Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

MODELADO CINEMÁTICO INVERSO Y SEGUIMIENTO DE TRAYECTORIAS DE UN MANIPULADOR BHR

1524

Autor:

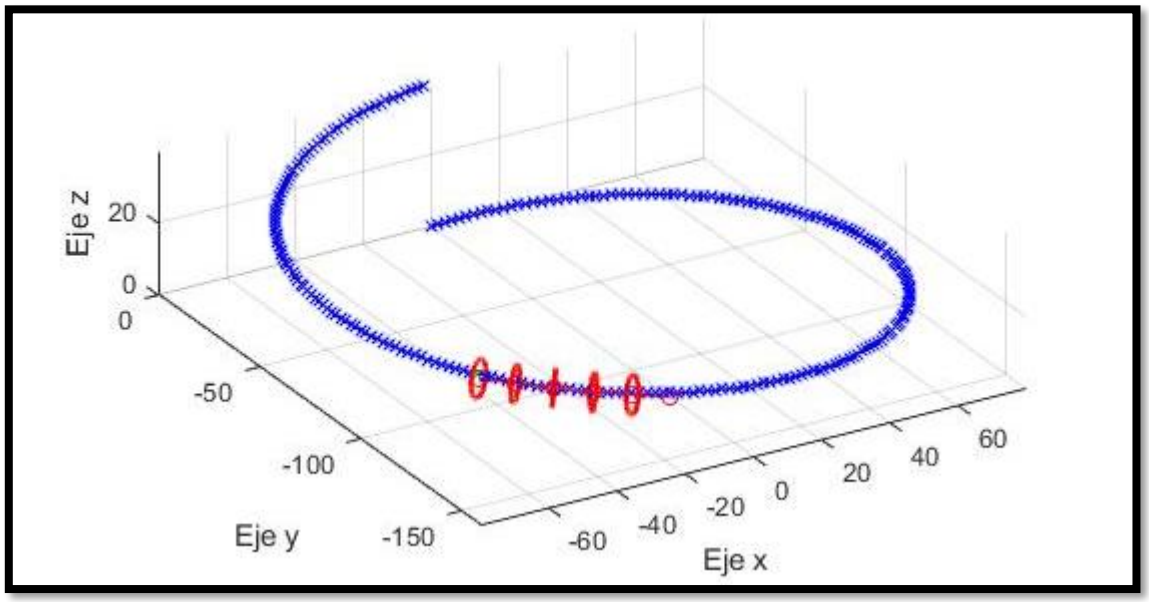
Martínez Pablos, Carlos

Grado en Ingeniería Mecánica

Tutores:

Pérez Rueda, Mª Ángeles. Dpto. de CMeIM, EGI, ICGyF, IM, IPF.

González Sánchez, José Luis Dpto. de ISA



UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Grado en Ingeniería Mecánica

MODELADO CINEMÁTICO INVERSO Y SEGUIMIENTO DE TRAYECTORIAS DE UN MANIPULADOR BHR

Autor:

Martínez Pablos, Carlos

Tutores:

Pérez Rueda, Mª Ángeles
Dpto. de CMeIM, EGI, ICGyF, IM, IPF.

González Sánchez, José Luis
Dpto. de ISA

Valladolid, Enero de 2021

Agradecimientos

A mis tutores Mª Ángeles Pérez y José Luis González por su tiempo, consejos y esfuerzo a pesar de las circunstancias que nos acontecen.

A mi familia y amigos por su ayuda durante esta etapa y las que han precedido en las que todos han colaborado de diferentes maneras.

Muchas gracias.

Resumen

El objetivo del presente trabajo es el desarrollo de un algoritmo de cinemática inversa que permita el seguimiento de trayectorias para posteriormente aplicarlo sobre un robot BHR (Binary Hyper-Redundant) ápodo y modular enfocado a la biomedicina, concretamente al campo de la endoscopia. Se definirán los parámetros del robot necesarios para el desarrollo del programa. Se empleará MATLAB para crear el algoritmo y se partirá como base del algoritmo de cinemática inversa "Forward And Backwards Reaching Inverse Kinematics" (FABRIK) haciendo uso del álgebra de cuaternios para las transformaciones. Se aplica el algoritmo sobre diferentes trayectorias y se comentan los resultados.

Palabras Clave

Robot ápodo, modular e hiper-redundante, robot BHRM, cuaternios duales, algoritmo FABRIK, seguimiento de trayectorias.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. Introducción

1.1 Contextualización

La industria de la robótica ha experimentado grandes avances tecnológicos en las últimas décadas y estos avances han sido aprovechados para mejorar muchos otros campos de la ciencia. Uno de los que se ha beneficiado de la robótica es la medicina. Se ha creado una gran variedad de robots para diferentes procedimientos medicinales, como por ejemplo cirugías o también prótesis. El trabajo conjunto de profesionales de la medicina junto con los robots combina la precisión y versatilidad de los robots con la valoración de la persona que esté operando la máquina. Así se consiguen disminuir diversos riesgos como pueden ser hemorragias, intolerancia a las invasiones o los perjuicios que pueden ocasionar la exposición continuada a rayos X requerida en ciertas operaciones [Abovitz, 2001].

A pesar de todos estos avances algunos procedimientos aún se realizan de forma manual como son los diagnósticos por endoscopia. Esta técnica conlleva ciertos riesgos. Principalmente son las perforaciones de las paredes intestinales con la consecuente hemorragia y la intolerancia del cuerpo del paciente a una invasión de ese tipo. Debido a que generalmente se realiza de forma manual existe un gran margen de mejora en el movimiento del endoscopio, facilitando la toma de imágenes y de tejido y al mismo tiempo reduciendo los riesgos previamente citados [Muñoz, 2013].

La forma de serpiente o gusano que debe tener el robot híper-redundante ápodo para acceder a través del tracto intestinal ya nos invita a pensar que el movimiento del robot debe ser similar a los métodos que emplean estos animales. Existen estudios sobre robots-híper-redundantes que imitan los movimientos de dichos animales. Estos movimientos presentan la ventaja de que le permiten al robot una gran adaptación a las características del medio por el que se mueve y a las irregularidades que puedan aparecer. Si se consigue diseñar un robot con un diámetro reducido se podrá tener una gran maniobrabilidad y consiguientemente una mayor eficacia en el diagnóstico de enfermedades intestinales [Muñoz, 2013].

1.2 Justificación del proyecto

El cáncer colorrectal es el segundo tipo de cáncer que más muertes provoca en España al año llegando a las 15.000 y siendo 37.000 las personas afectadas en el año 2018. Esta enfermedad afecta especialmente a personas de más de 50 años [AECC,15/04/20]. Como por el momento no se ha logrado avanzar demasiado en las estrategias de tratamiento, la mejor herramienta para aumentar las posibilidades de superar el cáncer se basa en el diagnóstico precoz. El diagnóstico en un estadio

precoz ha demostrado que la supervivencia es del 85% a los 5 años [Andrés Cervantes, 2003].

Esta enfermedad es solo una de las múltiples afecciones que se pueden sufrir a lo largo aparato digestivo. Por este motivo es de gran interés profundizar en la búsqueda de nuevos métodos de detección endoscópica y mejorar la toma de imágenes evitando los puntos ciegos y complicaciones que puedan aparecer por estrechamientos. De esta manera se podrán proporcionar mayores garantías de un diagnóstico correcto.

1.3 Objetivos del proyecto

El objetivo principal del trabajo será crear un algoritmo empleando el programa MATLAB que permita seguir trayectorias a un robot modular hiper-redundante. Este trabajo es parte de un proyecto del Instituto de las Tecnologías Aplicadas de la Producción (ITAP) que pertenece a la Universidad de Valladolid. El diseño del robot aún no está completado por lo que los objetivos se establecerán teniendo en cuenta que el diseño puede cambiar y que ciertos aspectos como apartado de materiales aún no ha sido abordado.

- Realizar una investigación sobre el estado actual de la robótica aplicada a las exploraciones endoscópicas.

- Describir la forma y características que tendrá el robot sobre el que se realizarán las simulaciones.

- Comparar diferentes estrategias de cálculo de la cinemática inversa que proporcionen el control cinemático del robot con el mínimo error.

- Añadir condiciones y restricciones al algoritmo de cinemática inversa seleccionado para que simule lo más fielmente posible el robot.

- Diseñar un programa para probar el seguimiento de trayectorias sobre el que aplicar el algoritmo y comprobar que funciona correctamente siguiendo de forma adecuada las trayectorias requeridas.

1.4 Estructura del proyecto

El proyecto tendrá la siguiente estructura:

- CAPÍTULO 1: Se realiza una breve introducción a la problemática que se quiere abordar y se exponen los objetivos del Trabajo de Fin de Grado.

- CAPÍTULO 2: Se exponen los diferentes temas que engloba el proyecto: robótica, anatomía, fisiología y afecciones del aparato digestivo, los diferentes tipos de robot que existen hoy en día para realizar endoscopias.

- CAPÍTULO 3: Se realiza una breve explicación de las matrices de transformación homogéneas y una explicación más extensa de cuaternios y cuaternios duales.

- CAPÍTULO 4: Se describe la constitución del robot y los movimientos que puede realizar.

- CAPÍTULO 5: Se realiza una comparación entre los algoritmos de cinemática inversa FABRIK y CCD para decidir cuál es el más apropiado para desarrollar el programa.

- CAPÍTULO 6: Se optimiza paso a paso el algoritmo hasta que cumpla los requisitos necesarios para recrear fielmente los movimientos que puede realizar el robot.

- CAPÍTULO 7: Se trata de seguir diferentes trayectorias empleando el algoritmo optimizado.

- CAPÍTULO 8: Se exponen las conclusiones obtenidas y los diferentes aspectos que pueden trabajar las personas que continúen con el proyecto.

CAPÍTULO 2

2. Estado del arte

2.1 Robótica

2.1.1 Historia de la robótica

Desde el origen de las primeras civilizaciones los seres humanos han sentido curiosidad por las máquinas que imitan los movimientos de los seres humanos y animales. Estas máquinas denominadas autómatas tenían principalmente fines lúdicos y decorativos, como es el caso del *Gallo de Estrasburgo* (1352) que se encontraba en el reloj de la ciudad y cuando daban las horas movía el pico y las alas. En los siglos posteriores se realizaron diversos autómatas que se accionaban de diversas maneras como poleas o de forma hidráulica.

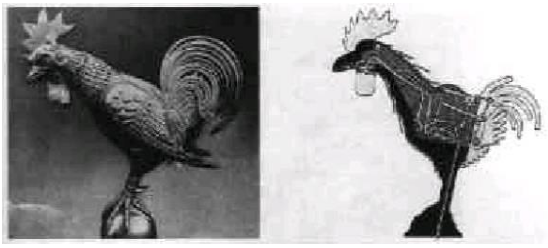


Figura 2.1 Gallo de Estrasburgo

A finales del siglo XVIII comenzaron a crearse máquinas destinadas a la fabricación textil en lugar de al entretenimiento. En 1801 se crea el telar de Jacquard. Esta máquina incorporó el programa de las acciones mediante una cinta de papel perforado. A partir de aquí comenzaron a implementarse las automatizaciones en algunas de las máquinas que se creaban.

La palabra robot apareció por primera vez en 1921 la obra de teatro *Rossum's Universal Robot* del escritor Karel Capek. La palabra viene de *robota* que es una palabra eslava que se refiere al trabajo de manera forzada. Posteriormente esta palabra sería utilizada en otras obras de diferentes autores de ciencia ficción afianzándose la palabra robot para referirse a máquinas autónomas.

En 1945 el escritor Isaak Asimov enunció en la revista '*Galaxy Science Fiction*' las tres leyes de la robótica que son las siguientes:

1. Un robot no puede perjudicar a un ser humano, ni con su inacción permitir que un ser humano sufra daño.
2. Un robot ha de obedecer las órdenes recibidas de un ser humano, excepto si tales órdenes entran en conflicto con la primera ley.

3. Un robot debe proteger su propia existencia mientras tal protección no entre en conflicto con la primera o segunda ley.

Posteriormente en 1985 el propio Asimov incorporaría la ley cero de la robótica que dice que un robot no puede lastimar a la humanidad o, por falta de acción permitir que la humanidad sufra daño. Esta ley tendría mayor prioridad que la primera, anteponiendo el bien comunitario al individual [Barrientos, 1997].

En el ambiente industrial se comenzaron utilizando los telemanipuladores en que eran máquinas que requerían de un operario para manipularlas. Debido a las ventajas de que las máquinas operasen por sí mismas sin necesidad de un operario se comenzaron a automatizar, convirtiéndose ahora sí en robots. De esta manera, bastaba con introducir el programa y en caso de querer que el robot realizase otras operaciones simplemente valdría con introducir el programa nuevo.

Desde que en 1954 se solicitase la primera patente de un dispositivo robótico, se han ido creando diferentes tipos de robot para las miles de aplicaciones industriales que hay. Actualmente, existen robots de todo tipo de tamaños y formas que facilitan ciertas operaciones e incluso realizan algunas que los humanos no podríamos hacer.

2.1.2 Definición de robot

Una de las definiciones más aceptadas en el mercado europeo es la de la *Asociación de Industrias Robóticas* (RIA) según la cual:

- Un robot industrial es un manipulador multifuncional reprogramable, capaz de mover materias piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas.

La Organización Internacional de Estándares incorporó la definición dada por la RIA con una ligera modificación. Y es que además de lo definido según la RIA para que sea considerado un robot industrial, el robot debe tener varios grados de libertad.

En último lugar, la Federación Internacional de Robótica (IFR) distingue entre robot industrial de manipulación y otros robots:

Por robot industrial de manipulación se entiende una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o

dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento.

2.1.3 Clasificación de los robots

Existen muchas formas de clasificar los robots, nosotros los clasificaremos según su construcción. Encontramos dos grupos *blandos* y *duros* dentro de los cuales se hace la subdivisión en *rígidos*, *hiper-redundantes discretos* y *flexibles*. En la Figura 2.2 se muestra el esquema de un robot rídigo y el de un robot flexible [Trivedi,D. 2008].

Figura 2.2 Esquema de robot rígido(izq) y de robot flexible (dcha)

Para comprender mejor la siguiente tabla se definirán previamente algunas de las características de los robots.

- Grados de libertad (gdl): Cada uno de los movimientos independientes que puede realizar una articulación respecto de la anterior. El número de grados de libertad será la suma de los grados de libertad de todas sus articulaciones.

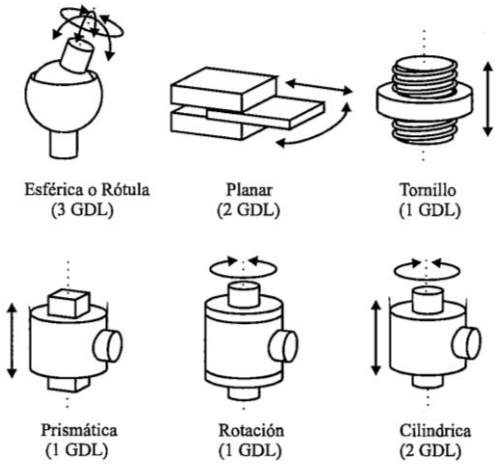


Figura 2.3 Distintos tipos de articulaciones para robots

- Actuadores: Mecanismo encargado de modificar la posición u orientación de una articulación respecto de la anterior.

- Precisión: Capacidad del robot de repetir un movimiento múltiples veces y obtener el mismo resultado. (Es diferente de exactitud, si el robot tiene una precisión alta y una exactitud baja se podría calibrar, en el caso contrario no).
- Movilidad: Capacidad de movimiento y comportamiento dinámico.
- Cálculo de la posición: Capacidad del robot para conocer donde se encuentra el cada punto.

	Rígido	Hiper-redundante discreto	Rígido continuo	Flexible
Propiedades				
Gdl	Pocos	Muchos	Infinitos	Infinitos
Actuadores	Pocos, discretos	Muchos, discretos	Continuos	Continuos
Deformación del material	Ninguna	Ninguna	Pequeña	Grande
Materiales	Metales, plásticos	Metales, plásticos	Aleaciones con memoria de forma	Goma, polímeros electroactivos
Capacidades				
Precisión	Muy alta	Alta	Alta	Baja
Capacidad de carga	Alta	Baja	Baja	Muy baja
Seguridad	Peligroso	Peligroso	Peligroso	Seguro
Movilidad	Baja	Alta	Alta	Alta
Ambiente de trabajo	Solo estructurado	Estructurado y desestructurado	Estructurado y desestructurado	Estructurado y desestructurado
Objetos manipulables	Tamaño fijo	Tamaño variable	Tamaño variable	Tamaño variable
Adaptabilidad a obstáculos	Ninguna	Alta	Media	Muy alta
Diseño				
Controlabilidad	Fácil	Medio	Difícil	Difícil
Cálculo de trayectoria	Fácil	Difícil	Muy difícil	Muy difícil
Cálculo de la posición	Fácil	Difícil	Muy difícil	Muy difícil
Inspiración	Articulaciones de mamíferos	Serpientes, peces		Muscular hydrostats (Lengua, trompa de elefante,gusanos)

Tabla 2.1 Características de los diferentes tipos de robots

2.2 Anatomía, fisiología y afecciones del aparato digestivo

El aparato digestivo se encarga de la preparación y absorción de los alimentos mediante la digestión para así alimentar a todas las células que forman el

organismo. Este proceso involucra a diferentes órganos que el alimento va atravesando desde que entra por la boca hasta que se expulsa en forma de residuo por el ano. La Figura 2.4 muestra los diferentes órganos que componen el aparato digestivo. Pese a que lo conforman más de una decena de órganos, hay muchos que no se encuentran en el tubo digestivo como por ejemplo el hígado o el páncreas. Por lo tanto, nos centraremos en los que sí se encuentran en el tracto gastrointestinal que serán los que en una endoscopia será posible recorrer y observar [Thibodeau, G. 2007].

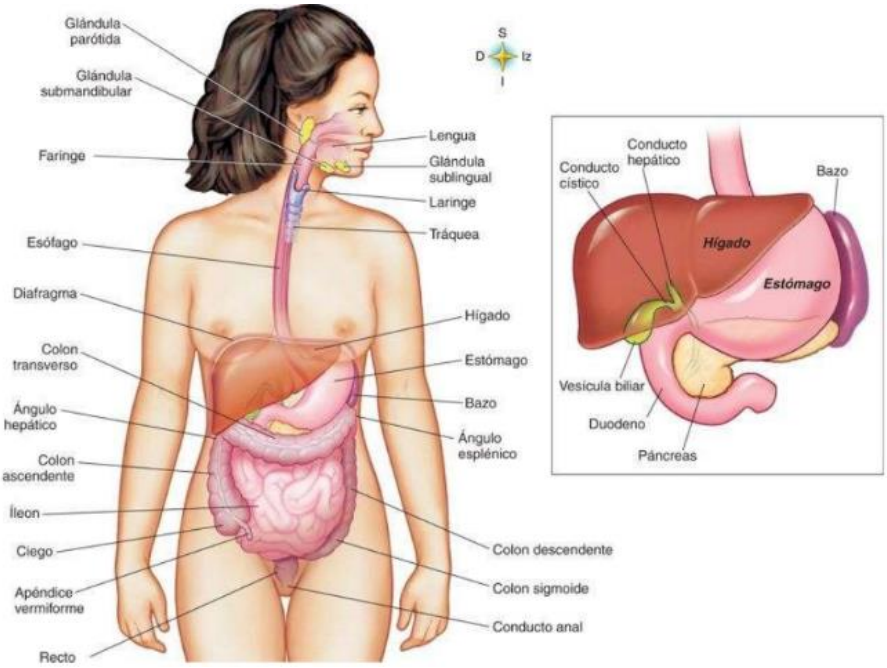


Figura 2.4 Órganos del aparato digestivo

2.2.1 Anatomía y fisiología del tubo digestivo

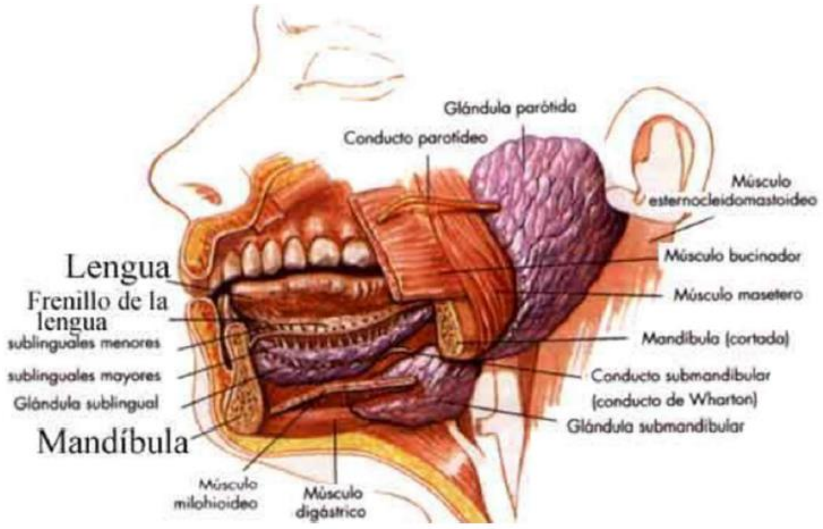


Figura 2.5 Anatomía de la cavidad oral

- Cavidad oral

  Es la parte del cuerpo por la que se ingieren los alimentos. En la cavidad oral se encuentran los dientes, la lengua y las glándulas salivales que se encargan de la digestión mecánica, es decir, trituración e insalivación de los alimentos para formar el bolo alimenticio. Figura 2.5 [Thibodeau, G. 2007].

- Faringe

  La faringe es un conducto une la boca con el esófago y tráquea, cambiando la posición de la epiglotis se permite el paso por los dos conductos como en el caso del aire al respirar o solo hacia el esófago si se trata del bolo. Como se observa en la Figura 2.6 la faringe se divide en tres partes. La parte superior es la nasofaringe que es la que conecta con la cavidad nasal. La orofaringe conecta con la cavidad oral. Por último, la hipofaringe es la parte inferior donde actúa la epiglotis. Una vez el bolo atraviesa la faringe entra en el tubo digestivo propiamente dicho, ya que las partes anteriores no tienen funciones exclusivas en la digestión, sino que también participan en la respiración [Thibodeau, G. 2007].

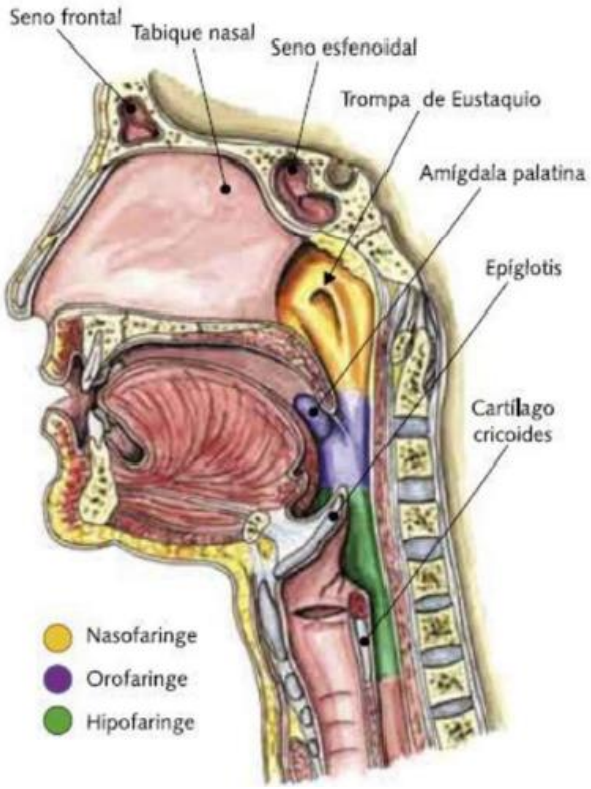


Figura 2.6 Anatomía de la faringe

- Esófago

  El esófago es el primer segmento del tubo digestivo. Se trata de un conducto de aproximadamente 25 cm de longitud que conecta la faringe con el estómago. En sus extremos está protegido por un esfínter muscular. El

esfínter superior evita la entrada de aire. El esfínter inferior protege del reflujo gástrico [Thibodeau, G. 2007].

