorno de ROS.

Espacios de nombres

- namespace iiwa_control_class

4.2. Referencia del archivo TFM_JavierLara/iiwa_surgery/src/iiwa_control_node.py

Clases

- class iiwa_surgery_node_class

  Clase que define el nodo ROS para el control del robot KUKA LBR iiwa con fines quirúrgicos.

Espacios de nombres

- namespace iiwa_control_node

Variables

- node = iiwa_surgery_node_class()

Índice alfabético