- Estómago

El estómago es un ensanchamiento del tubo digestivo. Tiene forma de saco y su tamaño varía dependiendo de si contiene alimentos o no. Cuando los alimentos se encuentran en su interior se dilata y alcanza un volumen de entre 1-1,5 litros. Su función es mezclar el bolo alimenticio con los jugos gástricos que ayudan a digerir los alimentos formando así el quimo. También almacena los alimentos hasta que pueden pasar a la siguiente zona del tubo digestivo. Además, con los ácidos que contienen los jugos gástricos se ayuda a destruir las bacterias que se han ingerido con los alimentos [Thibodeau, G. 2007].

Las paredes interiores del estómago están recubiertas por una mucosa que las protege de los ácidos. El estómago está formado por músculos que se encargan de mezclar los alimentos de modo que el quimo quede homogéneo [Thibodeau, G. 2007].

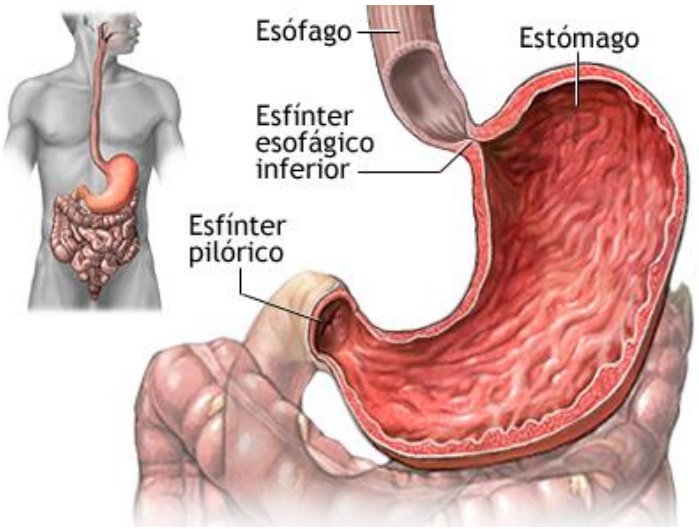


Figura 2.7 Anatomía del estómago

- Intestino delgado

El intestino delgado es un tubo de aproximadamente 6 m de largo y 2,5 cm de diámetro. Ocupa la mayor parte del abdomen y se encuentra plegado de forma laberíntica. El intestino se puede dividir en tres partes, según es recorrido por el quimo serían el duodeno, el yeyuno y el íleon. El duodeno mide aproximadamente 25 cm. Corresponden a la primera parte del intestino que conecta con el estómago. Tiene forma de 'C'. El yeyuno comienza donde el intestino gira bruscamente hacia abajo y hacia delante. Mide aproximadamente 2,5 m. Por último, se encuentra el íleon que corresponde a los últimos 3,5 m del intestino delgado. No existe una división clara de donde termina el yeyuno y comienza el íleon.

El interior del intestino está formado por millones de vellosidades que multiplican la superficie. Estas vellosidades se encargan de absorber los nutrientes a medida que se va haciendo la digestión a lo largo del intestino [Thibodeau, G. 2007].

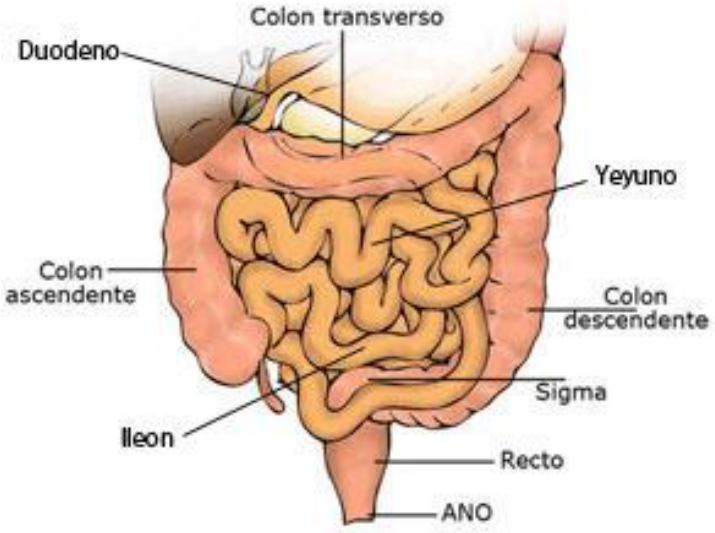


Figura 2.8 Anatomía de los intestinos delgado y grueso

- Intestino grueso

  El intestino grueso es la última parte del tubo digestivo. Mide entre 1,5-1,8 m y tiene un diámetro de aproximadamente 6 cm. El intestino se divide en tres partes: ciego, colon y recto.

  El primer tramo es el ciego, es una cavidad sin salida, de ahí el nombre, en cuyo extremo se encuentra el apéndice. El segundo tramo es el colon, que a su vez se divide en cuatro partes: ascendente, transversal descendente y sigmoide. El primer tramo es el ascendente. Su nombre se debe a que parte del cuadrante inferior derecho del abdomen hacia el cuadrante superior derecho, es decir, asciende por el abdomen. Es el que comunica con el intestino delgado a través de la válvula ileocecal que permite solo el paso de materia del intestino delgado al grueso. El segundo tramo, el transversal recorre el abdomen transversalmente del cuadrante superior derecho al cuadrante superior izquierdo. El último tramo del colon es el sigmoide, en la Figura 2.8 aparece como 'Sigma', que realiza un giro y termina en la parte inferior baja del abdomen donde conecta con el recto. Por último, se encuentra el recto que un tramo de 15 cm que desciende encontrándose al final el ano por donde se expulsan las heces.

  A lo largo del intestino grueso se van absorbiendo el agua, minerales y vitaminas (K y B12) que contiene en quimo y en el recto se almacenan las heces hasta que se expulsan mediante la defecación [Thibodeau, G. 2007].

2.2.2 Afecciones del tubo digestivo

Debido a que la cavidad oral no requiere de endoscopia para examinarla no hablaremos de las enfermedades que le afectan. Por lo tanto, comenzaremos con la faringe.

- Faringe: La mayoría de los problemas se deben a catarros, infecciones víricas o bacterianas y también a reflujos. Sin embargo, también se pueden sufrir enfermedades más serias. Las principales son la amigdalitis, cáncer de faringe, crup (inflamación que principalmente afecta a niños pequeños y causa tos seca) y faringitis. Por lo general, de estas enfermedades solo se necesita un endoscopio para la prueba del cáncer, ya que además de examinar la zona visualmente, también se necesita realizar una biopsia.

- Esófago: A partir la parte órganos exclusivos del tubo digestivo. Nos limitaremos a citar las enfermedades que le afectan. Éstas serían acalasia, dolor torácico de origen esofágico, esófago de Barrett, esofagitis eosinofílica y varices esofágicas. En el diagnóstico de todas estas enfermedades es necesario realizar una endoscopia oral, bien sea para observar el estado del tejido, comprobar si hay tumores o realizar extracciones de tejido para examinar. También se puede sufrir la enfermedad por reflujo gástrico (ERGE), que se puede diagnosticar solamente con los síntomas y aplicar un tratamiento. En caso de que el tratamiento no fuera efectivo sí se realizaría endoscopia para comprobar si hay un problema mayor.

- Estómago: En el estómago las principales enfermedades son la úlcera péptica y la gastroparesia. Ambas requieren de endoscopias orales para el diagnóstico y además para la primera se suelen realizar también biopsias.

- Intestino delgado: Como hay múltiples enfermedades que afectan al intestino delgado, pero no todas requieren de endoscopias para su diagnóstico, solamente citaremos las que sí la necesiten. Serían la enfermedad celiaca, enfermedad de Crohn, enfermedad de Whipple e ileítis aguda. También hay que decir, que no siempre requerirán de endoscopia, pues algunas de estas enfermedades tienen métodos de diagnóstico alternativo según la gravedad con la que afecte al paciente. Por último, hay que destacar que dependiendo de la zona del intestino se quiera examinar se realizará una gastroscopia o una colonoscopia.

- Intestino grueso: Es el órgano que más tipos enfermedades sufre. Las principales son el cáncer colorrectal que a su vez tiene varios tipos, colitis que también existen varios tipos, enfermedad de Crohn perianal, enfermedad diverticular del colon, pólipos de colon, reservoritis, síndrome de Ogilvie,

síndrome del intestino irritable. En el diagnóstico de todas estas enfermedades se hace el uso de la colonoscopia y aunque existen métodos alternativos, la colonoscopia suele ser el más fiable.

2.3 Sistemas robóticos en endoscopia

Los robots se están empleando cada vez más para realizar exámenes laparoscópicos. Estas prácticas tienen los beneficios de que las pruebas causan un menor trauma en el paciente y reducen la fatiga en el médico [Muñoz 2013].

Diversos estudios han demostrado una mayor fiabilidad y seguridad en varios campos de la medicina que implican invasión en el cuerpo del paciente al utilizar sistemas robotizados. La alta precisión de los robots permite realizar cortes menores y en consecuencia se reduce el riesgo de hemorragia y el tiempo de recuperación. Además, dado que el robot se opera sentado la posición para el cirujano será mucho más cómoda reduciendo el cansancio y mejorando así las condiciones del cirujano a lo largo de las operaciones con el consecuente impacto positivo en el resultado [Muñoz 2013].

El primer robot que permitió realizar este tipo de operaciones fue el Da Vinci, Figura 2.9, creado en 2000. Este robot permite al cirujano manipularlo sentado mediante una consola que controla los brazos robóticos. Además, a través del endoscopio proporciona una visión tridimensional de alta definición [Muñoz 2013].

Figura 2.9 Robot Da Vinci

En los años posteriores han ido creándose nuevos robots con diferentes avances. Uno de los más importantes es el robot Zeus creado en 1998, pero que fue en 2003 cuando incorporó un sistema de control mediante comandos de voz llamado AESOP (Automated Endoscopic System for Optimal Positioning). Este robot es muy útil también para cirugías relacionadas con el corazón gracias a su alta precisión [Muñoz 2013].

Otro tipo de robots para realizar exámenes digestivos son cápsulas, Figura 2.10, que se ingieren por la cavidad oral y recorren el tubo digestivo tomando imágenes de todo el tracto digestivo. Una de las limitaciones es la dificultad de orientar la vista de modo que se observe con detalle el lugar deseado, ya que los movimientos naturales de esófago, estómago e intestinos complican la tarea. Se han comenzado a desarrollar formas de conseguir que el robot se adhiera a las paredes como ganchos o patas. Este robot sería capaz de realizar biopsias y de orientarse mejor para tomar imágenes en detalle de un lugar concreto.

Figura 2.10 Cápsula endoscópica y componentes

Por el momento no existen endoscopios robóticos. Sin embargo, sí que hay múltiples investigaciones sobre este tipo de sistemas, debido a que los endoscopios flexibles de fibra óptica, Figura 2.11, son los únicos que permiten detectar carcinomas. La evidente limitación de no poder dirigir el endoscopio flexible provoca que el riesgo de perforación de las paredes intestinales sea relativamente elevado. Por lo tanto, si se lograse combinar las ventajas de los endoscopios flexibles con el control y precisión de los robots obtendríamos una herramienta excelente para realizar colonoscopias. En este trabajo buscamos precisamente crear un robot para realizar esa tarea. Las investigaciones sobre este robot parten de que es un robot ápodo, modular e hiper-redundante.

Figura 2.11 Endoscopio flexible

2.4 Robots ápodos modulares hiper-redundantes

Debido a que en este trabajo se va a utilizar un robot ápodo modular e hiper-redundante. Expondremos brevemente algunos de los robots creados a lo largo de la historia y las características de este tipo de robots.

En primer lugar, explicaremos que significa el nombre. Los robots ápodos son aquellos que no disponen de extremidades como patas ni de elementos rotatorios como ruedas que utilice para desplazarse. La palabra "modular" indica que el robot está compuesto por segmentos llamados módulos. Estos módulos habrá unos actuadores que permitan el movimiento relativo entre los discos que los separan. La palabra "hiper-redundante" hace referencia a los grados de libertad e indica que para llegar a un punto tiene muchas configuraciones posibles. Esto es muy importante ya que le permite adaptarse al recorrido laberíntico que previsiblemente se va a encontrar en el intestino, así como a las irregularidades, obstáculos que puedan aparecer. Coloquialmente se les conoce como "robots serpiente" debido a las similitudes que tienen con éstas [Muñoz 2013].

Los primeros pasos en este tipo de robots los dio Shigeo Hiroshe cuando en 1972 creó el "robot serpiente" con el nombre de Mecanismo de Cuerdas Activas o ACM III, Figura 2.12. Este robot tenía un grado de libertad de revolución por cada módulo. En total tenía 20 grados de libertad y su movimiento imitaba al de las serpientes [Yttersad Pettersen & Transeth, 2006].

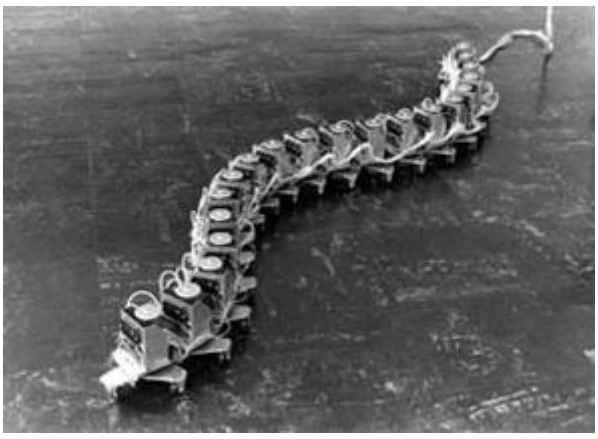


Figura 2.12 Robot ACM III

En 1995 el Jet Propulsion Laboratory de la NASA creó un robot llamado Serpentine Robot cuya finalidad era realizar inspecciones en la estación espacial por lo que eran lugares poco accesibles para los robots articulados por la dificultad de movimiento. Era un robot de 12 grados de libertad que tenía en cada de sus 5 articulaciones dos grados de libertad, ambos rotativos y los dos últimos en el extremo que permitían orientar la cámara [Paljug, E. 1995].

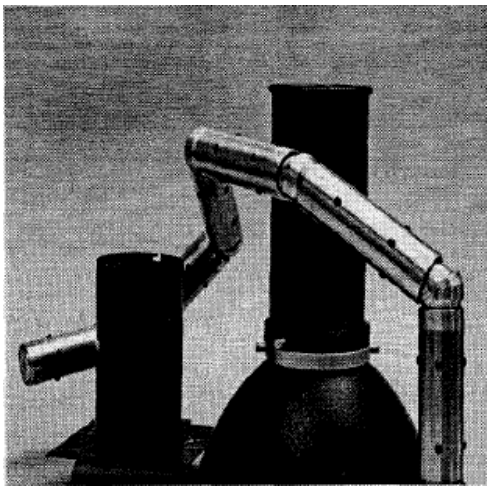


Figura 2.13 Serpentine Robot

En 2001 Hannan y Walker diseñaron un robot que imitaba a la trompa de un elefante, de donde recibe su nombre Elephant Trunk, Figura 2.14. Los movimientos los realizaba mediante un sistema de cables y resortes [Hannan, M. 2002].

Figura 2.14 Robot Elephant Trunk

Debido a que estos robots son ápodos los desplazamientos no los pueden realizar de las formas más habituales como son las patas o las ruedas accionadas por un motor. Por lo tanto, el desplazamiento debe ser deslizando o arrastrándose por el suelo mediante deformaciones del robot que hagan fuerzas que le impulsen. Para realizar los movimientos los investigadores se han inspirado en los que realizan los animales ápodos. Existen tres tipos de movimientos: peristáltico, ondulación lateral y arrastre con dos puntos.

El movimiento peristáltico consiste en la compresión y extensión de un segmento del cuerpo, de modo que se comprime un tramo comenzando por atrás y la

compresión va avanzando hacia adelante mientras se comienza a extender la parte trasera, Figura 2.15. Es un movimiento similar al de las lombrices.

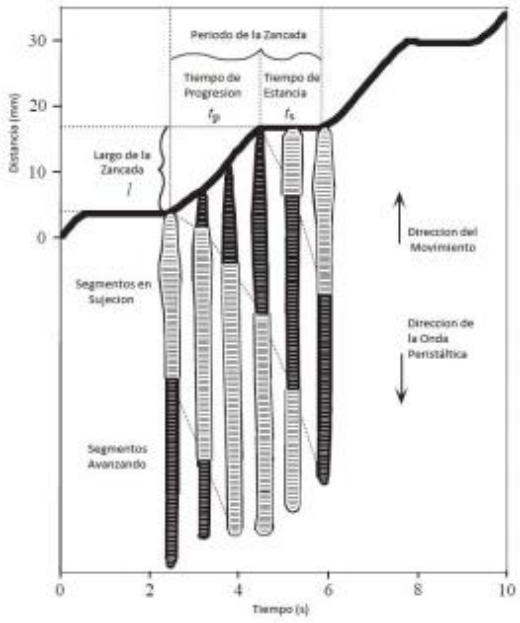


Figura 2.15 Movimiento peristáltico

El desplazamiento mediante ondulación lateral consiste en la deformación del cuerpo de forma senoidal de modo que las ondas se van desplazando hacia la parte trasera del cuerpo permitiendo el avance. Al contrario que en el movimiento peristáltico que las contracciones se asemejan a impulsos, en la ondulación las ondas son continuas y de la misma amplitud. La zancada corresponde a la longitud de onda. Este tipo de movimiento es el que utilizan la mayoría de las serpientes.

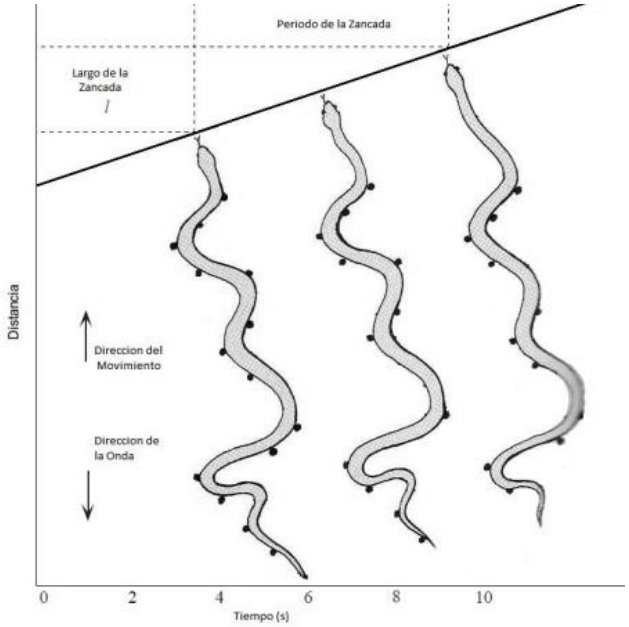


Figura 2.16: Movimiento de Ondulación Lateral - Se muestra un esquema del movimiento de ondulación lateral realizado por una serpiente, se señalan los puntos de apoyo

Por último, se encuentra el arrastre mediante dos apoyos. Este movimiento no es continuo como el ondulado, sino que es un ciclo que se repite. En primer lugar, se levanta el apoyo trasero, a continuación, se contrae el cuerpo y al final de la contracción se vuelve a apoyar la parte trasera. Una vez se apoya la parte trasera se levanta la delantera y se expande el cuerpo. La zancada depende de la capacidad del animal de contraer y extender su cuerpo. Es el utilizado por las sanguijuelas.

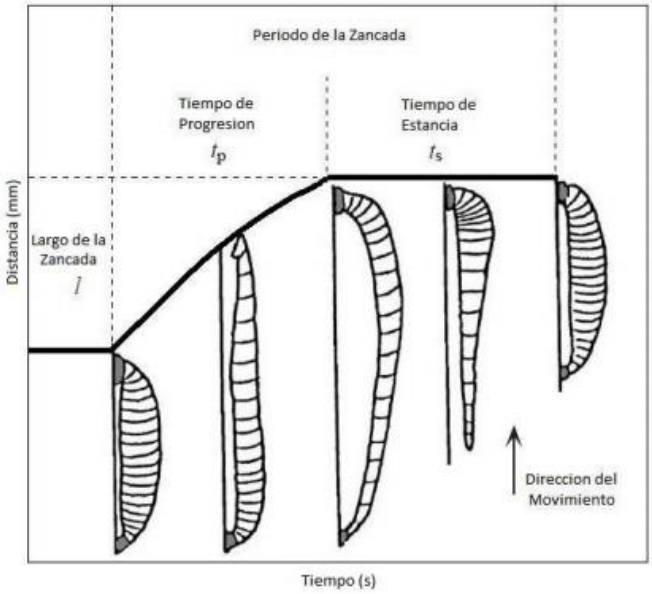


Figura 2.17: Arrastre Con Dos Puntos de Apoyo - Se muestra un esquema sencillo del movimiento de una sanguijuela, indicando además las variables cinemáticas independientes. Las zonas sombreadas representan los segmentos en sujeción

CAPÍTULO 3

3. Herramientas matemáticas para representar la cinemática

Para representar un sólido rígido en el espacio se necesita una herramienta que permita localizar sus puntos y su orientación. Para ello, se suelen emplear ejes de coordenadas cartesianas. Uno de los ejes llamado global (O, x, y, z) está fijo en el espacio y el otro se fija al sólido y se llama local (O_l, x_l, y_l, z_l). En la Figura 3.1 se muestra un ejemplo dónde el eje global aparece bajo la nomenclatura (O, X, Y, Z) y el eje local (O', U, V, W). En este trabajo se emplearán los cuaternios duales ya que cada cuaternio dual está formado por 8 componentes, la mitad de los que se necesitan empleando álgebra matricial lo que reducirá la carga computacional.

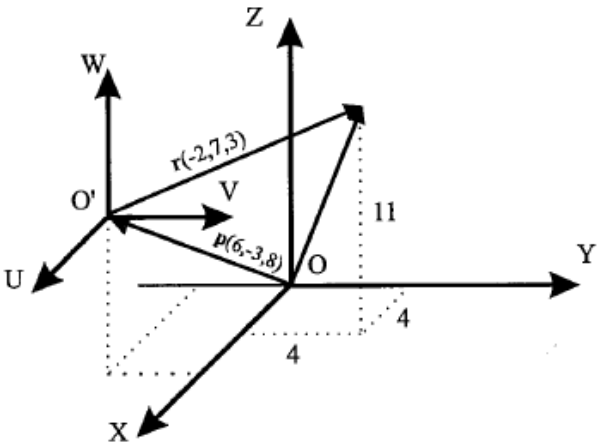


Figura 3.1 Representación de eje global (O, X, Y, Z) y eje local (O', U, V, W)

3.1 Matrices de transformación homogéneas

Las matrices de transformación homogénea son una herramienta que permiten la representación conjunta de la posición y la orientación de un sistema de referencia respecto a otro mediante coordenadas homogéneas. Las coordenadas homogéneas tienen una dimensión n+1 en la que habrá n elementos que corresponden a las coordenadas del vector y un elemento arbitrario que corresponde a un factor de escala y generalmente vale 1. Por lo tanto, un vector en coordenadas homogéneas tiene la siguiente forma p(w_x, w_y, w_z, w). Las matrices de transformación homogénea **T** tienen dimensión (4x4) y está compuesta por 4 submatrices: rotación, traslación, perspectiva y escalado [Barrientos, 1997].

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Persepectiva & Escalado \end{bmatrix} \qquad (\boldsymbol{Ec.3.1})$$

En robótica se considera perspectiva **f** nula y escalado **w** unidad por lo que solamente interesa conocer la matriz **R** de rotación y el vector **p** de traslación. De esta manera quedaría la matriz **T** de la siguiente forma.

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ 0 & 1 \end{bmatrix} \qquad (\boldsymbol{Ec.3.2})$$

La matriz de rotación **R** es una matriz ortonormal 3x3 que representa la orientación del sistema de referencia local O'UVW en el sistema global OXYZ. La primera columna corresponde a la representación del eje U en el sistema de referencia global. La segunda corresponde al eje V y la tercera al eje W. Para obtener esta matriz se tiene en cuenta alrededor de que ejes del sistema OXYZ se ha de girar para alcanzar la orientación del sistema OUVW. Para girar alrededor de cada eje se creará una matriz diferente de rotación Ec. 3.3, Ec. 3.4, Ec. 3.5.

$$R(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sen\alpha \\ 0 & sen\alpha & cos\alpha \end{bmatrix} \qquad (\boldsymbol{Ec.3.3})$$

$$R(y,\phi) = \begin{bmatrix} cos\phi & 0 & sen\phi \\ 0 & 1 & 0 \\ -sen\phi & 0 & cos\phi \end{bmatrix} \qquad (\boldsymbol{Ec.3.4})$$

$$R(z,\theta) = \begin{bmatrix} cos\theta & -sen\theta & 0 \\ sen\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (\boldsymbol{Ec.3.5})$$

Para obtener la matriz de rotación final se multiplican las matrices R(x,α), R(y,ϕ) y R(z,θ) teniendo en cuenta el orden que se ha seguido pues el producto de matrices no es conmutativo.

El vector de traslación **p** es el vector del punto O' (origen del sistema de coordenadas local) respecto al sistema de referencia global OXYZ.

Estas matrices sirven para lo siguiente:

1) Representar la orientación y posición de un sistema girado y trasladado O'UVW con respecto al sistema de referencia OXYZ.

2) Transformar un vector expresado en coordenadas con respecto a un sistema O'UVW, a su expresión en coordenadas del sistema de referencia OXYZ.

3) Rotar y trasladar un vector con respecto a un sistema de referencia fijo OXYZ [Barrientos,1997].

3.2 Cuaternios y cuaternios duales

3.2.1 Cuaternios

Los cuaternios son una herramienta matemática que se pueden emplear para trabajar con giros y orientaciones. Un cuaternio Q está compuesto por cuatro componentes (q_0, q_1, q_2, q_3). La primera componente es la parte

escalar y las otras tres son las coordenadas (i, j, k) del vector. Se puede representar como [Barrientos,1997]:

$$Q = [q_0, q_1, q_2, q_3] = [s, v] \qquad (\boldsymbol{Ec.3.6})$$

$$q = w + xi + yj + zk \qquad (\boldsymbol{Ec.3.7})$$

donde s es la parte escalar y v la parte vectorial. Para la utilización de los cuaternios como metodología de representación de orientaciones se asocia el giro de un ángulo θ sobre el vector k al cuaternio definido por [Barrientos,1997]:

$$Q = \boldsymbol{Rot}(\boldsymbol{k}, \theta) = \left(\cos\frac{\theta}{2}, \boldsymbol{k}\sin\frac{\theta}{2}\right) \qquad (\boldsymbol{Ec.3.8})$$

Al realizar operaciones se debe tener en cuenta las siguientes propiedades:

$$i^2 = j^2 = k^2 = ijk = -1 \qquad (\boldsymbol{Ec.3.9})$$
$$ij = k = -ji \qquad (\boldsymbol{Ec.3.10})$$
$$jk = i = -kj \qquad (\boldsymbol{Ec.3.11})$$
$$ki = j = -ik \qquad (\boldsymbol{Ec.3.12})$$

Para expresar las operaciones básicas se denotará por comodidad como un par compuesto por un escalar y un vector 3D:

$$q = [w, \vec{v}], con\ \vec{v} = (x, y, z) \qquad (\boldsymbol{Ec.3.13})$$

<u>Operaciones básicas</u>

Suma:

$$q + q' = [w + w', \vec{v} + \vec{v}'] = [w + w', (x + x', y + y', z + z')] \qquad (\boldsymbol{Ec.3.14})$$

Producto:

$$qq' = [ww' - \vec{v} \cdot \vec{v}', \vec{v} \times \vec{v}' + w\vec{v}' + w'\vec{v}] \neq q'q \qquad (\boldsymbol{Ec.3.15})$$

Módulo:

$$|q| = \sqrt{w^2 + x^2 + y^2 + z^2} \qquad (\boldsymbol{Ec.3.16})$$

Conjugado:

$$q^* = [w, -\vec{v}] \qquad (\boldsymbol{Ec.3.17})$$

Inversa:

$$q^{-1} = \frac{q^*}{|q|^2} \qquad (\boldsymbol{Ec.3.18})$$

Utilización de cuaternios

Para realizar rotaciones utilizando cuaternios se debe crear el cuaternio del punto que se quiere rotar (Ec. 3.19) y el cuaternio de rotación definido por el ángulo θ que se quiere girar y el eje de giro $\vec{n}$, como se definió previamente en (Ec. 3.8). Para obtener el punto rotado P' se debe realizar la operación de (Ec. 3.21), dónde P' es el punto rotado, P es el punto al inicio, Q es el cuaternio de rotación y Q^* es el conjugado del cuaternio de rotación Q.

$$P = \left(0, v_x i + v_y j + v_z k\right) \qquad (\boldsymbol{Ec.3.19})$$

$$Q = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2}(n_x i, n_y j, n_z k\right) \qquad (\boldsymbol{Ec.3.20})$$

$$P' = Q \cdot P \cdot Q^* \qquad (\boldsymbol{Ec.3.21})$$

A continuación, se muestra un ejemplo de rotación empleando cuaternios. En la Figura 3.2 se muestra gráficamente la operación. Se parte de un punto A=(1, 0, 1) y se rota un ángulo θ=180º respecto al eje $\vec{n}$=(-1, 0, 0) para alcanzar el punto A'.

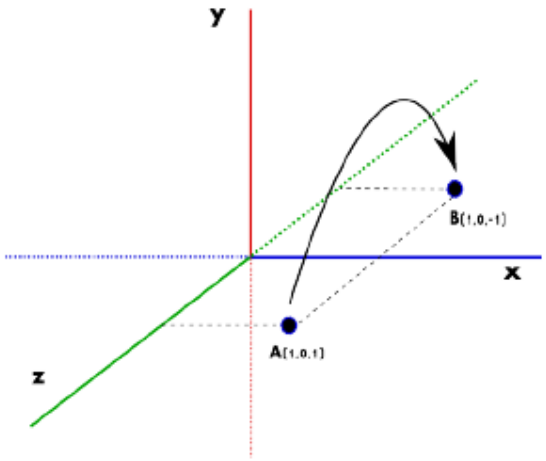


Figura 3.2 Ejemplo de rotación de un punto mediante cuaternios

En primer lugar, creamos el cuaternio del punto *A* (Ec. 3.22). A continuación, calculamos el cuaternio de rotación $q_{x,180º}$ (Ec. 2.23) y su conjugado $q^*_{x,180º}$ (Ec. 3.24) mediante el vector del eje de giro *n* y el ángulo de giro θ.

$$q_A = (0 + \vec{r}) = \left(0 + \vec{\imath} + \vec{k}\right) \qquad (\boldsymbol{Ec.3.22})$$

$$q_{x,180^\circ} = [w, \vec{v}] = \left(\cos\frac{\theta}{2}, \sin\frac{\theta}{2} \cdot \vec{n}\right) = (0 - \vec{\imath}) \qquad (\boldsymbol{Ec.\,3.23})$$

$$q^*_{x,180^\circ} = [w, -\vec{v}] = (0 + \vec{\imath}) \qquad (\boldsymbol{Ec.\,3.24})$$

Una vez tenemos los cuaternios calculamos el cuaternio del punto *A'* recordando como es el producto de dos cuaternios (Ec. 3.15).

$$q_{A'} = q_{x,180^\circ} \cdot q_A \cdot q^*_{x,180^\circ} = (0 - \vec{\imath}) \cdot \left(0 + \vec{\imath} + \vec{k}\right) \cdot (0 + \vec{\imath}) \qquad (\boldsymbol{Ec.\,3.25})$$

$$q_{A'} = (0 + \vec{r}') = (0 + 1 + \vec{\jmath} + 0 + 0) \cdot (0 + \vec{\imath}) \qquad (\boldsymbol{Ec.\,3.26})$$

$$q_{A'} = (0 + \vec{r}') = \left(0 + 0 - \vec{k} + \vec{\imath} + 0\right) = \left(0 + \vec{\imath} - \vec{k}\right) \qquad (\boldsymbol{Ec.\,3.27})$$

$$q_{A'} = (0 + \vec{r}') = \left(0 + \vec{\imath} - \vec{k}\right) \;\rightarrow\; A' = r' = (1, 0, -1) \qquad (\boldsymbol{Ec.\,3.28})$$

Obtenemos el cuaternio $q_{A'}$ del que extraemos el vector *r'* que corresponde al punto A' (Ec. 3.28).

También se pueden realizar varios giros. Para obtener el cuaternio de giro se debe realizar la composición a partir de los cuaternios de cada giro por separado. En caso de que la rotación sea respecto un sistema de referencia fijo se premultiplica (Ec. 3.29). En caso de que la rotación se haga respecto de un sistema de referencia transformado se postmultiplica (Ec. 3.30).

$$q = q_n \cdot q_{n-1} \cdot \ldots \cdot q_2 \cdot q_1 \qquad (\boldsymbol{Ec.\,3.29})$$

$$q = q_1 \cdot q_2 \cdot \ldots \cdot q_{n-1} \cdot q_n \qquad (\boldsymbol{Ec.\,3.30})$$

3.2.2 Cuaternios duales

Los cuaternios son útiles para realizar rotaciones, pero no permiten realizar traslaciones. Para realizar traslaciones o combinar rotaciones con traslaciones se emplean los cuaternios duales. Los cuaternios duales pertenecen al espacio 8D y están formados por dos cuaternios que en la ecuación 3.31 son *a* y *b*. La componente ε es la componente de dualidad y debe cumplirse que $\varepsilon^2 = 0$.

$$\hat{q} = a + b\varepsilon \qquad (\boldsymbol{Ec.\,3.31})$$

Operaciones básicas

Suma:

$$\hat{q} + \hat{q}' = a + a' + (b + b')\varepsilon \qquad (\boldsymbol{Ec.3.32})$$

Producto:

$$\hat{q}\hat{q}' = aa' + (ab' + a'b)\varepsilon \qquad (\boldsymbol{Ec.3.33})$$

Módulo:

$$|\hat{q}| = \sqrt{\hat{q}\hat{q}^*} = \sqrt{\hat{q}^*\hat{q}} = |a| + \frac{a \cdot b}{|a|}\varepsilon \qquad (\boldsymbol{Ec.3.34})$$

Conjugados:

$$\hat{q}^* = a^* + b^*\varepsilon \qquad (\boldsymbol{Ec.3.35})$$
$$\overline{\hat{q}^*} = \overline{a^* + b^*\varepsilon} = a^* - b^*\varepsilon \qquad (\boldsymbol{Ec.3.36})$$

Inversa:

$$\hat{q}^{-1} = \frac{\hat{q}^*}{|\hat{q}|^2}, \forall a \neq 0 \qquad (\boldsymbol{Ec.3.37})$$

Cuaternio unitario:

$$\widehat{q_u}, |a| = 1, a \cdot b = 0 \qquad (\boldsymbol{Ec.3.38})$$

Aplicación al movimiento rígido

Para realizar un movimiento definido por una rotación y un desplazamiento se deben crear el cuaternio dual Â del punto A(A_x, A_y, A_z) que sufre la transformación (Ec. 3.39), el cuaternio de rotación $\hat{r}$ a partir del ángulo de rotación θ y el vector eje de giro $\vec{n}$ (Ec. 3.40), el cuaternio de desplazamiento $\hat{d}$ a partir del vector del desplazamiento $\vec{d}$ (Ec. 3.41) y por último el cuaternio dual compuesto se obtendrá premultiplicando $\hat{r}$ y $\hat{d}$ según el orden de las operaciones (Ec. 3.42) y (Ec. 3.43).

$$\hat{A} = 1 + \varepsilon\left(A_x\vec{\imath} + A_y\vec{\jmath} + A_z\vec{k}\right) \qquad (\boldsymbol{Ec.3.39})$$

$$\hat{r} = \cos\left(\frac{\theta}{2}\right) + \vec{n}\sin\left(\frac{\theta}{2}\right) \qquad (\boldsymbol{Ec.3.40})$$

$$\hat{d} = 1 + \varepsilon\frac{\vec{d}}{2} \qquad (\boldsymbol{Ec.3.41})$$

$$\hat{q} = \hat{r} * \hat{d} \;\; (primero\ desplazamiento, luego\ rotación) \qquad (\boldsymbol{Ec.3.42})$$

$$\hat{q} = \hat{d} * \hat{r} \ \ (primero\ rotación, luego\ desplazamiento) \quad (\boldsymbol{Ec.3.43})$$

En caso de que se realicen varias rotaciones se seguirá el mismo procedimiento premultiplicando los cuaternios duales para obtener el cuaternio dual compuesto.

La operación final para obtener el punto A transformado al igual que en los cuaternios consiste en (Ec. 3.44).

$$\hat{A}' = \hat{q}\hat{A}\overline{\hat{q}^*} \quad (\boldsymbol{Ec.3.44})$$

A continuación, se muestra un ejemplo de cada tipo de caso simple: traslación pura, rotación seguida de traslación y traslación seguida de rotación.

<u>-Traslación pura</u>

Se traslada el punto A(1, 0, 1) un vector d(1, 0, 1) respecto al sistema de referencia fijo como se muestra gráficamente en la Figura 3.3. En primer lugar, se crean los cuaternios duales necesarios $\hat{A}$ (Ec. 3.45), $\hat{d}$ (Ec. 3.46), $\hat{r}$ (Ec. 3.47) (al ser traslación pura el ángulo rotado θ será 0 y el eje de rotación $\vec{n}$ = i por comodidad), el cuaternio dual de la transformación $\hat{q}$ (Ec. 3.48) y su conjugado $\overline{\hat{q}^*}$ (Ec. 3.49).

$$A(1,0,1) \Rightarrow \hat{A} = 1 + \vec{\imath}\varepsilon + \vec{k}\varepsilon, \text{el punto rotado} \quad (\boldsymbol{Ec.3.45})$$

$$d(1,0,1) \Rightarrow \hat{d} = 1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon, \text{el desplazamiento} \quad (\boldsymbol{Ec.3.46})$$

$$\hat{r} = \cos\frac{0}{2} + \vec{n}\sin\frac{0}{2} = 1, \text{cuaternio rotación} \quad (\boldsymbol{Ec.3.47})$$

$$\hat{q} = \hat{d}\cdot\hat{r} = \left(1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon\right)\cdot(1) = 1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon \quad (\boldsymbol{Ec.3.48})$$

$$\overline{\hat{q}^*} = a^* - b^*\varepsilon = 1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon \quad (\boldsymbol{Ec.3.49})$$

Una vez tenemos los cuaternios duales necesarios ya podemos realizar la transformación y obtener el punto A'.

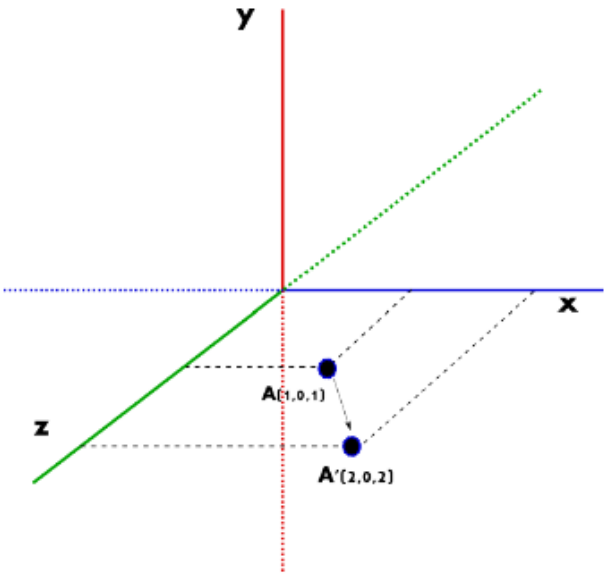


Figura 3.3 Ejemplo de traslación de un punto mediante cuaternios duales

$$\hat{A}' = \hat{q}\hat{A}\overline{\hat{q}^*} = \left(1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon\right) \cdot \left(1 + \vec{\imath}\varepsilon + \vec{k}\varepsilon\right) \cdot \left(1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon\right) \quad (\boldsymbol{Ec.3.50})$$

$$\hat{A}' = \left(1 + \frac{3}{2}\vec{\imath}\varepsilon + \frac{3}{2}\vec{k}\varepsilon\right) \cdot \left(1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon\right) \quad (\boldsymbol{Ec.3.51})$$

$$\hat{A}' = 1 + 2\vec{\imath}\varepsilon + 2\vec{k}\varepsilon \quad (\boldsymbol{Ec.3.52})$$

$$\hat{A}' = 1 + 2\vec{\imath}\varepsilon + 2\vec{k}\varepsilon \Rightarrow A' = (2,0,2) \quad (\boldsymbol{Ec.3.53})$$

Teniendo en cuenta (Ec. 3.39) obtenemos el punto A' en (Ec. 3.53).

-Rotación seguida de desplazamiento

El punto a transformar es A(1, 0, 1), el desplazamiento es d(1, 0, 1) y la rotación tiene los parámetros θ = π y $\vec{n} = (1,0,1)$. Se muestra gráficamente en la Figura 3.4. Por lo tanto, los cuaternios duales $\hat{A}$ y $\hat{d}$ son los mismos que se calcularon en (Ec. 3.45) y (Ec. 3.46) y solo tenemos que calcular el nuevo cuaternio dual de rotación $\hat{r}$ (Ec. 3.54).

$$\hat{r} = \cos\frac{\pi}{2} + \vec{n}\sin\frac{\pi}{2} = \vec{\imath} \quad (\boldsymbol{Ec.3.54})$$

Dado que primero se realiza la rotación y después el desplazamiento el nuevo cuaternio dual de transformación es (Ec. 3.55).

$$\hat{q} = \hat{d} \cdot \hat{r} = \left(1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon\right) \cdot \vec{\imath} = \vec{\imath} - \frac{1}{2}\varepsilon + \frac{1}{2}\vec{\jmath}\varepsilon \quad (\boldsymbol{Ec.3.55})$$

$$\overline{\hat{q}^*} = -\vec{\imath} + \frac{1}{2}\varepsilon + \frac{1}{2}\vec{\jmath}\varepsilon \qquad (\boldsymbol{Ec.3.56})$$

Una vez tenemos el cuaternio dual de transformación y su conjugado (Ec. 3.56) calculamos el punto transformado B (Ec. 3.59).

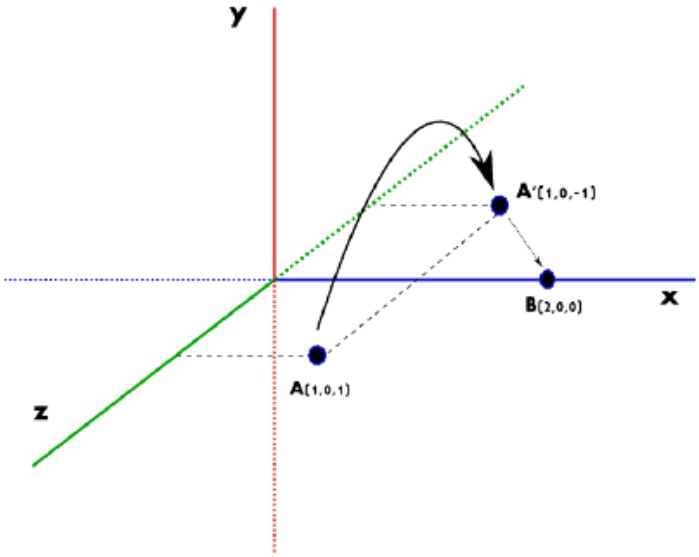


Figura 3.4 Ejemplo de rotación seguida de traslación de un punto mediante cuaternios duales

$$\hat{B} = \hat{q}\hat{A}\overline{\hat{q}^*} = \left(\vec{\imath} - \frac{1}{2}\varepsilon + \frac{1}{2}\vec{\jmath}\varepsilon\right)\cdot\left(1 + \vec{\imath}\varepsilon + \vec{k}\varepsilon\right)\cdot\left(-\vec{\imath} + \frac{1}{2}\varepsilon + \frac{1}{2}\vec{\jmath}\varepsilon\right) \qquad (\boldsymbol{Ec.3.57})$$

$$\hat{B} = \left(\vec{\imath} - \frac{3}{2}\varepsilon - \frac{1}{2}\vec{\jmath}\varepsilon\right)\cdot\left(-\vec{\imath} + \frac{1}{2}\varepsilon + \frac{1}{2}\vec{\jmath}\varepsilon\right) \qquad (\boldsymbol{Ec.3.58})$$

$$\hat{B} = 1 + 2\vec{\imath}\varepsilon \Rightarrow B = (2,0,0) \qquad (\boldsymbol{Ec.3.59})$$

-Traslación seguida de rotación

Por último, se va a realizar una transformación en la que en primer lugar se realizará la traslación y le seguirá una rotación. Se muestra gráficamente la transformación en la Figura 3.5. El punto a rotar es A(1, 0, 1), el desplazamiento es d(1, 0, 1) y la rotación es de θ=π respecto al eje $\vec{n} = (1,0,0)$. Puesto que no varían los parámetros respecto al ejemplo anterior solamente debemos calcular el nuevo cuaternio dual compuesto (Ec. 3.60) y su conjugado (Ec. 3.61).

$$\hat{q} = \hat{r}\cdot\hat{d} = \vec{\imath}\cdot\left(1 + \frac{1}{2}\vec{\imath}\varepsilon + \frac{1}{2}\vec{k}\varepsilon\right) = \vec{\imath} - \frac{1}{2}\varepsilon - \frac{1}{2}\vec{\jmath}\varepsilon \qquad (\boldsymbol{Ec.3.60})$$

$$\overline{\hat{q}^*} = -\vec{\imath} - \frac{1}{2}\varepsilon - \frac{1}{2}\vec{\jmath}\varepsilon \qquad (\boldsymbol{Ec.3.61})$$

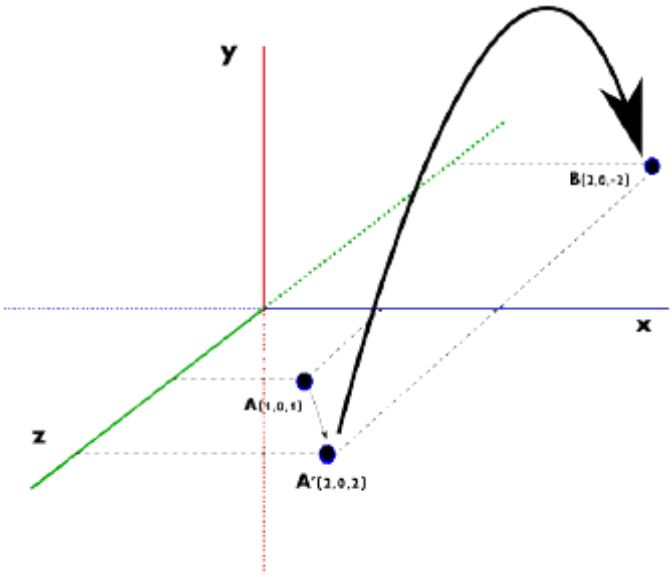


Figura 3.5 Ejemplo de traslación seguida de rotación de un punto mediante cuaternios duales

Una vez tenemos el cuaternio dual de transformación $\hat{q}$ y su conjugado $\overline{\hat{q}^*}$ podemos obtener el punto B (Ec. 3.63).

$$\hat{B} = \hat{q}\hat{A}\overline{\hat{q}^*} = \left(\vec{\imath} - \frac{1}{2}\varepsilon - \frac{1}{2}\vec{\jmath}\varepsilon\right) \cdot \left(1 + \vec{\imath}\varepsilon + \vec{k}\varepsilon\right) \cdot \left(-\vec{\imath} - \frac{1}{2}\varepsilon - \frac{1}{2}\vec{\jmath}\varepsilon\right) \quad (\boldsymbol{Ec.\,3.62})$$

$$\hat{B} = \left(\vec{\imath} - \frac{3}{2}\varepsilon - \frac{3}{2}\vec{\jmath}\varepsilon\right) \cdot \left(-\vec{\imath} - \frac{1}{2}\varepsilon - \frac{1}{2}\vec{\jmath}\varepsilon\right) \quad (\boldsymbol{Ec.\,3.63})$$

$$\hat{B} = 1 + 2\vec{\imath}\varepsilon - 2\vec{k}\varepsilon \Rightarrow B = (2, 0, -2) \quad (\boldsymbol{Ec.\,3.63})$$

CAPÍTULO 4

4. Descripción del robot

El robot diseñado será un robot ápodo, modular e híper-redundante, término explicado previamente en el Apartado 2.4. Este trabajo ha sido previamente desarrollado por antiguos alumnos de la Universidad de Valladolid [Hontiyuelo, E. 2018], [Álvarez, J. 2019].

4.1 Estudio del diseño

El robot estará compuesto por una serie de módulos con forma de disco que estarán acoplados de tal forma que el robot tenga forma de serpiente Figura 4.1. Los actuadores se encuentran en la parte superior de cada módulo.

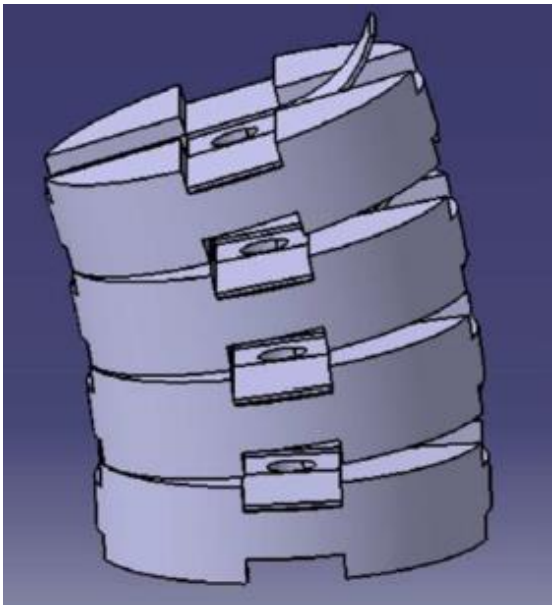


Figura 4.1 Diseño del robot ápodo modular e híper-redundante con 4 módulos [Hontiyuelo, E. 2018]

Como se puede observar en la Figura 4.1 en el centro de cada módulo existe un agujero cilíndrico a través del cual se introducirán los cables necesarios para alimentar los actuadores y para la cámara y utensilios que el robot pueda llevar en el extremo.

Los actuadores están formados por cuatro pletinas cada una sujeta a un hilo de nitinol que es un material biocompatible. Los extremos de cada pletina se conectan al siguiente módulo mediante pares prismáticos que permiten el desplazamiento relativo entre el actuador y el siguiente módulo Figura 4.2.

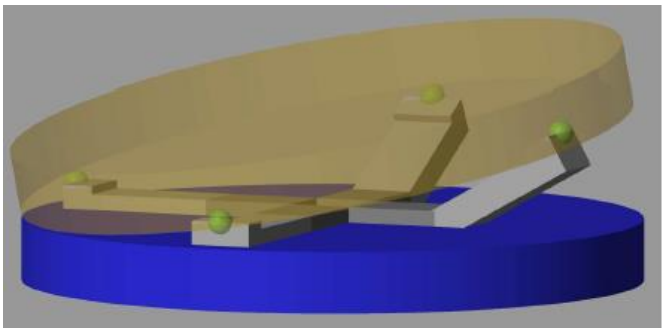


Figura 4.2 Modelo básico de actuador [Cabezas, G. 2019]

Los actuadores funcionan mediante el paso de corriente a través de los hilos de nitinol. El paso de corriente produce un cambio de temperatura en el hilo que produce la dilatación o la contracción según si está pasando corriente o no. Este cambio de longitud provoca la elevación de la pletina, logrando así el cambio de posición relativa entre módulos. En la Figura 4.3 se muestra un prototipo de una pletina de actuador.

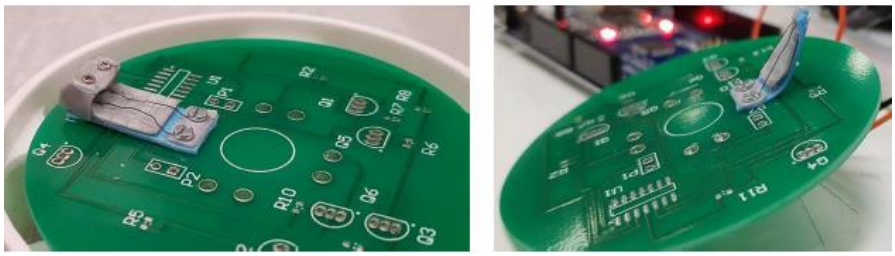


Figura 4.3 Pletina de actuador de hilos de nitinol [Cabezas, G. 2019]

El robot estará definido por una serie de parámetros, Figuras 4.4 y 4.5:

- d_p: Diámetro de los módulos. El diámetro de cada módulo debe ser reducido (menos de 15 mm) para que pueda realizar el recorrido a través del intestino sin riesgo de que roce con las paredes (Figura 4.4).

- h_0: Altura mínima de los módulos excluyendo las pletinas (Figura 4.5).

- h: Carrera de los actuadores. Es distancia entre la superficie del módulo sobre la que se encuentran las pletinas y el punto más elevado de la pletina cuando se encuentra activada (Figura 4.5).

- h_1: altura máxima del módulo, $h_1=h+h_0$ (Figura 4.5).

- L_1: Longitud de la parte móvil del actuador (Figura 4.4).

- c: Distancia del centro del cilindro a la parte móvil de las pletinas (Figura 4.4).

- L_2: Altura de la pletina (Figura 4.4).

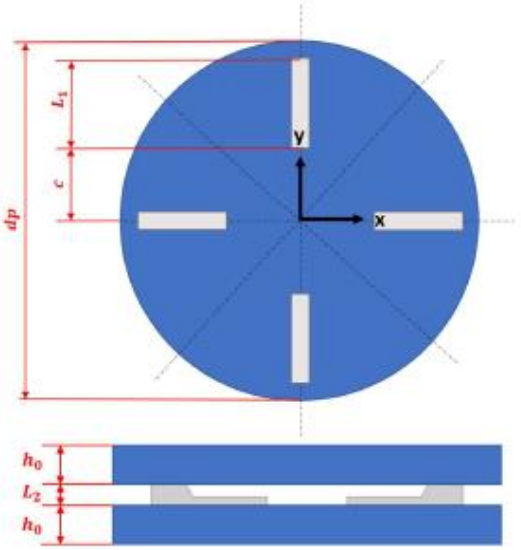


Figura 4.4 Parámetros de los módulos (Primera imagen) [Cabezas, G. 2019]

Figura 4.5 Parámetros de los módulos (Segunda imagen)

4.2 Configuraciones de los actuadores

En función de que pletinas se activen se obtendrá una configuración u otra. Sin embargo, no todas las configuraciones son válidas. A continuación, se explica el sistema empleado para designar las configuraciones y cuáles se pueden emplear.

En primer lugar, a cada pletina se le asigna un número como se muestra en la Figura 4.6. Esta numeración se asigna para comprender con mayor facilidad la Tabla 4.1, dónde se muestran todas las configuraciones posibles. A cada configuración se le asigna un número en código binario según las pletinas que estén activas.

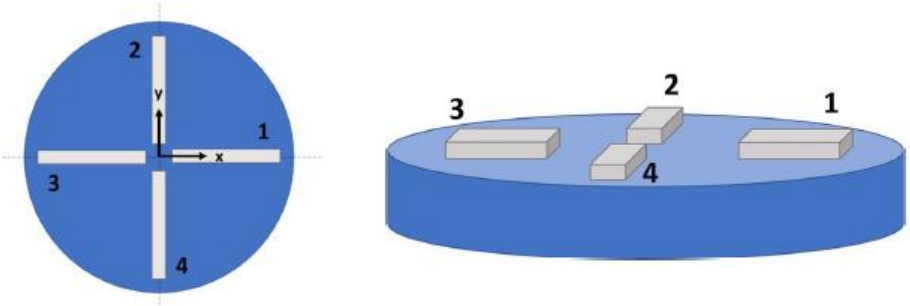


Figura 4.6 Numeración de los actuadores de un módulo [Cabezas, G. 2019]

Código decimal	Código binario	Actuador 4	Actuador 3	Actuador 2	Actuador 1
0	0000	0	0	0	0
1	0001	0	0	0	1
2	0010	0	0	1	0
3	0011	0	0	1	1
4	0100	0	1	0	0
5	0101	0	1	0	1
6	0110	0	1	1	0
7	0111	0	1	1	1
8	1000	1	0	0	0
9	1001	1	0	0	1
10	1010	1	0	1	0
11	1011	1	0	1	1
12	1100	1	1	0	0
13	1101	1	1	0	1
14	1110	1	1	1	0
15	1111	1	1	1	1

Tabla 4.1 Configuraciones posibles de los actuadores

Como se observa en la tabla existen 16 configuraciones posibles, sin embargo, no todas se pueden emplear. Por motivos de estabilidad no se pueden emplear las configuraciones en las que solamente hay un actuador activo, por lo que las configuraciones 1, 2, 4 y 8 no son útiles. Las configuraciones en las que hay tres actuadores activos o hay dos actuadores activos, pero están enfrentados (esto es actuadores 1 y 3 ó actuadores 2 y 4 activos), tampoco se pueden emplear al presentar problemas de estabilidad, además de no permitir girar correctamente los módulos entre sí. Por lo tanto, las configuraciones 5, 7, 10, 11, 13 y 14 también están descartadas. Esto nos deja con 6 configuraciones útiles que son las configuraciones 0, 3, 6, 9, 12 y 15.

Las configuraciones en las que los cuatro actuadores se encuentran en el mismo estado realizan un movimiento relativo de traslación entre los módulos. La configuración 0 es la posición de reposo del actuador. En la configuración 15 al estar los cuatro actuadores activos la distancia entre los módulos es mayor que en la configuración 0, lo que modifica la longitud del robot. En los casos útiles en los que hay dos actuadores activos se realiza un giro relativo entre módulos. La Tabla 4.2 recoge las configuraciones a modo de resumen y las Figuras 4.7 y 4.8 muestran las posiciones de traslación y rotación respectivamente.

Código decimal	Código binario	Actuador 4	Actuador 3	Actuador 2	Actuador 1	Movimiento
0	0000	0	0	0	0	Traslación
3	0011	0	0	1	1	Rotación
6	0110	0	1	1	0	Rotación
9	1001	1	0	0	1	Rotación
12	1100	1	1	0	0	Rotación
15	1111	1	1	1	1	Traslación

Tabla 4.2 Configuraciones útiles de los actuadores

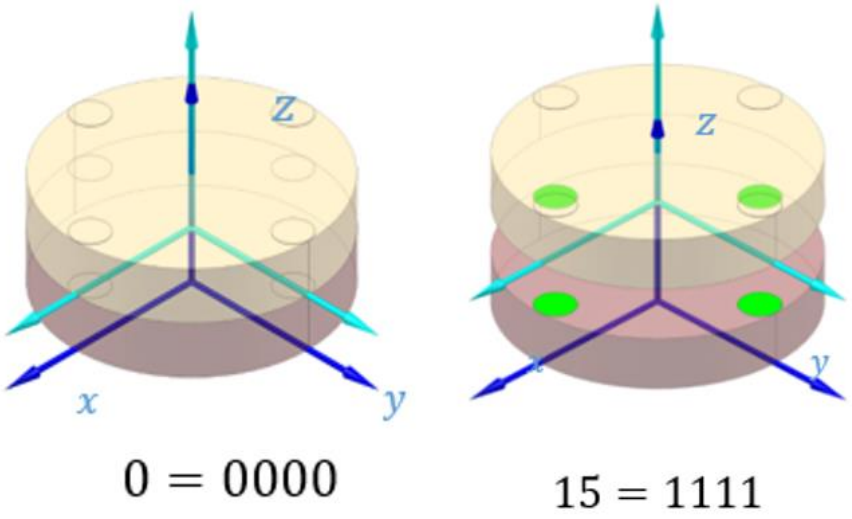


Figura 4.7 Configuraciones de traslación de los módulos

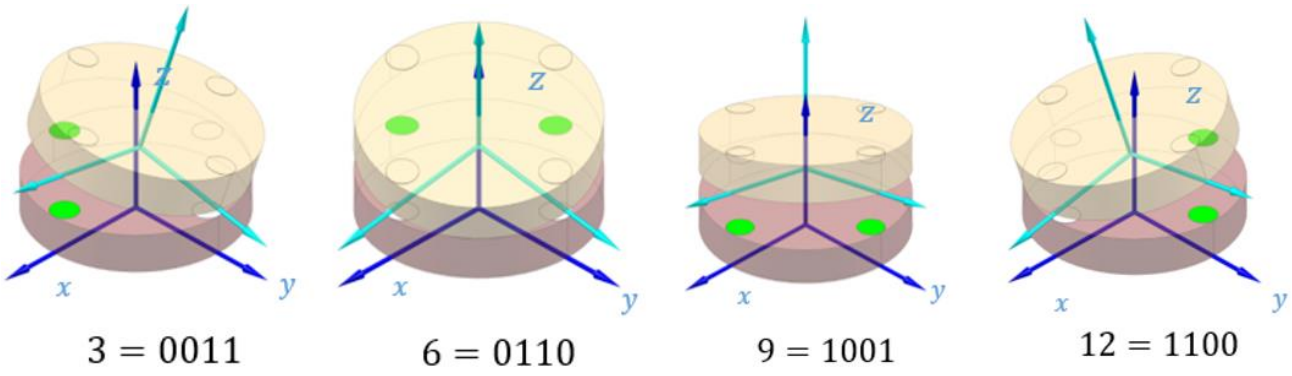


Figura 4.8 Configuraciones de rotación de los módulos

El ángulo formado entre las bases de dos módulos se denominará ***α***. Este ángulo viene determinado por la relación entre *h* y una variable llamada *L*. ***L*** es la distancia entre el punto del módulo superior sobre el que se mide *h* y el punto más bajo del módulo superior. En la Figura 4.9 se muestra la representación gráfica de las variables *h* y *L*.

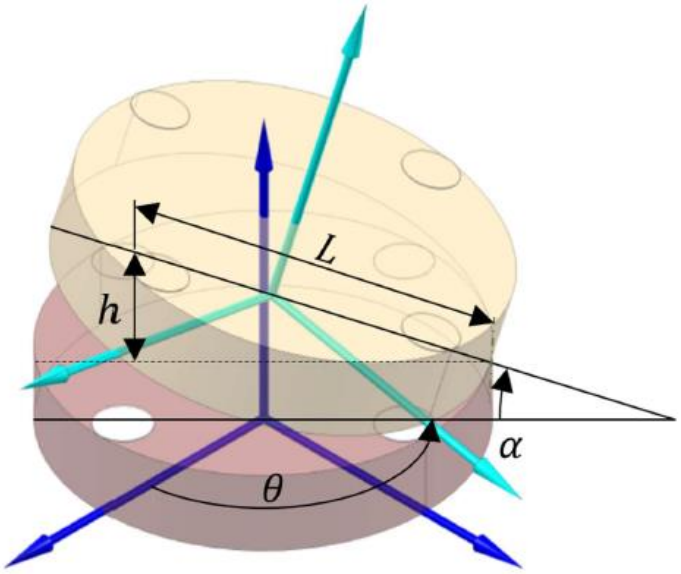


Figura 4.9 Representación gráfica del ángulo α [Hontiyuelo, E. 2018]

Una vez definidos los parámetros se puede calcular el ángulo α que se obtendría mediante la expresión Ec. 4.1.

$$\alpha = \sin^{-1}\left(\frac{h}{L}\right) \qquad (\boldsymbol{Ec.\,4.1})$$

El ángulo α se utilizará más adelante para establecer las limitaciones de giro al robot en las simulaciones.

CAPÍTULO 5

5. Algoritmos para la representación de la cinemática inversa

En robótica existen dos problemas fundamentales a resolver en la cinemática del robot. El primero de ellos se conoce como el problema cinemático directo, y consiste en determinar cuál es la posición y orientación del extremo final de robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot. El segundo problema, denominado problema cinemático inverso, resuelve la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas [Barrientos, A. 1997].

En nuestro caso resolveremos el problema de cinemática inversa, debido que queremos conocer la configuración que debe tomar el robot para alcanzar los diferentes puntos de la trayectoria.

5.1 Algoritmos de cinemática inversa

Resolver la cinemática inversa nos plantea un problema, ya que para un robot con n módulos, teniendo 6 configuraciones diferentes cada módulo supondría que existen 6^n configuraciones diferentes del robot. Por lo tanto, un robot de 5 módulos dispondría de 7776 configuraciones diferentes. Evidentemente calcular todas las configuraciones tendría una carga computacional y un coste de tiempo muy elevado. Por este motivo se ha decidido emplear algoritmos iterativos para resolver la cinemática inversa. En nuestro caso interesa un algoritmo que tenga un tiempo de cálculo breve, ya que el cálculo de la cinemática inversa tendrá lugar en tiempo real, es decir, a medida que robot avanza se van calculando las nuevas posiciones de los módulos. A continuación, se describirán los algoritmos seleccionados y las ventajas y desventajas de cada uno de ellos.

5.1.1 Algoritmo Forward And Backward Reaching Inverse Kinematics (FABRIK)

El algoritmo llamado Forward And Backward Reaching Inverse Kinematics (FABRIK) es un método heurístico de cálculo de cinemática inversa. Este método no utiliza ángulos ni matrices de rotación si no que calcula la posición de cada punto conforme a las restricciones de tamaño y movimiento de cada módulo. Es común utilizarlo en aplicaciones que requieran de cálculo en tiempo real como por ejemplo los videojuegos. Esto se debe a que por la simplicidad de sus cálculos la carga computacional y el tiempo requerido son pequeños. [Aristidou, A. 2011]

Para el cálculo de las posiciones se necesita la posición inicial de los módulos, el punto en el que se encuentra la base, el punto objetivo, la longitud de cada uno de los módulos, la tolerancia y el número máximo de iteraciones. El algoritmo es iterativo y cada iteración comprende el cálculo de las posiciones del TCP (Tool

Center Point) del robot a la base y de la base al TCP. Cuando se inicia el algoritmo en primer lugar se comprueba si el punto es alcanzable, es decir, si la distancia entre la base y el punto objetivo es menor que la longitud del robot. Si el punto no es alcanzable se buscará la configuración que más nos acerque al objetivo.

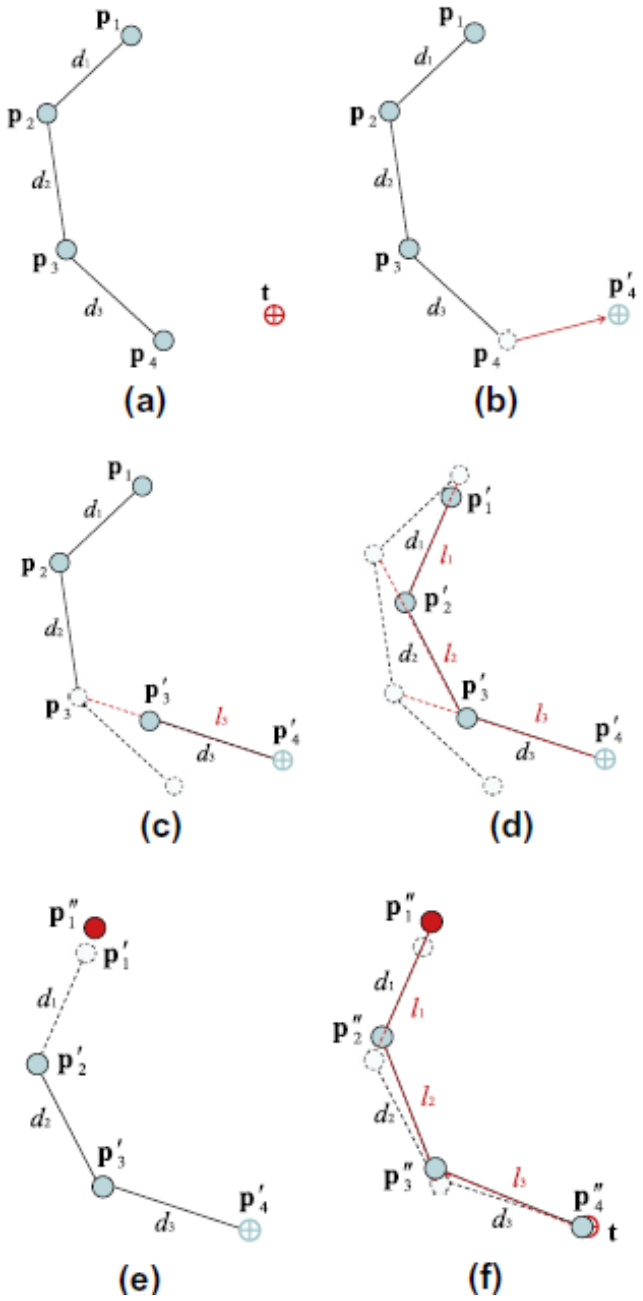


Figura 5.1 Iteración empleando el algoritmo FABRIK

En caso de que el objetivo sí sea alcanzable se iniciará el bucle. Cada iteración del algoritmo está compuesta primero por un recorrido del extremo a la base y un segundo recorrido de la base al extremo. Siempre se realizan en dicho orden siguiendo los siguientes pasos (seguir explicación con la Figura 5.1) [Aristidou, A. 2011]:

a) Se muestra la configuración inicial de un robot de 3 módulos en el que P_1 es la base y P_4 la cabeza del robot. El punto rojo *t* es el punto objetivo al que debe llegar el punto P_4.

b) Se comienza el recorrido de cabeza a base. En primer lugar, se traslada el punto P_4 a la posición de *t* y pasa a llamarse P_4'.

c) Se traza la recta que une los puntos P_4' y P_3. Se sitúa sobre esta recta a una distancia d_3 (longitud del módulo 3) del punto P_4' el nuevo P_3 bajo el nombre de P_3'.

d) Para obtener el punto P_2' se traza una recta de P_3' a P_2 y al igual que en el paso c se coloca el nuevo punto, en este caso P_2', sobre la recta a una distancia d_2 del punto P_3'. Para obtener el punto P_1' habría que repetir los mismos pasos que para los puntos anteriores.

e) Una vez se ha calculado el punto de la base P_1', se comienza el recorrido en sentido opuesto, es decir, de base a cabeza. El punto P_1' se trasladará a la posición en la que se encontraba al inicio (esto se debe a que la base del robot debe permanecer fija, por lo que el punto que la representa al finalizar el algoritmo se debe encontrar en la misma posición). El nuevo punto se llamará P_1''.

f) A continuación, se realizarán los mismos pasos que los descritos en los pasos c y d. Para calcular P_2'' se traza la recta entre P_1'' y P_2' y se sitúa el nuevo punto a una distancia d_1 del punto P_1''. Se repiten los mismos pasos hasta calcular todos los puntos. Una vez se han calculado los puntos se habría finalizado la primera iteración. Se calcularía el error de posición de la cabeza del robot y se compararía con la tolerancia definida. Si se satisface la tolerancia la configuración del robot obtenida sería válida y se finalizaría el bucle. En caso de no cumplirse la tolerancia, se repetirían de nuevo todos los pasos descritos. En caso de no satisfacerse la tolerancia, pero se haya alcanzado el límite de iteraciones marcado, también se finalizaría el bucle.

Con este algoritmo también podemos introducir la orientación del TCP. Al igual que para la posición, para la orientación también se puede establecer una tolerancia mediante el ángulo que forman el vector orientación que queremos y el que se obtiene. El cálculo de la configuración de los puntos se realizará empleando el mismo método que cuando se calcula solamente con la posición, con la excepción del punto P_{n-1}. Para obtener el punto P_{n-1} se buscará el punto a la distancia d_{n-1} del punto P_n en el sentido contrario a la orientación deseada. De este modo el TCP sí tendrá la orientación que queramos [Aristidou, A. 2011]. Este proceso se explica más detalladamente en el Capítulo 6.1.

Además, se pueden imponer restricciones de movimiento que limiten los desplazamientos angulares de cada articulación, así como los ejes entorno a los que se realice el giro. De esta forma se puede simular de forma bastante realista el robot con las particularidades que lo caractericen. En la Figura 5.2 se muestra cómo se calcula la trayectoria con limitación del ángulo de giro de los módulos. El giro está limitado en ángulo, pero no en los ejes respecto a los que se puede realizar. A continuación, se explica cada imagen de la Figura 5.2, [Aristidou, A. 2011]:

a) Se muestra la posición inicial de los módulos, el punto objetivo *t*, las orientaciones empleando los cuadrados abiertos rojos y la orientación objetivo (cuadrado rojo abierto en *t*). El punto P_1 corresponde a la base del robot y el punto P_4 a la cabeza.

b) Al igual que en la explicación de la Figura 5.1, se traslada el punto P_4 que corresponde a la cabeza a la posición del punto objetivo *t.*

c) El punto P_3' se obtiene de la forma explicada previamente en la Figura 5.1 debido a que no hay ningún módulo anterior que restrinja la orientación del módulo 3.

d) Se reorienta el cuadrado abierto correspondiente a P_3' de tal forma que la línea central del cuadrado abierto sea perpendicular y las líneas laterales sean paralelas a la recta del módulo.

e) A partir de este punto se comienzan a aplicar restricciones de giro entre los módulos. Para ello se crea un cono cuyo eje de revolución coincide con la orientación del módulo 3. El ángulo máximo que pueden girar los módulos se denomina α. Este ángulo es el que forman el eje de revolución $\vec{n}_3$ que corresponde a la orientación del módulo anterior y la hipotenusa del triángulo revolucionado, Figura 5.3. Para obtener P_2' se sigue el proceso habitual trazando la recta entre P_3' y P_2.

f) El punto P_2' debe encontrarse en el interior del cono. En caso de que se encuentre en el exterior se situará sobre la superficie del cono, ya que es el lugar que más se aproxima al punto ideal que cumple la restricción de giro.

g) Se sitúa P_2' a una distancia d_2 del punto P_3'.

h) Se orienta el cuadrado abierto conforme la orientación del módulo 2. El punto P_1' se obtendría de la misma manera que el punto P_2'. Al finalizar el recorrido se trasladaría P_1' a la posición en la que se encontraba P_1 y se repetirían los mismos pasos en sentido opuesto.

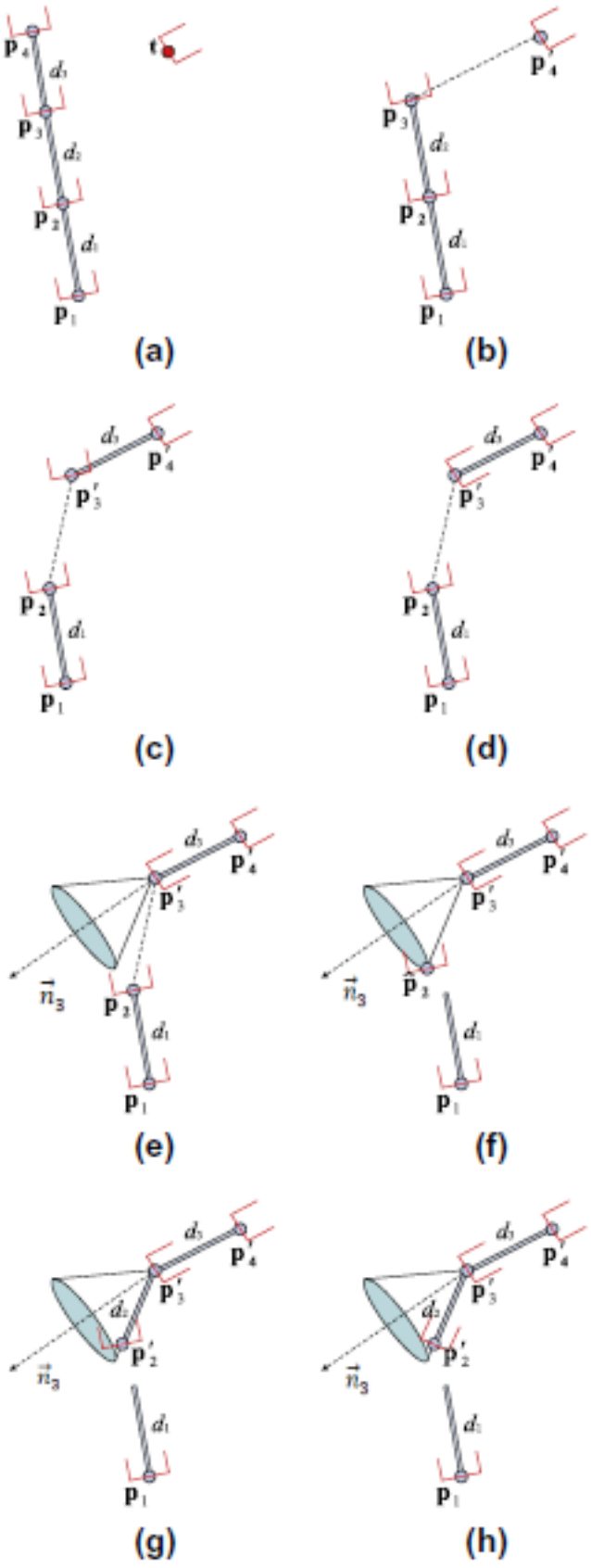


Figura 5.2 Algoritmo FABRIK con restricciones de giro entre módulos

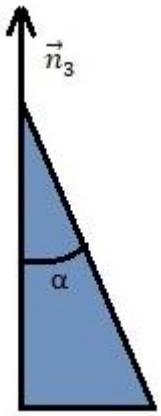


Figura 5.3 Triánglo de revolución de los conos

5.1.2 Algoritmo Cyclic Coordinate Descent (CCD)

El algoritmo Cyclic Coordinate Descent (CCD) es un método de cálculo de la cinemática inversa. Tiene ciertas similitudes en cuanto a sus características con el método FABRIK. Una de ellas es que es un método heurístico, por lo tanto, el resultado es una aproximación del punto objetivo que se obtiene mediante prueba y error. Este algoritmo al igual que FABRIK tiene baja carga computacional y es rápido. Las principales desventajas es que cabe la posibilidad de que no converja hacia la solución correcta y de que en caso de sí hacerlo el resultado sea poco natural.

El algoritmo consiste en un bucle que se repite hasta conseguir una solución conforme a la tolerancia establecida o hasta que se alcance el número máximo de iteraciones fijado. Para ejecutar el algoritmo es necesario el número de módulos, la longitud de cada módulo, la posición inicial de cada punto, el punto objetivo, la tolerancia y el número máximo de iteraciones. En primer lugar, como en el resto de los algoritmos, se comprobará que el punto es alcanzable por el robot mediante la comparación de la longitud total del robot y la distancia de la base al punto objetivo. En caso de que no se pueda alcanzar se buscará la configuración que más se aproxime al objetivo, que generalmente será el robot completamente extendido como si fuera una recta [Song, W. 2011].

En caso de que sí sea alcanzable se comenzará el bucle. Existen dos formas de realizarlo, del extremo a la base y de la base al extremo. Debido a que al realizarlo de la base al extremo los módulos que más varían su posición angular son los de la base y en nuestro robot interesa lo contrario explicaremos solamente el caso del extremo a la base, pese a esto en algunas imágenes aparecerán ambos métodos.

El algoritmo consiste en rotar una a una las articulaciones, comenzando por la más próxima al TCP, para alinear el TCP con el punto objetivo y el punto de la articulación que se está rotando. En la Figura 5.4 el TCP corresponde al punto P_4 y a base al punto P_1. A continuación se explican los pasos que siguen el algoritmo (seguir Figura 5.4):

a) Se muestra la configuración de partida del robot y el punto objetivo t. Se trata de un robot de 3 módulos. Ve es el vector formado por el punto de la articulación a rotar, en este caso P_3 y el TCP que en la figura es P_4. $\vec{v}_t$ es el vector formado por el punto de la articulación que se va a rotar y el punto objetivo t. θ es el ángulo formado entre los vectores $\vec{v}_e$ y $\vec{v}_t$, se obtiene mediante (Ec. 5.1).

b) Se rota el punto P_4 un ángulo θ respecto a P_3 para que así se logre alinear los puntos P_3, P_4' y t. El eje $\vec{v}_r$ que se ha empleado para rotar el punto P_4 es el vector perpendicular al plano formado por los vectores $\vec{v}_e$ y $\vec{v}_t$ y se obtiene mediante la ecuación (Ec. 5.2). Al finalizar la iteración se comprueba el error de posición. En caso de cumplirse la tolerancia se finalizaría el bucle. Si el error supera la tolerancia se realizará otra iteración, rotando la siguiente articulación, en este caso P_2.

c) Se calculan los nuevos vectores $\vec{v}_e$ y $\vec{v}_t$ y se obtiene el ángulo θ.

d) Se rotan los puntos P_3' y P_4' el ángulo θ calculado respecto a P_2'. El vector del eje de giro, $\vec{v}_r$, se obtiene de la misma manera que en la iteración anterior. Al finalizar la iteración se calcula el error de posición para determinar si la solución alcanzada es válida.

e) En la tercera iteración se vuelve a calcular el ángulo θ de la misma manera que en las iteraciones anteriores.

f) Se rotan los puntos P_2'', P_3'' y P_4''. Se puede apreciar como los módulos rotan como un sólido rígido respecto a P_1''. Se vuelve a calcular el error de posición y a compararlo con la tolerancia.

g) Una vez se han rotado todos los puntos, en caso de no satisfacer aún la tolerancia, se retrocederá a la situación inicial en la que la articulación de giro es la más próxima al extremo y se seguirán los mismos pasos que en el resto de las iteraciones anteriores.

h) Se rotaría respecto a P_3''' para obtener el punto P_4^{IV} y se volvería a calcular el error. El bucle se repetirá hasta que se obtenga una solución satisfactoria conforme a la tolerancia, o bien se alcance el límite máximo de iteraciones marcado.

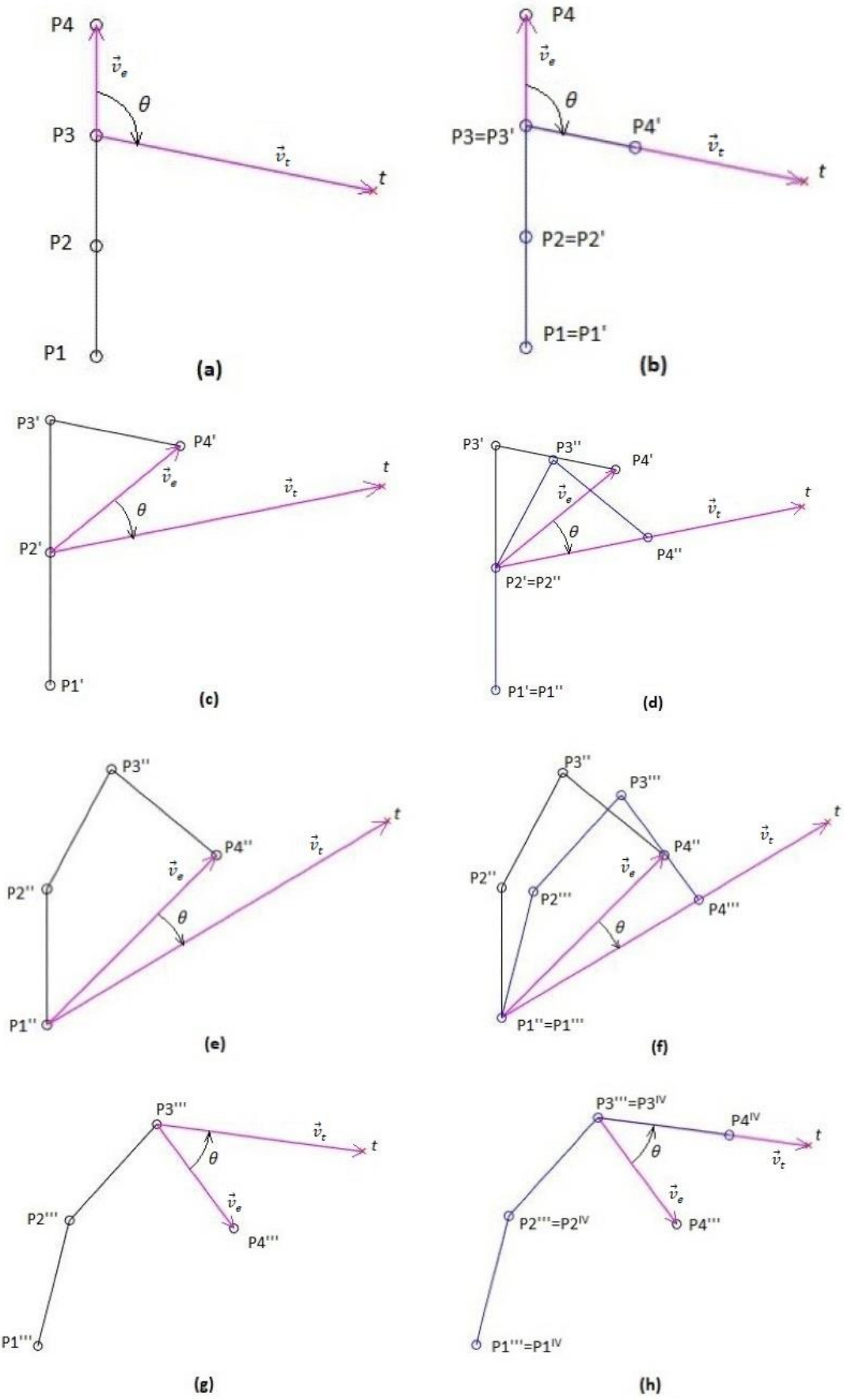


Figura 5.4 Explicación del algoritmo CCD

$$\theta = \cos^{-1}\left(\frac{\vec{v}_e \cdot \vec{v}_t}{|\vec{v}_e| \cdot |\vec{v}_t|}\right) \qquad (\boldsymbol{Ec.5.1})$$

$$\vec{v}_r = \frac{\vec{v}_e \times \vec{v}_t}{|\vec{v}_e \times \vec{v}_t|} \qquad (\boldsymbol{Ec.5.2})$$

Existe una variante de este algoritmo. Ésta consiste en realizar un "rebote" que consiste en volver al módulo de cabeza antes de haber calculado las posiciones de todas las articulaciones. En concreto, se comienza en la primera iteración actuando sobre las dos articulaciones más próximas al TCP, en este momento, en lugar de actuar en la tercera, se finaliza la iteración, se comprueba si se cumple la tolerancia y en caso de no cumplirse se iniciará una nueva. En esta iteración se actuará sobre una articulación más que en la iteración anterior. Se repetirá este proceso hasta que o bien se haya alcanzado un resultado satisfactorio o se llegue a la iteración dónde se desplacen todas las articulaciones. Si se ha llegado a la iteración en la que se utilizan todas las articulaciones y aun así el resultado no es válido se comenzará otra vez el proceso con las dos últimas articulaciones y se repetirá otra vez el bucle. Esta variante del algoritmo CCD provoca que los módulos próximos a la base apenas se muevan como se observa en la Figura 5.5 [Kenwright, B. 2012].

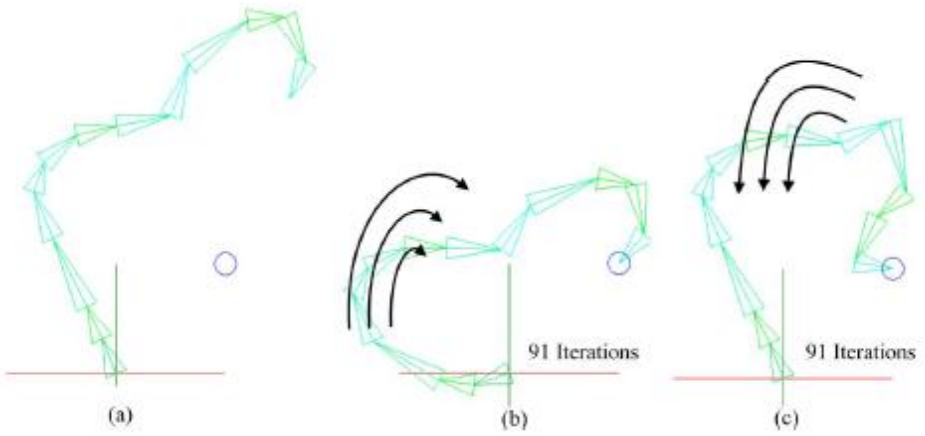


Figura 5.5 Demostración algoritmo CCD con rebote. (a) Posición inicial. (b) Algoritmo de la base al extremo con rebote. (c) Algoritmo del extremo a la base con rebote.

Al igual que en FABRIK con CCD también se pueden añadir restricciones angulares al robot para evitar que los módulos giren más de un ángulo determinado y así se pueda representar lo más fielmente al robot real. Sin embargo, a la hora de añadir al algoritmo una orientación determinada se convierte en una tarea muy complicada.

5.1.3 Algoritmo Follow The Leader (FTL)

El algoritmo Follow The Leader (FTL) conocido en castellano como seguimiento del líder es un método muy utilizado en robots de tipo serpiente como es el nuestro. El funcionamiento como su propio nombre indica consiste en que el último módulo toma la configuración más conveniente para alcanzar el punto de la trayectoria que corresponda. El resto de los módulos van repitiendo las configuraciones que ha tomado el primer módulo a medida que pasan por los diferentes puntos de la trayectoria. Se puede aplicar para diferentes situaciones.

Una de estas situaciones es que el último módulo sea teleoperado por una persona que decide en qué dirección debe ir. El resto de los módulos se limitará a seguirlo [Williams, R. 1997]. Otra situación es que haya un camino a seguir y a medida que el robot avanza se van calculando las posiciones de los módulos. Por lo general, el avance que se realiza en cada paso es menor que la longitud de los módulos. Es por esto por lo que se tienen que calcular las configuraciones intermedias que hay desde que el último módulo pasa por un punto hasta que el siguiente módulo llega a ese mismo punto. Por lo general estos cálculos se hacen con métodos geométricos ya que el cálculo numérico es más lento y costoso [Xie, H. 2019].

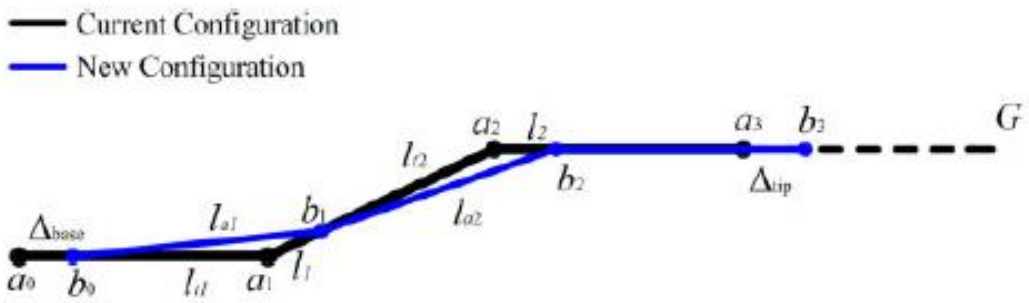


Figura 5.6 Configuración inicial y configuración nueva al avanzar en un robot tipo serpiente que emplea el algoritmo FTL.

Se realizará una comparación entre los algoritmos FABRIK y CCD sin ninguna restricción para comprobar cuál es más preciso, más rápido y en consecuencia seleccionar uno para continuar su desarrollo y finalmente aplicarlo en una simulación del robot. El algoritmo FTL no se puede comparar en la prueba que se va a hacer, pues la prueba será de alcanzar un punto sin que el robot avance y el algoritmo FTL requiere del avance del robot para que funcione, pero se puede aplicar combinado con el algoritmo de cinemática inversa que decidamos desarrollar.

5.2 Definición de errores

Se utilizarán tres tipos de errores: posición, orientación y global. En las tablas y figuras los errores se calculan para el último punto en el caso de la posición y el último módulo para la orientación.

5.2.1 Error de posición

El error de posición es la distancia entre el punto que se marcado como objetivo y el punto que realmente se ha alcanzado. Para obtener está distancia se resta las coordenadas del punto obtenido a las coordenadas del punto objetivo (Ec.5.3) (o viceversa pues el módulo es un escalar y la distancia debe ser positiva siempre) y se calcula el módulo del vector obtenido de la resta anterior (Ec. 5.4). En la Figura 5.8 la posición objetivo es $\vec{p}^0$ y la posición alcanzada es $\vec{p}$.

$$\vec{e}_p = \vec{P}_{Objetivo} - \vec{P}_{alcanzado} = \vec{p}^0 - \vec{p} = (x_p, y_p, z_p) \qquad (\boldsymbol{Ec.5.3})$$

$$e_p = |\vec{e}_p| = \sqrt{{x_p}^2 + {y_p}^2 + {z_p}^2} \qquad (\boldsymbol{Ec.5.4})$$

5.2.2 Error de orientación

El error de orientación es el ángulo formado entre el vector orientación objetivo y el vector orientación obtenido. Se calcula realizando la inversa del coseno del resultado obtenido de realizar el producto escalar del vector orientación objetivo y el vector orientación obtenido (Ec. 5.5). Los vectores deben ser unitarios. En la Figura 5.7 la orientación objetivo es $\vec{d}^0$ y la dirección obtenida es $\vec{d}$.

$$e_{ori} = \cos^{-1}(\vec{d}^0 \cdot \vec{d}) \qquad (\boldsymbol{Ec.5.5})$$

Lo habitual es calcular los ángulos en radianes, pero en este trabajo se mostrarán en grados sexagesimales con el objetivo de que resulte más sencillo e intuitivo interpretar los resultados.

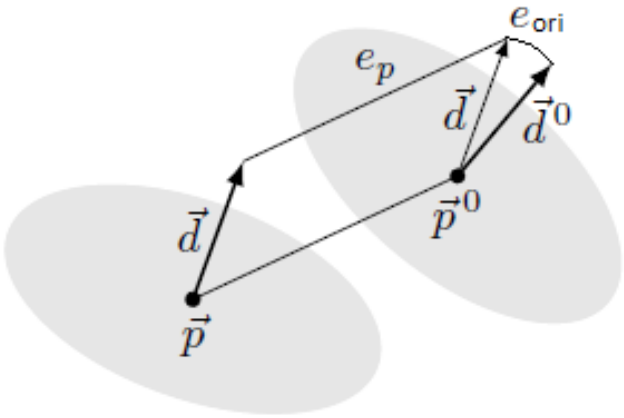


Figura 5.7 Errores de posición y orientación

5.2.3 Error global

El error global es una medida adimensional que combina el error de posición con el error de orientación, de esta forma se obtiene una forma para ver como de preciso es el resultado. Se establece un error de referencia de posición y otro de orientación que en este trabajo serán las tolerancias de posición y orientación. Se multiplica cada fracción por un factor según el peso que se le quiera dar a cada error, cuanto más peso queramos que tenga mayor debe ser el factor. La suma del factor de posición y el factor de orientación debe ser 1. El error global se obtiene mediante la suma de los errores de posición y orientación normalizados al ser divididos por los errores de referencia (Ec. 5.6).

$$error_{global} = \frac{error_{posición}}{error_{posición}{}^{ref}} \cdot X_{pos} + \frac{error_{orientación}}{error_{orientación}{}^{ref}} \cdot X_{ori} \qquad (\boldsymbol{Ec.\,5.6})$$

5.3 Comparación de los algoritmos FABRIK Y CCD y análisis de los resultados

Se realizarán varias pruebas en 2D y en 3D y también con diferente número de módulos, concretamente con 5 y 10 módulos. La disposición inicial del robot tiene el primer punto en el origen (0, 0, 0) y se encuentra recto a lo largo del eje 'Y' en el sentido positivo del mismo. Cada eslabón mide 2 unidades. El punto objetivo se denominará *P.*

Las pruebas se han realizado empleando Matlab R2020a en un ordenador con procesador Intel core i7. Por lo tanto, los tiempos de ejecución de los algoritmos pueden variar si se cambian estos parámetros.

5.3.1 Ensayo 1: 5 módulos – 2D

Ensayo 1: PRUEBA 1

- P = (3, 9, 0)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.8 y Tabla 5.1.

	FABRIK	CCD
Tiempo requerido (s)	0.001	0.073
Nº iteraciones	12	16
Error de Posición	0.00874	0.00907

Tabla 5.1 Resultados prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (3, 9, 0)

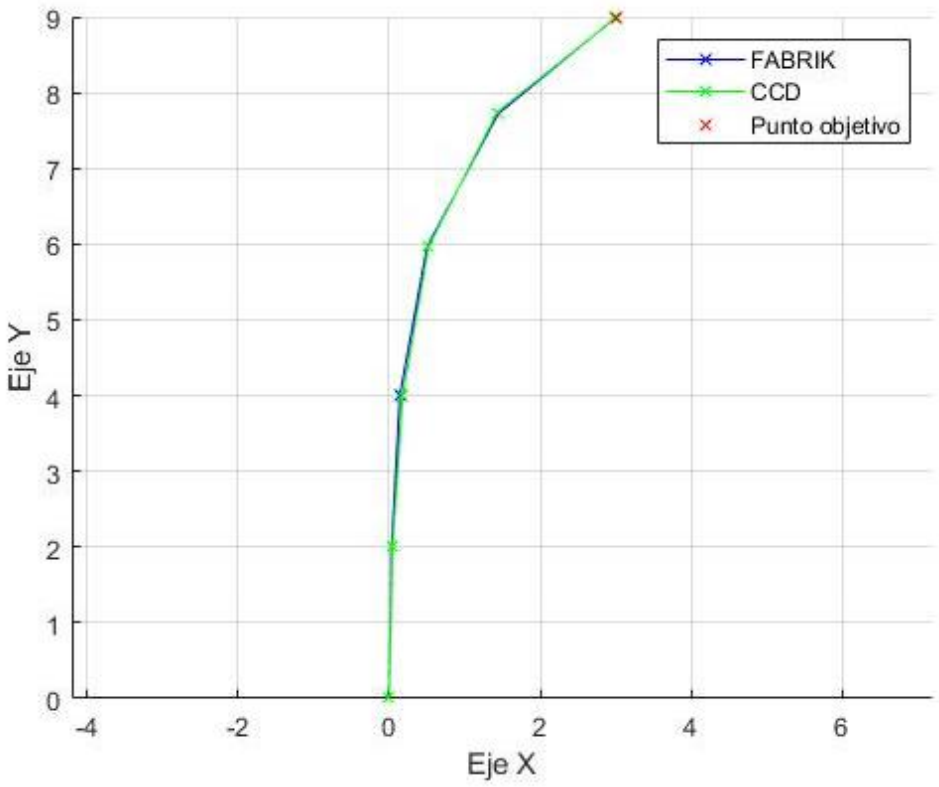


Figura 5.8 Prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (3, 9, 0)

Ensayo 1: PRUEBA 2

- P = (7, 5, 0)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.9 y Tabla 5.2.

	FABRIK	CCD
Tiempo requerido (s)	0.003	0.084
Nº iteraciones	8	20
Error de Posición	0.00666	0.00921

Tabla 5.2 Resultados prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (7, 5, 0)

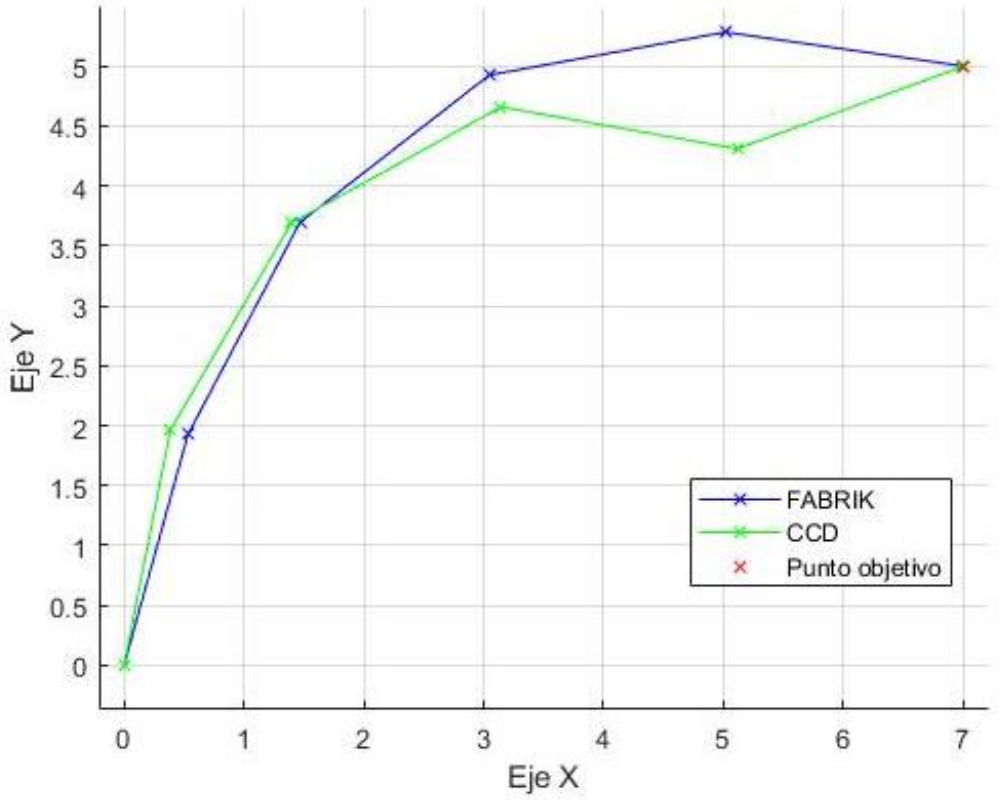


Figura 5.9 Prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (7, 5, 0)

Ensayo 1: PRUEBA 3

- P = (6, -5, 0)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.10 y Tabla 5.3.

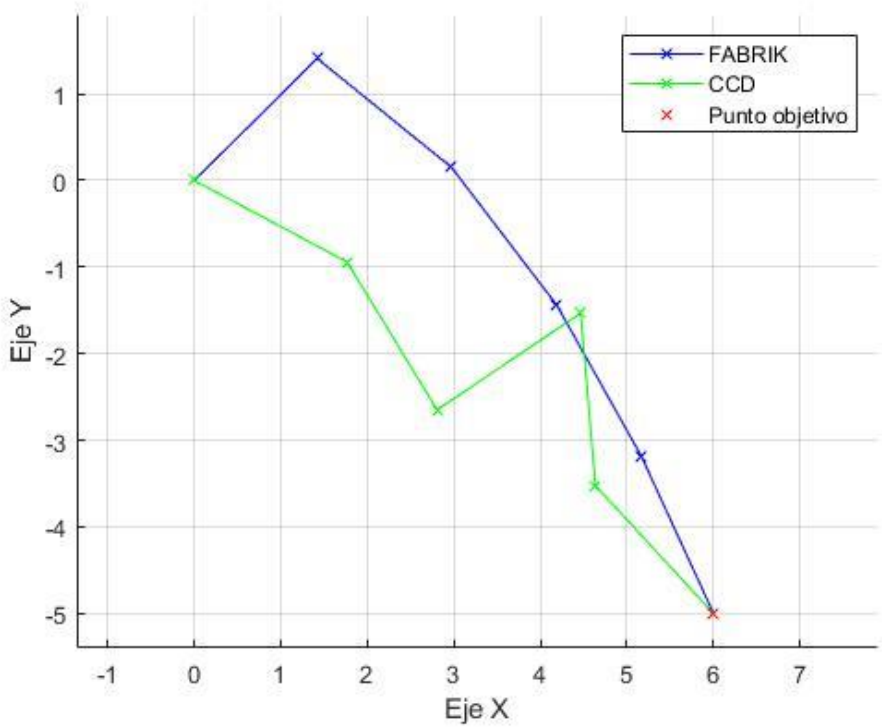


Figura 5.10 Prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (6, -5, 0)

	FABRIK	CCD
Tiempo requerido (s)	0.001	0.100
Nº iteraciones	2	30
Error de Posición	0.00055	0.00907

Tabla 5.3 Resultados prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (6, -5, 0)

En estas tres pruebas se ve claramente que el algoritmo FABRIK es más rápido y preciso que el CCD y además da unos resultados más naturales como se aprecia en la Figura 5.10.

5.3.2 Ensayo 2: 5 módulos – 3D

Ensayo 2: PRUEBA 1

- P = (4, 2, 7)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.11 y Tabla 5.4.

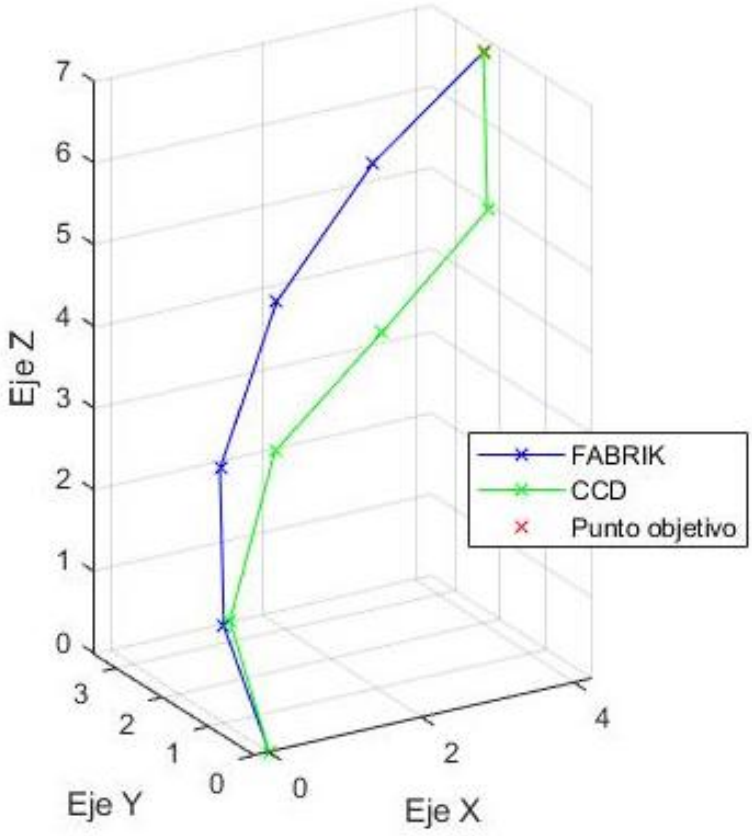


Figura 5.11 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (4, 2, 7)

	FABRIK	CCD
Tiempo requerido (s)	0.001	0.063
Nº iteraciones	7	15
Error de Posición	0.00496	0.00775

Tabla 5.4 Resultados prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (4, 2, 7)

Ensayo 2: PRUEBA 2

- P = (3, -2, 7)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.12 y Tabla 5.5.

	FABRIK	CCD
Tiempo requerido (s)	0.001	0.038
Nº iteraciones	4	8
Error de Posición	0.00356	0.00648

Tabla 5.5 Resultados prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (3, -2, 7)

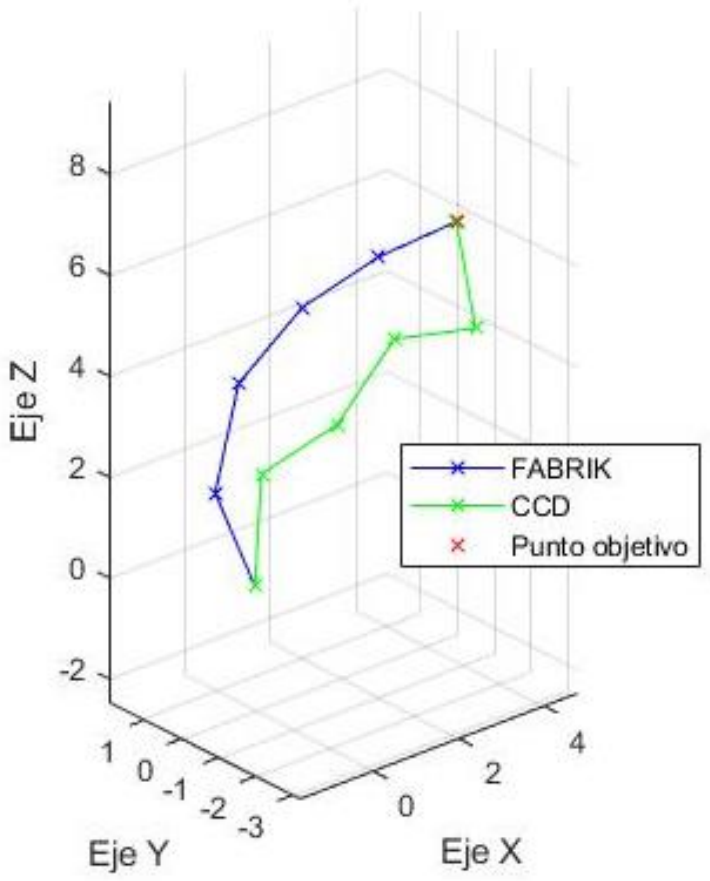


Figura 5.12 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (3, -2, 7)

Ensayo 2: PRUEBA 3

- P = (3, -5, -3)
- Tolerancia = 0,01
- Límite de iteraciones

Resultados: Figura 5.13 y Tabla 5.6.

	FABRIK	CCD
Tiempo requerido (s)	0.001	0.064
Nº iteraciones	3	18
Error de Posición	0.00680	0.00933

Tabla 5.6 Resultados prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (3, -5, -3)

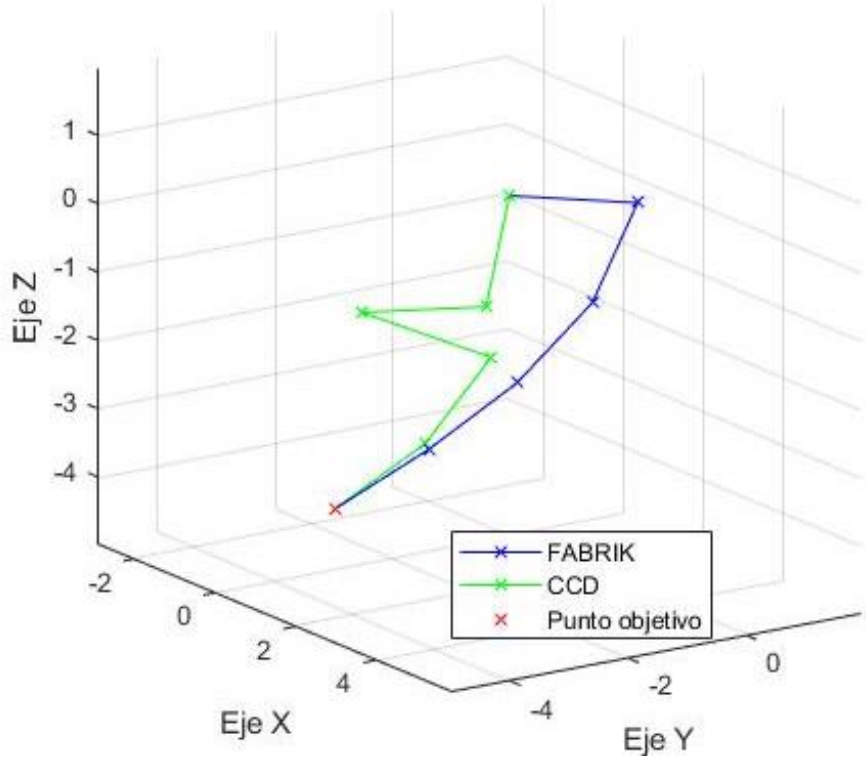


Figura 5.13 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (3, -5, -3)

Al igual que en 2D el comportamiento del algoritmo FABRIK es mejor que el de CCD ya que obtienen un resultado satisfactorio en menor tiempo, con menos iteraciones, por lo tanto, la carga computacional es menor y con ello el tiempo que tarda en obtener una solución. A continuación,

comprobaremos el comportamiento cuando el robot tiene 10 eslabones. A excepción de los eslabones el resto de los parámetros son los mismos tanto la longitud de cada eslabón como la posición inicial.

5.3.3 Ensayo 3: 10 módulos – 2D

Ensayo 3: PRUEBA 1

- P = (15, 6, 0)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.14 y Tabla 5.7.

	FABRIK	CCD
Tiempo requerido (s)	0.031	0.432
Nº iteraciones	12	12
Error de Posición	0.00890	0.00646

Tabla 5.7 Resultados prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (15, 6, 0)

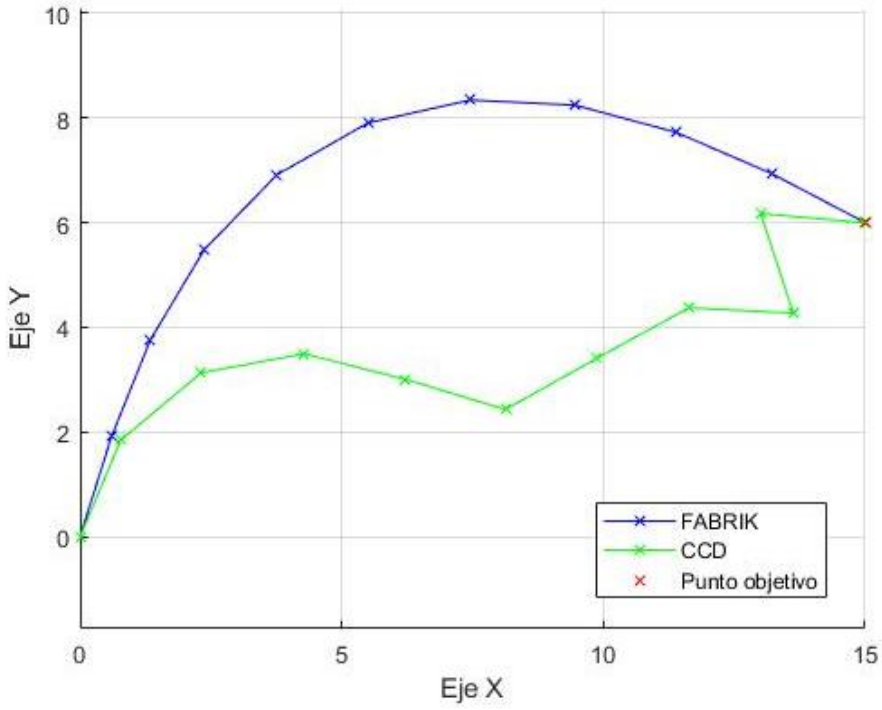


Figura 5.14 Prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (15, 6, 0)

Ensayo 3: PRUEBA 2

- P = (10, 0, 0)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figura 5.15 y Tabla 5.8.

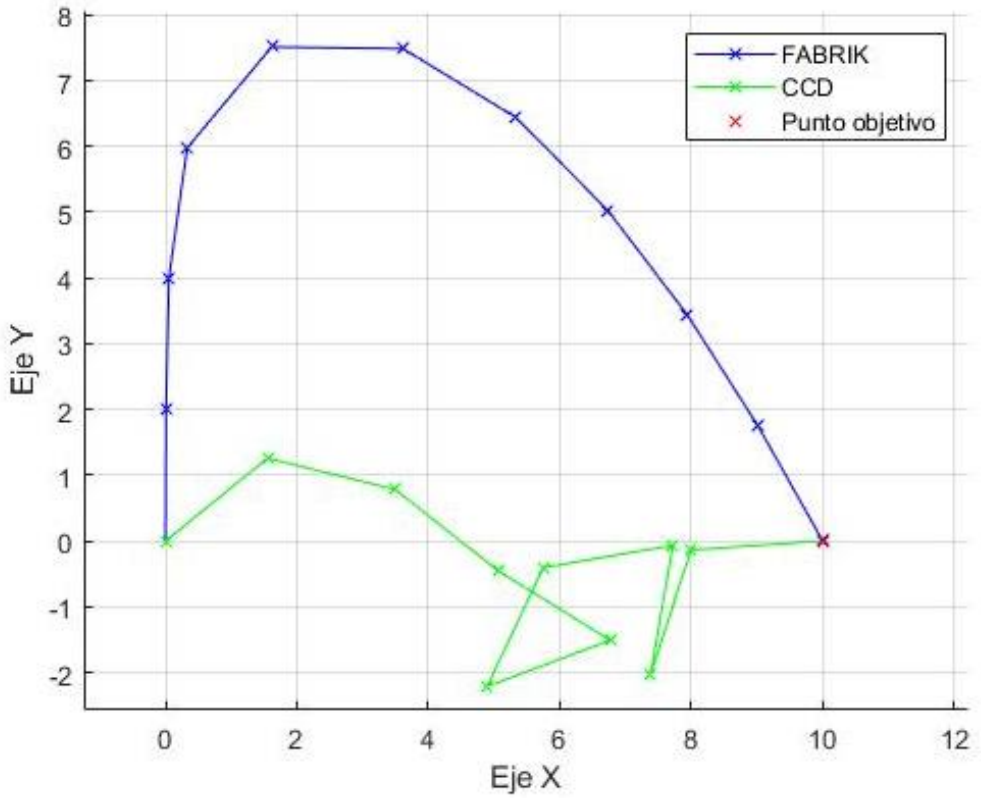


Figura 5.15 Prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (10, 0, 0)

	FABRIK	CCD
Tiempo requerido (s)	0.008	0.447
Nº iteraciones	3	15
Error de Posición	0.00478	0.00634

Tabla 5.8 Resultados prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (10, 0, 0)

Ensayo3: PRUEBA 3

- P = (6, -10, 0)
- Tolerancia = 0,01
- Límite de iteraciones = 30

Resultados: Figura 5.16 y Tabla 5.9.

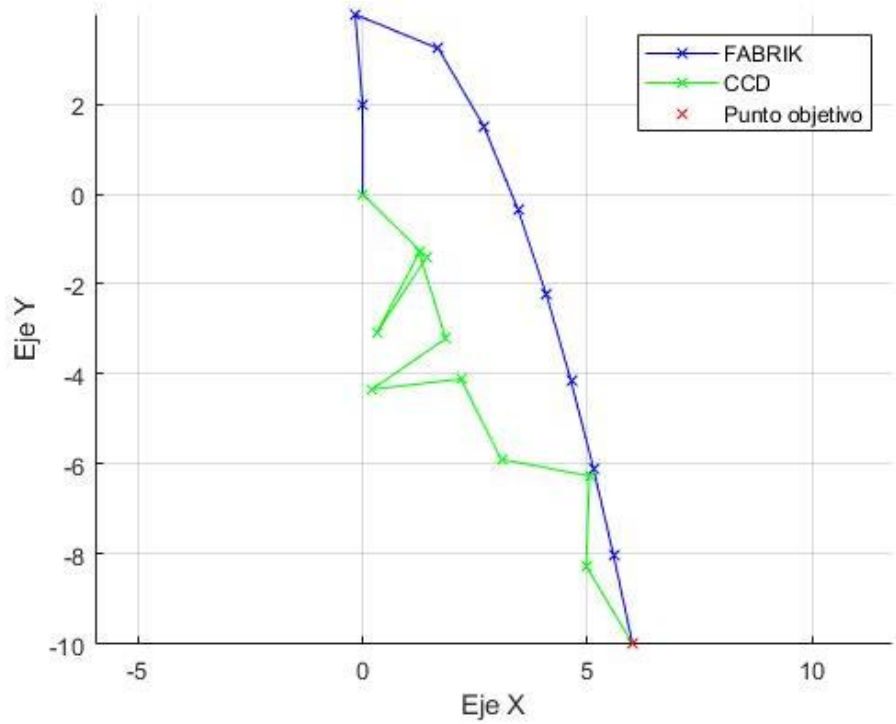


Figura 5.16 Prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (6, -10, 0)

	FABRIK	CCD
Tiempo requerido (s)	0.005	0.513
Nº iteraciones	2	30
Error de Posición	0.00366	0.01510

Tabla 5.9 Resultados prueba 2D de los algoritmos FABRIK y CCD Punto objetivo (6, -10, 0)

Al igual que con 5 módulos, el tiempo de cálculo es menor al utilizar el algoritmo FABRIK. El error es similar, pero al necesitar más iteraciones el algoritmo CCD para cumplir la tolerancia cabe el riesgo de que ocurra como en la tercera prueba dónde podemos ver en la Tabla 5.9 que el error es mayor que la tolerancia, pero al haber alcanzado el límite de iteraciones se ha finalizado el bucle. Otro problema destacable que aparece con el algoritmo

CCD es que la configuración es poco natural y en ocasiones el robot se enrolla sobre sí mismo.

5.3.4 Ensayo 4: 10 módulos – 3D

Ensayo 4: PRUEBA 1

- P = (6, 10, 4)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figuras 5.17, 5.18 y Tabla 5.10.

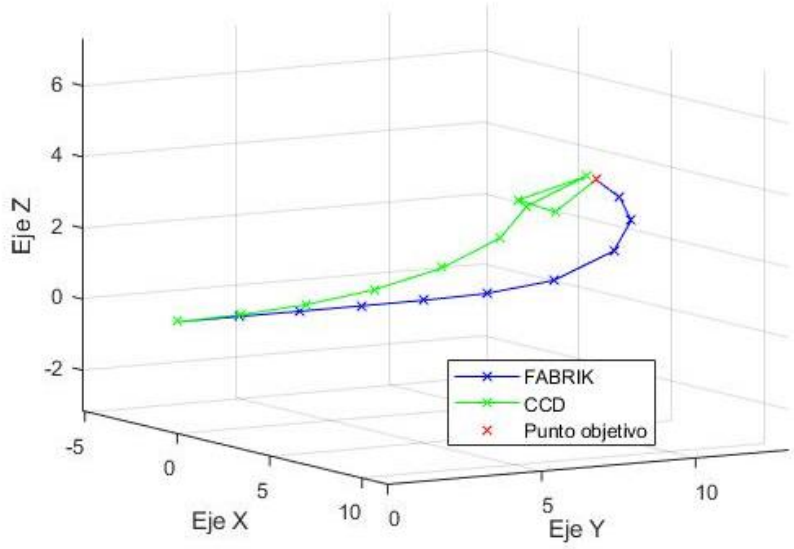


Figura 5.17 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (6, 10, 4) (Vista 1)

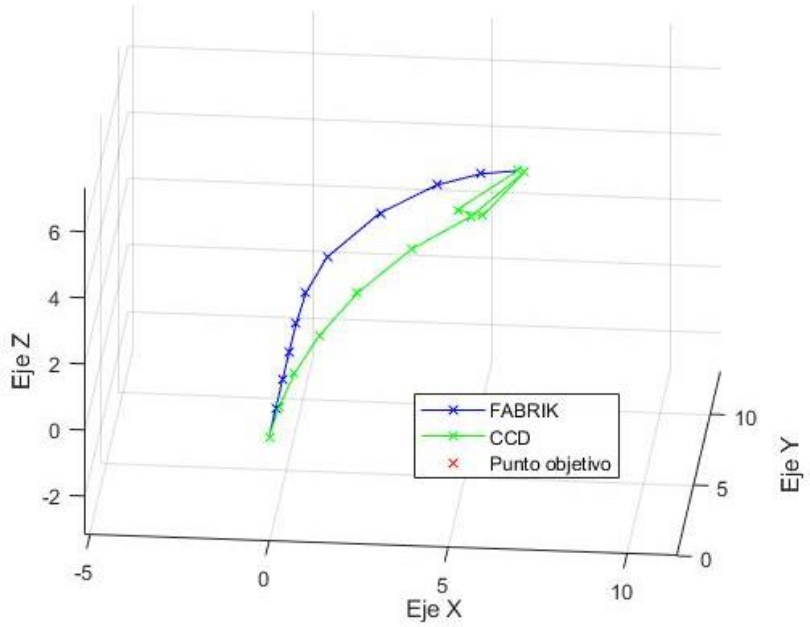


Figura 5.18 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (6, 10, 4) (Vista 2)

	FABRIK	CCD
Tiempo requerido (s)	0.023	0.165
Nº iteraciones	3	4
Error de Posición	0.00620	0.00299

Tabla 5.10 Resultados prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (6, 10, 4)

Ensayo 4: PRUEBA 2

- P = (-7, 10, 7)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figuras 5.19-5.20 y Tabla 5.11.

	FABRIK	CCD
Tiempo requerido (s)	0.001	0.105
Nº iteraciones	6	7
Error de Posición	0.00954	0.00289

Tabla 5.11 Resultados prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (-7, 10, 7)

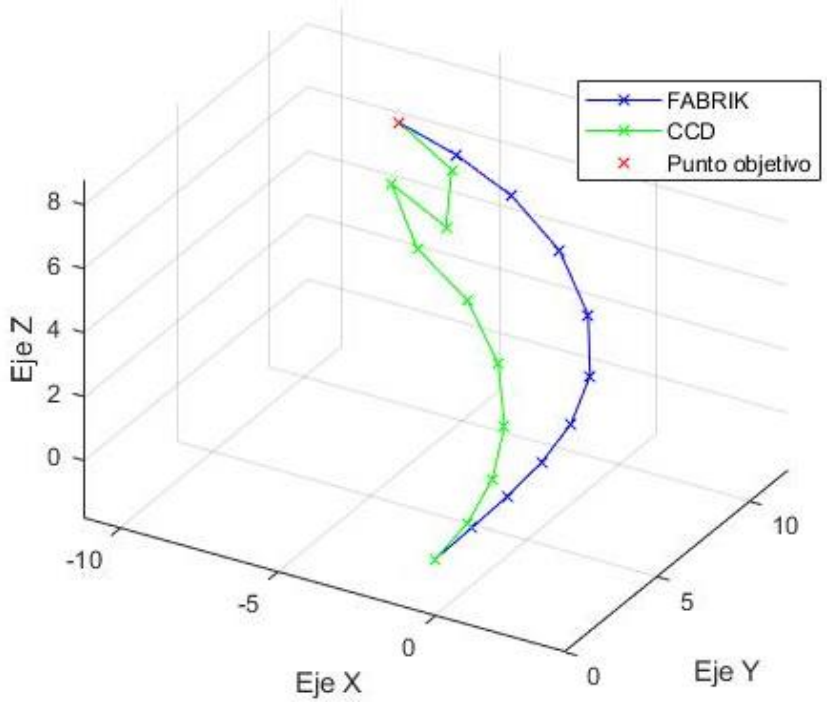


Figura 5.19 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (-7, 10, 7) (Vista 1)

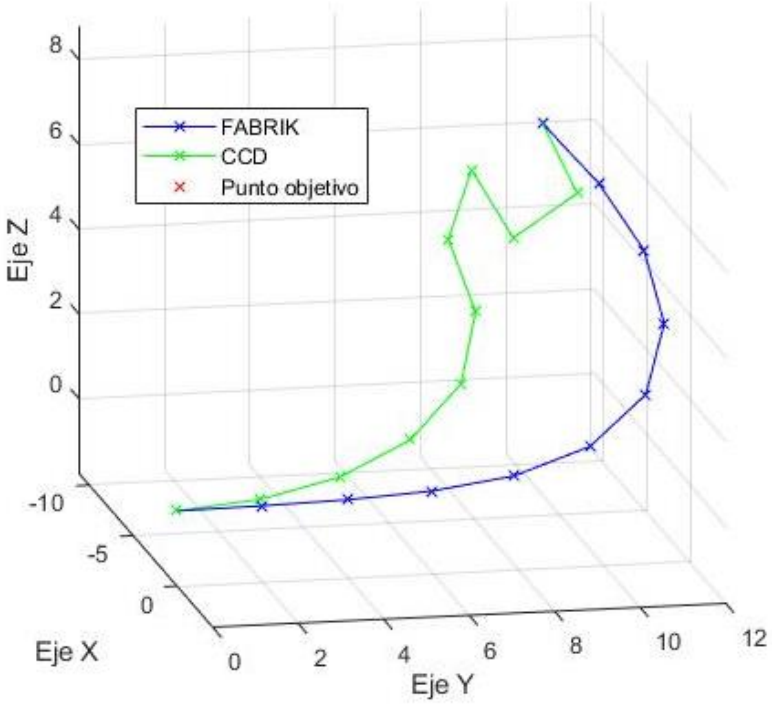


Figura 5.20 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (-7, 10, 7) (Vista 2)

Ensayo 4: PRUEBA 3

- P = (-11, -5, 3)
- Tolerancia = 0,01
- Límite de iteraciones: 30

Resultados: Figuras 5.21,5.22 y Tabla 5.12.

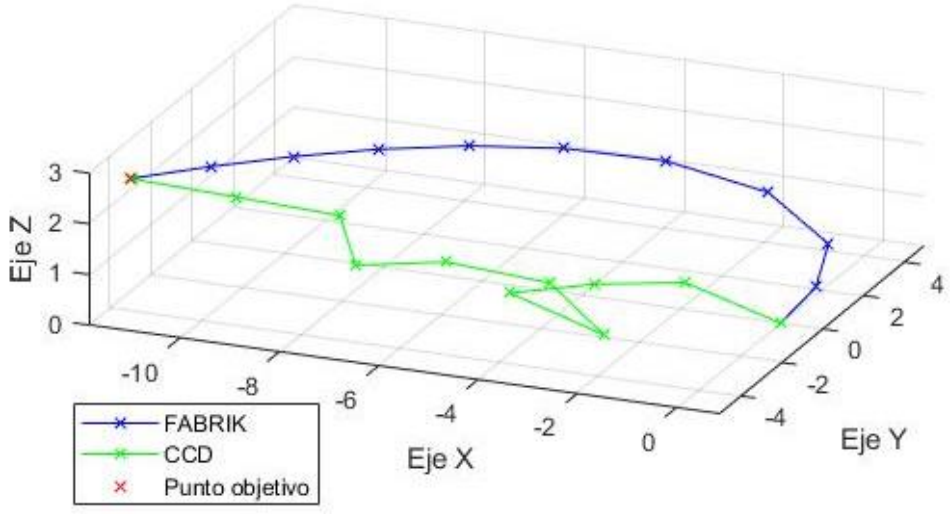


Figura 5.21 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (-11, -5, 3) (Vista 1)

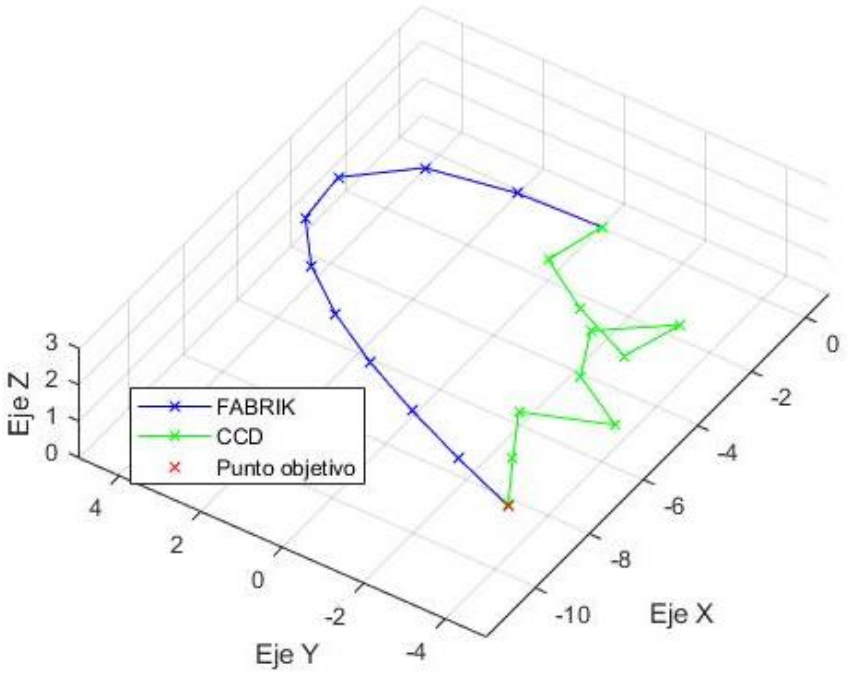


Figura 5.22 Prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (-11, -5, 3) (Vista 2)

	FABRIK	CCD
Tiempo requerido (s)	0.016	0.642
Nº iteraciones	3	30
Error de Posición	0.00670	0.02937

Tabla 5.12 Resultados prueba 3D de los algoritmos FABRIK y CCD Punto objetivo (-11, -5, 3)

Como se puede comprobar en estas tres últimas pruebas el tiempo requerido para alcanzar el objetivo con el algoritmo FABRIK es menor que con el algoritmo CCD. Además, pese a que en las dos primeras pruebas el algoritmo CCD sí que ha alcanzado el objetivo casi en las mismas iteraciones que el algoritmo FABRIK, en la última se ha llegado al límite de iteraciones y el error de posición era casi el triple que el límite que habíamos marcado por la tolerancia. Esto nos indica que cuando el punto está en una posición poco favorable respecto a la posición de la que partimos (en este caso estaba en el lado negativo del eje 'y' cuando el robot inicialmente se encontraba estirado en el sentido positivo del eje 'y') el algoritmo CCD tiene problemas para alcanzar el objetivo. Por último, lo que más llama la atención a primera vista es la forma en la que se enrolla el robot sobre sí mismo al emplear el algoritmo CCD.

5.3.5 Conclusión de la comparación de los algoritmos FABRIK y CCD

Tras las pruebas realizadas es evidente que el algoritmo FABRIK es el más adecuado para desarrollar el algoritmo de seguimiento de trayectorias. Además, hay que tener en cuenta que el robot debe presentar restricciones de giro entre módulos. Estas restricciones resultan demasiado laboriosas de programar en el algoritmo CCD debido al funcionamiento que este tiene. Por lo tanto, el mejor algoritmo de los estudiados para realizar el seguimiento de la trayectoria es el algoritmo FABRIK.

CAPÍTULO 6

6. Adaptación y mejora del algoritmo de FABRIK

Una vez se ha decidido que el algoritmo que se empleará es el de FABRIK se debe adaptar a las capacidades del robot aplicando las restricciones correspondientes. Debido a que el algoritmo del que partimos es bastante básico, deberemos desarrollar el código inicial para introducir las restricciones necesarias para simular al robot.

En primer lugar, se eliminará la parte del código que comprueba si el punto objetivo es alcanzable, para que, aunque no sea alcanzable el robot aplique el algoritmo y quede con una configuración más natural, en lugar de trazar una línea recta con sus módulos. A continuación, se añadirá la restricción de orientación del último módulo del robot, ya que sería necesario poder orientar la cámara y las herramientas que lleve el robot. La segunda restricción es el giro permitido entre los módulos. Como se mostró en la descripción del robot, los actuadores son binarios por lo tanto el ángulo girado entre módulos, que se denominará β (Figura 6.1), solamente puede valer 0 ó ±α y además habrá que limitar los ejes de giro, ya que solamente se puede girar respecto a 2 ejes. Recordemos que α es el ángulo máximo que pueden girar dos módulos entre sí, determinado por los parámetros geométricos del robot (Ec. 4.1).

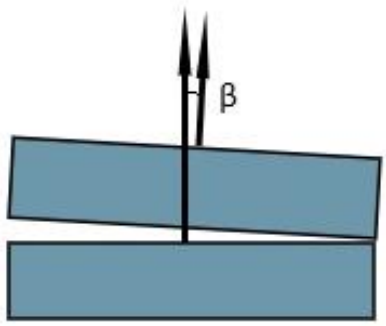


Figura 6.1 Ángulo de giro entro dos módulos, β

Se irán añadiendo paso a paso las restricciones y se realizarán las modificaciones necesarias para que el algoritmo funcione correctamente. Por último, se tratará de seguir una trayectoria para comprobar que el algoritmo sí que funciona en la simulación de un caso real.

Los errores de orientación que se muestran es el ángulo formado entre el vector de orientación objetivo y el vector formado por los puntos P_n y P_{n-1} y se mostrará en grados.

6.1 Restricción de orientación en el extremo

La restricción de orientación se aplica sobre el último módulo. Se aplica cuando se realiza el recorrido desde el extremo del robot a la base. Se aplica siguiendo los siguientes pasos (pasos explicados respecto a la Figura 6.2):

1) Se traslada el punto P_6 al punto objetivo t y toma el nombre P_6'.

2) En lugar de trazar la recta entre los puntos P_5 y P_6' como se haría sin la condición de orientación obteniendo P_5*, se situará el punto P_5' a una distancia l_5 de P_6' en el sentido opuesto a la orientación deseada $\vec{d}$.

3) A partir de este punto, el algoritmo realiza los mismos pasos que se explicaron en el apartado 5.1.1. Trazando la recta entre P_5' y P_4, situando P_4' a una distancia l_4 sobre la propia recta.

4) Se repetirá esta operación sucesivamente hasta obtener P_1'.

5) El recorrido de base a extremo no está afectado por la restricción de orientación, por lo que se calculan los puntos de la misma manera que se calcularon en el recorrido de extremo a base una vez se había obtenido el punto P_5'.

6) Al obtener el punto P_6'' se calcula el error de posición. En caso de superar la tolerancia se repetirán los mismos pasos hasta que se obtenga una solución satisfactoria o se alcance el límite de iteraciones.

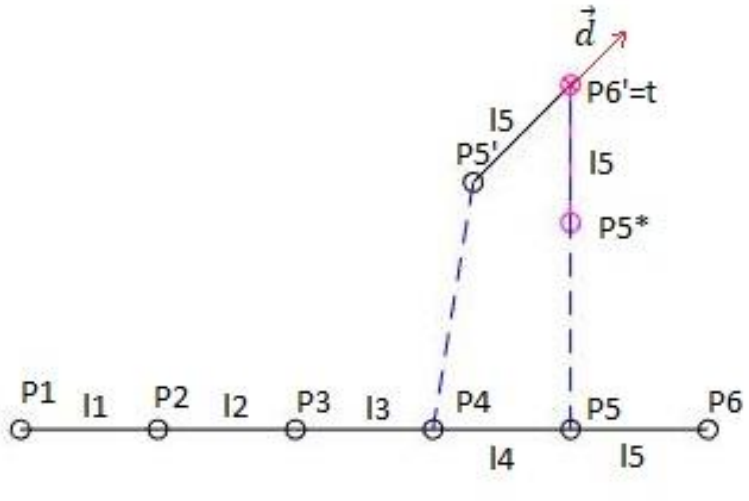


Figura 6.2 Restricción de orientación en el algoritmo FABRIK

Prueba restricción de orientación

- Número de módulos: 10
- Longitud de módulos: 2
- Punto objetivo = (7, 13, 9)
- Orientación objetivo = (-1, 1, 1) → (-1/√3, 1/√3, 1/√3)
- Tolerancia de posición = 0,1
- Tolerancia de orientación = 1°
- Límite de iteraciones: 30

Resultados: Figura 6.3-6.5

La tolerancia de orientación se ha establecido baja para asegurarnos de que se puede obtener una precisión alta, pero en el futuro se elevará para cumplir con el límite de giro que debe haber entre módulos.

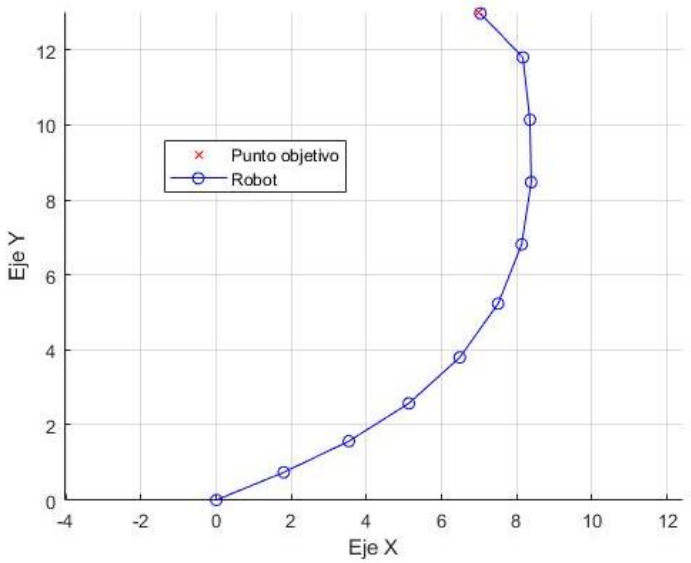


Figura 6.3 Punto objetivo P = (7, 13, 9) Orientación = (-1/√3, 1/√3, 1/√3). Vista X-Y

Al realizar la prueba se obtiene un error de posición de 0,0499 y un error de orientación de 0,8577°. Se han requerido 13 iteraciones para cumplir con las tolerancias.

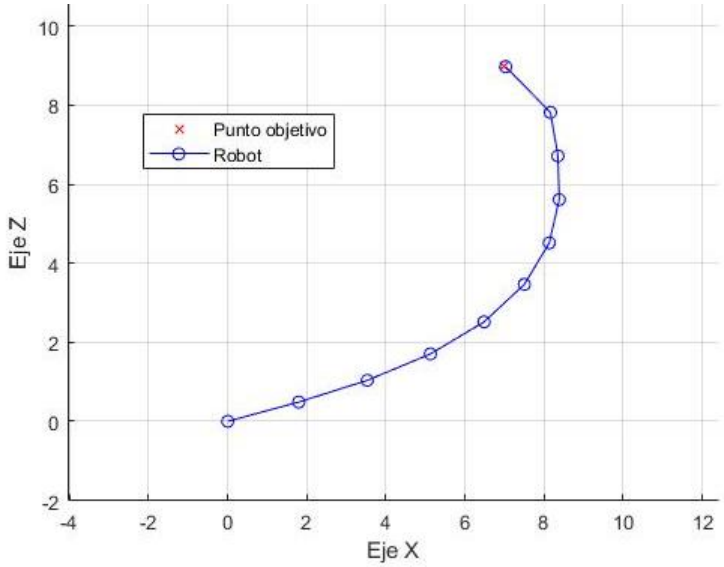


Figura 6.4 Punto objetivo P = (7, 13, 9) Orientación = (-1/√3, 1/√3, 1/√3). Vista X-Z

En la Figura 6.3 se observa como la orientación del último módulo corresponde al vector (-1, 1) siendo la primera posición la correspondiente al eje X y la segunda la del eje Y. En la Figura 6.4 se observa lo mismo que en la figura anterior con la variación de que en lugar del eje Y se tiene el eje Z. Por lo tanto, la orientación del último sí que coincide con la que se había establecido como objetivo (-1/√3, 1/√3, 1/√3). En la Figura 6.5 se tiene una vista con perspectiva del resultado obtenido.

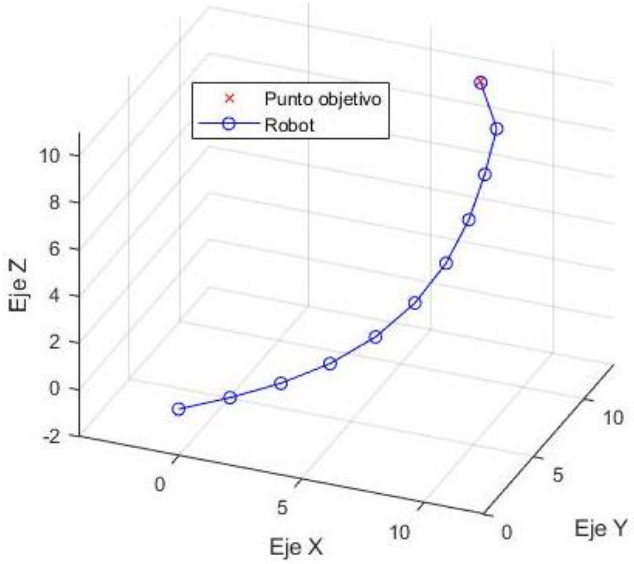


Figura 6.5 Punto objetivo P = (7, 13, 9) Orientación = (-1/√3, 1/√3, 1/√3). Vista en perspectiva

6.2 Restricción de giro

6.2.1 Actuadores no binarios

Una vez hemos comprobado que la restricción de orientación funciona correctamente pasamos a introducir la restricción de giro entre módulos. En primer lugar, introduciremos una restricción que simule actuadores no binarios, es decir, $\beta \in \forall[-\alpha, \alpha]$. Por el momento no se va a simular limitando el movimiento a los dos ejes de giro que tiene el robot. La restricción de giro se aplica en ambos sentidos, tanto en el cálculo del extremo a la base, como en el cálculo de la base al extremo. En el cálculo del extremo a la base el algoritmo contiene la restricción de orientación que fuerza la orientación del módulo de cabeza.

Para saber cuánto deben girar los módulos se ejecuta el algoritmo sin restricciones de giro y se mide el ángulo β que forman los módulos y se guarda el eje $\overrightarrow{v_r}$ respecto al que gira el módulo. Se pueden dar dos casos:

1) Si $|\beta| > \alpha$, se asigna $\beta = \alpha$
2) Si $\beta \in [-\alpha,\ \alpha]$, se mantiene el valor de β.

El cálculo de los nuevos puntos se realizará empleando cuaternios duales. Se generará un cuaternio del punto a transformar. Para calcular un punto P_i se utiliza de ayuda los puntos P_{i-1} y P_{i-2}. Estos dos puntos forman el vector $\overrightarrow{v_1}$, (Ec. 6.1).

$$\vec{v}_1 = \overrightarrow{P_{i-1}P_{i-2}} \qquad \boldsymbol{Ec.\,6.1)}$$

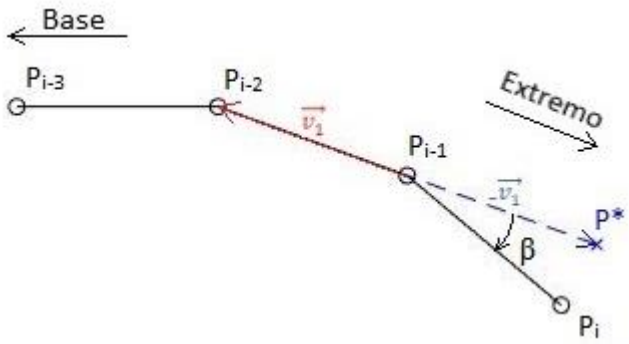


Figura 6.6 Explicación de giro con cuaternios

Debido a que todos los módulos son iguales el vector $-\overrightarrow{v_1}$ con origen en P_{i-1} coincide con el siguiente módulo en la configuración 0, es decir sin realizar ningún giro. Al punto del extremo del vector $-\overrightarrow{v_1}$ se le denominará P* (Ec. 6.2). Para obtener el punto P_i se debe rotar P* respecto al punto P_{i-1}. Por lo tanto, se creará el cuaternio dual $\widehat{DQP}$ de las coordenadas del punto P* en el sistema de referencia local de P_{i-1} (Ec.6.3).

$$-\overrightarrow{v_1} = \overrightarrow{P_{i-1}P^*} \qquad (\boldsymbol{Ec.6.2})$$

$$\widehat{DQP} = (1\ 0\ 0\ 0\ 0 - \overrightarrow{v_1}) = (1\ 0\ 0\ 0\ 0 - v_{1x} - v_{1y} - v_{1z}) \qquad (\boldsymbol{Ec.6.3})$$

También se crea el cuaternio dual de rotación $\widehat{DQR}$ (Ec. 6.4), (Ec. 6.5) empleando el ángulo β obtenido y el eje $\overrightarrow{v_r}$ respecto al que se debe rotar el punto P*.

$$\widehat{DQR} = \left(\cos\frac{\beta}{2}\ \ \overrightarrow{v_r}\cdot\sin\frac{\beta}{2}\ 0\ 0\ 0\ 0\right) \qquad (\boldsymbol{Ec.6.4})$$

$$\widehat{DQR} = \left(\cos\frac{\beta}{2}\ \ v_{rx}\cdot\sin\frac{\beta}{2}\ \ v_{ry}\cdot\sin\frac{\beta}{2}\ \ v_{rz}\cdot\sin\frac{\beta}{2}\ \ 0\ 0\ 0\ 0\right) \qquad \boldsymbol{Ec.6.5})$$

Se incluye el cuaternio dual de traslación $\widehat{DQD}$ (Ec. 6.6), pese a que inicialmente no se utilizará, pero en el futuro puede ser necesario. Para crear este cuaternio dual se necesita el vector desplazamiento $\vec{d}$.

$$\widehat{DQD} = \left(1\ 0\ 0\ 0\ 0\ \frac{\vec{d}}{2}\right) = \left(1\ 0\ 0\ 0\ 0\ \frac{d_x}{2}\ \frac{d_y}{2}\ \frac{d_z}{2}\right) \qquad (\boldsymbol{Ec.6.6})$$

El cuaternio dual de transformación $\widehat{DQ}$ (Ec. 6.7), se obtiene al multiplicar los cuaternios duales de rotación y de desplazamiento. Debido a que las configuraciones del robot no contemplan rotación y desplazamiento simultáneamente el orden en el que se multipliquen no afectará al resultado porque uno de los dos siempre será (1 0 0 0 0 0 0 0).

$$\widehat{DQ} = \widehat{DQR} \cdot \widehat{DQD} \qquad (\boldsymbol{Ec.6.7})$$

El nuevo punto se obtiene al multiplicar $\widehat{DQP}$ por el cuaternio dual de transformación $\widehat{DQ}$ y su conjugado dual $\widehat{DQ}^*$, (Ec. 6.8)

$$\widehat{DQP}_{\boldsymbol{nuevo}} = \widehat{DQ} \cdot \widehat{DQP} \cdot \widehat{DQ}^* \qquad (\boldsymbol{Ec.6.8})$$

El resultado obtenido será un cuaternio dual $\widehat{DQP}_{nuevo}$ con la forma de la (Ec. 6.9), dónde P_x, P_y y P_z son las coordenadas del nuevo punto P_i. El punto se encuentra en el sistema de coordenadas locales de P_{i-1}, para obtener las coordenadas globales se le suma el propio punto P_{i-1}.

$$\widehat{DQP}_{nuevo} = (1\ 0\ 0\ 0\ 0\ P_x\ P_y\ P_z) \qquad (\boldsymbol{Ec.6.9})$$

Prueba de restricción de giro, actuadores no binarios

- Número de módulos: 10
- Longitud de módulos: 2
- Punto objetivo = (4, 19, 0)
- Orientación objetivo = (1, 2, 0) → (1/√5, 2/√5, 0)
- Tolerancia de posición = 0,1
- Tolerancia de orientación = 10°
- Límite de iteraciones: 30
- α = 10°

Resultado: Figura 6.7

Es evidente que la solución obtenida no es satisfactoria. El **error de posición es de 3.7735** que es un 3673.5% mayor que la tolerancia establecida. El **error de orientación es 13.4349°** por lo que el error es un 34.349% mayor que la tolerancia de orientación.

Ambos errores son mayores que la tolerancia, pero es especialmente llamativo el error de posición. En la Figura 6.7 se puede observar cómo los módulos más cercanos a la base giran en dirección contraria a lo que se esperaría y debido a la restricción de giro no es posible recuperar el desplazamiento realizado y alcanzar el punto objetivo. Por ese motivo se decide estudiar la primera iteración del algoritmo para ver si se puede detectar el problema.

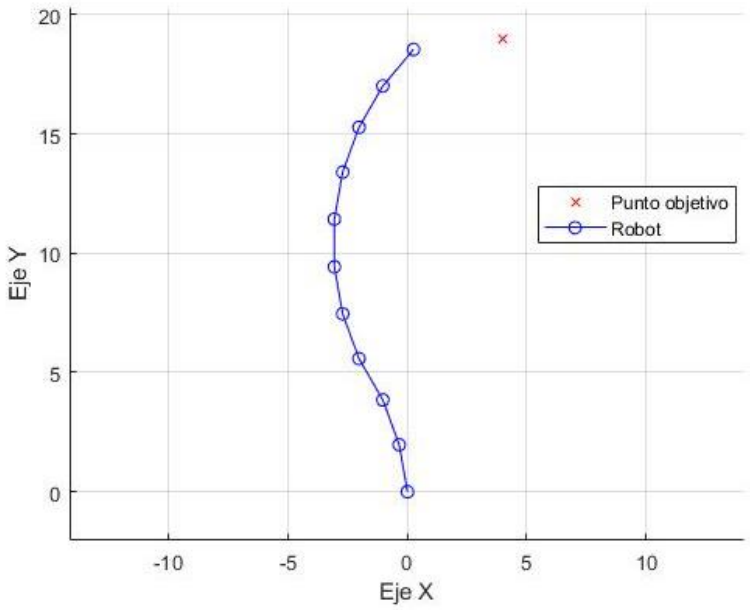


Figura 6.7 Algoritmo FABRIK con restricciones de orientación y giro. Actuadores no binarios. Punto objetivo = (4, 19, 0) Orientación objetivo = (1, 2, 0)

Para encontrar el origen de problema se decide observar la primera iteración del algoritmo, Figura 6.8. Se puede ver como en el recorrido de la base al extremo (color magenta) el punto P10' no va hacia el punto P10 debido a la orientación que hemos forzado.

A partir de ese punto se han pintado en verde las líneas que calcula el programa para calcular los siguientes puntos. Los primeros módulos giran entre sí en sentido horario respecto del punto que se calcula. Es decir, se tiene el punto P10' y el módulo gira en sentido horario para llegar al punto P9'. A partir del punto P7' comienzan a girar en sentido antihorario, pero a pesar de que todos los módulos giran en sentido antihorario el punto P1' está en el lado negativo del eje X. Esto provoca que los primeros puntos del recorrido de la base al extremo giren en sentido antihorario respecto los puntos que les preceden como se observa con las líneas amarillas. Este giro en sentido antihorario de los primeros módulos provoca que no se pueda recuperar el desplazamiento realizado y el punto P11'' quede muy alejado del punto objetivo.

- Configuración inicial en negro.
- Recorrido de extremo a base en rosa.
- Recorrido de base a extremo en azul.
- Líneas entre puntos calculadas por el programa en el recorrido de extremo a base en verde.
- Líneas entre puntos calculadas por el programa en el recorrido de base a extremo en amarillo.

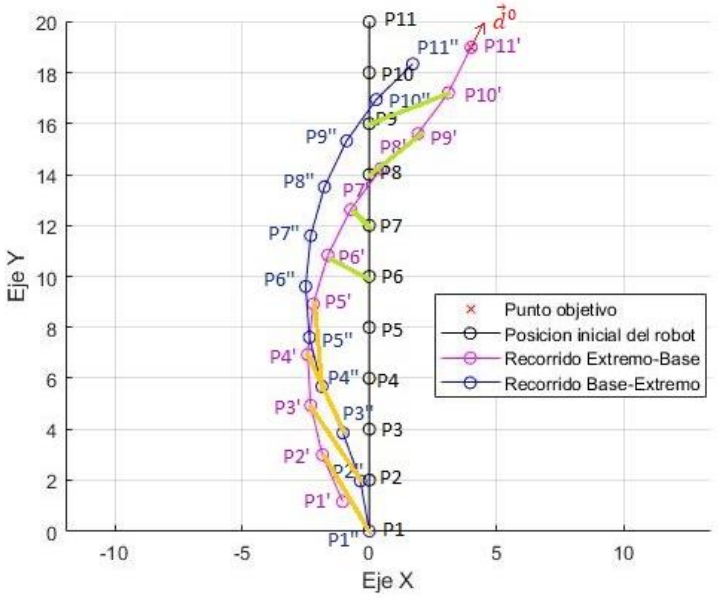


Figura 6.8 Primera iteración del algoritmo FABRIK con ambos recorridos. Punto objetivo = (4, 19, 0) Orientación objetivo = (1, 2, 0)

Para solucionar este problema se decide que **al calcular los puntos de extremo a base se continúe aplicando la restricción de orientación, pero no la de giro** eliminando de esta manera el ángulo límite que pueden girar los módulos entre sí. En el recorrido de base a extremo sí que se aplicarán las restricciones de giro, de forma que al aplicarse se guíen del resultado obtenido previamente y no se desvíen como ocurría en la Figura 6.7.

Se repite la prueba comparando el algoritmo antiguo con el algoritmo mejora.

Primera prueba de comparación de algoritmo FABRIK con y sin mejora

- Punto objetivo = (4, 19, 0)
- Orientación objetivo = (1, 2, 0) ⟶ (1/√5, 2/√5, 0)
- Tolerancia de posición = 0,1
- Tolerancia de orientación = 10°
- Límite de iteraciones: 30
- α = 10°

Resultado: Figura 6.9

El resultado de esta modificación se muestra en la Figura 6.9. Se aprecia visiblemente la mejora del resultado y los errores obtenidos confirman la mejora siendo el **error de posición 0.0188** y el **error de orientación 9.9362°** habiendo necesitado 20 iteraciones. Por lo tanto, podemos confirmar que la solución adoptada es efectiva.

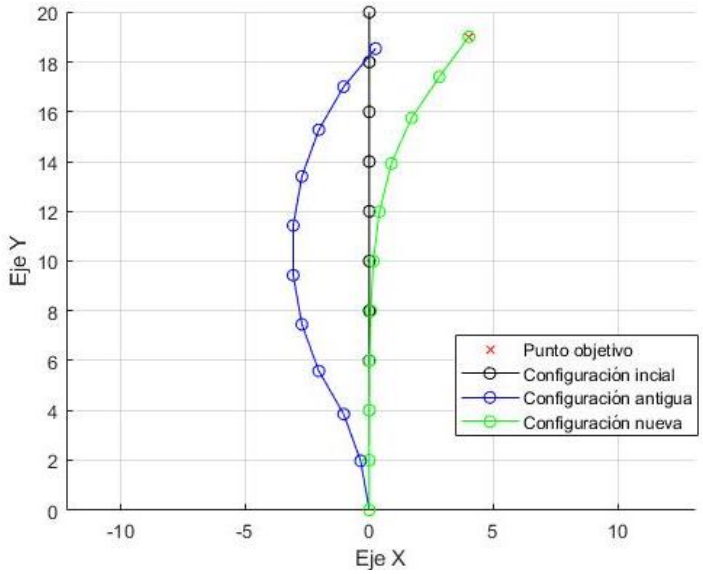


Figura 6.9 Comparación entre los algoritmos FABRIK con restricción de giro en ambos sentidos (azul) y restricción de giro solo de base a extremo (verde) Punto objetivo (4, 19, 0) Orientación objetivo = (1, 2, 0)

Se realiza otra prueba para descartar la posibilidad de que el resultado anterior haya coincidido de forma favorable a nuestro interés. Se mantienen las tolerancias, límite de giro (α) y límite de iteraciones empleados en la prueba anterior.

Segunda prueba de comparación de algoritmo FABRIK con y sin mejora

- Punto objetivo = (12, 14, 0)
- Orientación objetivo = (3,1, 0) → (3/√10, 1/√10, 0)

Resultado: Figura 6.10 y Tabla 6.1

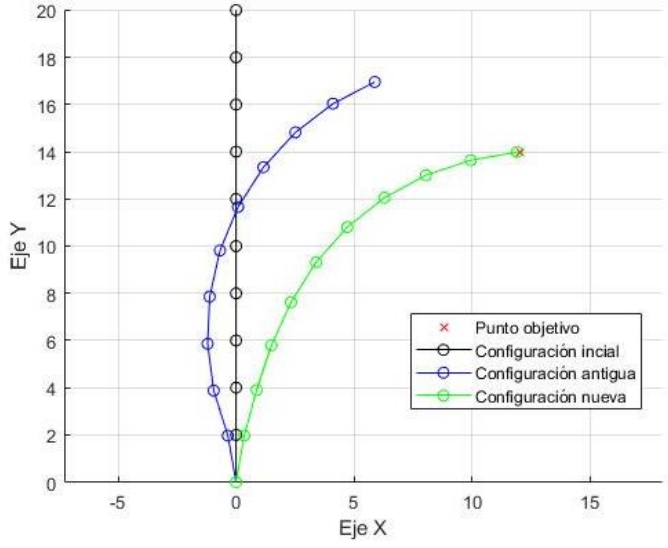


Figura 6.10 Comparación entre los algoritmos FABRIK con restricción de giro en ambos sentidos (azul) y restricción de giro solo de base a extremo (verde) Punto objetivo (12, 14, 0) Orientación objetivo = (3, 1, 0)

	Algoritmo antiguo	Algoritmo nuevo
Error posición	6,7993	0,0912
Error orientación (º)	9,0110	8,4630
Nº iteraciones	30 (límite)	14

Tabla 6.1 Resultados segunda prueba de comparación. Punto objetivo (12, 14, 0) Orientación objetivo (3, 1, 0)

Vuelve a demostrarse que la solución sí que mejora el resultado alcanzando las tolerancias objetivo. En cambio, el algoritmo sin la modificación, se repite el problema de la primera prueba y es que los módulos más cercanos a la base giran en sentido contrario a lo que se esperaría. Esto provoca que, aunque alcanza la tolerancia de orientación vuelve a tener un error de posición elevado, concretamente un 6699,3% mayor que la tolerancia establecida. Por lo tanto, podemos concluir que la solución sí funciona.

6.2.2 Actuadores binarios

Se introduce la condición de actuador binario, lo que significa que los actuadores solo pueden realizar como posición 0º ó ±α. Al igual que cuando eran actuadores no binarios se calculan los puntos aplicando el algoritmo sin restricciones de giro para obtener el ángulo β que se giraría en el caso ideal. A continuación, se comprueba:

$$\text{Si } \beta < \frac{\alpha}{2} \; Se\ asigna\ \beta = 0$$
$$\text{Si } \beta > \frac{\alpha}{2} \; Se\ asigna\ \beta = \alpha$$

Por el momento no se limitan los ejes respecto a los que pueden girar los actuadores.

Es de esperar que al reducir las posiciones que pueden tomar los actuadores la precisión obtenida disminuya. Por este motivo se comparará con dos ejemplos los algoritmos antes y después de aplicar la condición de actuador binario y se tratará de optimizar si para mejorar la precisión.

Primera prueba con actuadores binarios

- Número de módulos: 10
- Longitud de módulos: 2
- Punto objetivo = (4, 19, 0)
- Orientación objetivo = (1, 2, 0) → (1/√5, 2/√5, 0)

- Tolerancia de posición = 0,1
- Tolerancia de orientación = 10°
- Límite de iteraciones: 30
- α = 10°

Resultados: Figura 6.11 y Tabla 6.2

La Figura 6.11 muestra gráficamente el resultado obtenido y en la Tabla 6.2 podemos ver que efectivamente se ha perdido precisión tanto en la posición como en la orientación. Mientras que en el algoritmo con actuadores no binarios se cumplen ambas tolerancias sin necesitar un tercio de las iteraciones que se pueden realizar, en el algoritmo con actuadores binarios no se cumple ninguna de las dos tolerancias. El error relativo de posición es un 592,4% mayor que la tolerancia establecida. El error relativo de orientación es un 34,349% mayor que la tolerancia.

	No Binario	Binario
Error posición	0,0188	0,6924
Error orientación (º)	9,9362	13,4349
Nº Iteraciones	20	30 (límite)

Tabla 6.2 Resultados comparación algoritmo no binario-algoritmo binario Punto objetivo (4, 19, 0) Orientación objetivo (1, 2, 0)

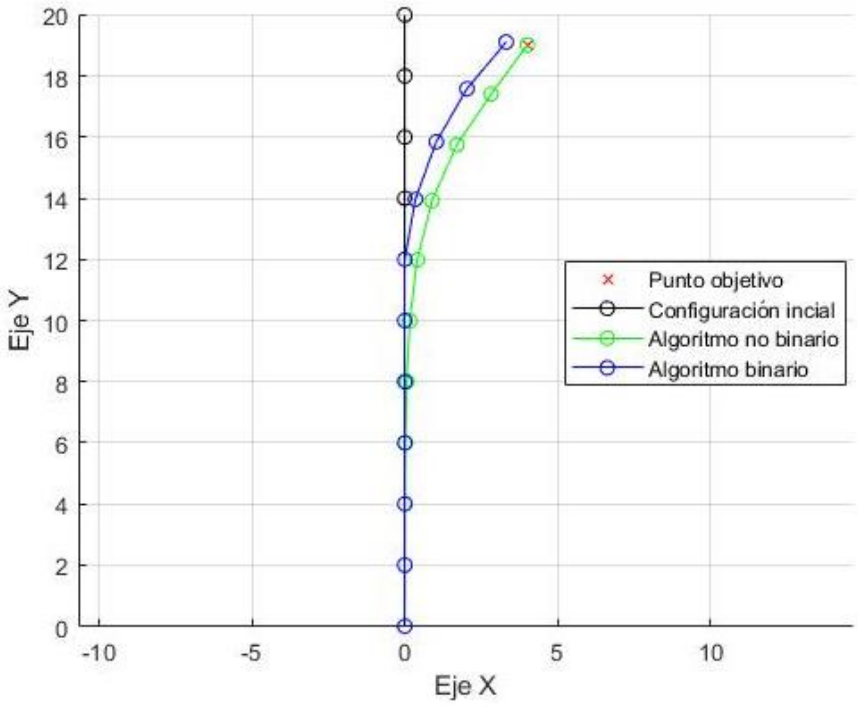


Figura 6.11 Comparación de los algoritmos FABRIK no binario (verde) con algoritmo FABRIK binario (azul). Punto objetivo (4, 19, 0) Orientación objetivo (1, 2, 0)

Segunda prueba con actuadores binarios

- Punto objetivo = (12, 14, 0)
- Orientación objetivo = (4, 1, 0) → (4/√17, 1/√17, 0)

Resultados: Figura 6.12 y Tabla 6.3

La orientación se selecciona en función de la configuración inicial del robot. Al estar inicialmente estirado a lo largo del eje Y cuando se ejecute el algoritmo los módulos próximos a la base tendrán una orientación similar a la del eje Y, por lo tanto, para alcanzar el punto objetivo los módulos del extremo deberían tener una orientación similar a la del eje X. El resto de los parámetros se mantienen igual que en la primera prueba con actuadores binarios.

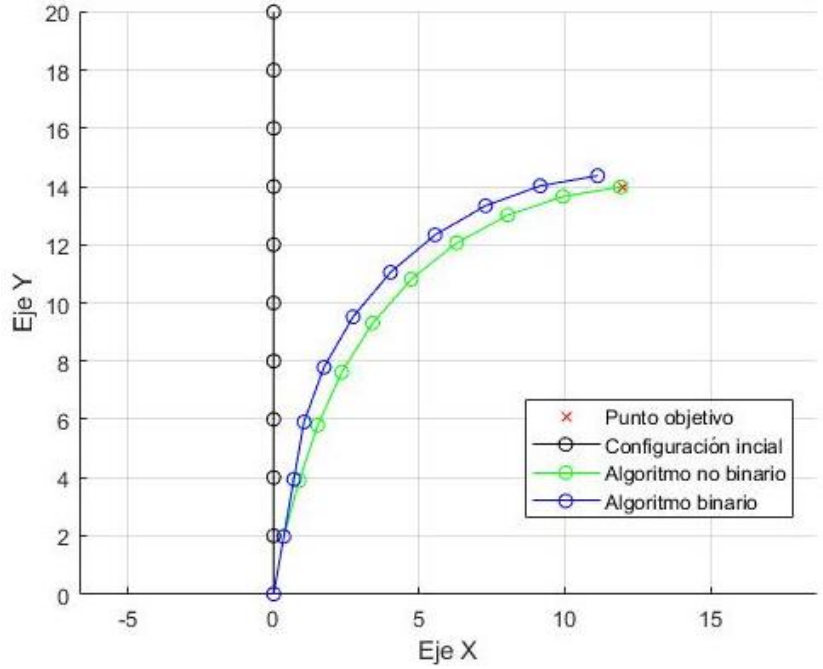


Figura 6.12 Comparación de los algoritmos FABRIK no binario (verde) con algoritmo FABRIK binario (azul). Punto objetivo (12, 14, 0) Orientación objetivo (4, 1, 0)

Al igual que en la primera prueba el algoritmo con los actuadores no binarios cumple ambas tolerancias. También se cumple la tolerancia de orientación en el algoritmo con actuadores binarios. Sin embargo, el error de posición es aún mayor que en la primera prueba, concretamente es un 850,1% mayor que la tolerancia de posición establecida.

	No Binario	Binario
Error posición	0,0939	0,9501
Error orientación (º)	4,6538	4,0362
Nº Iteraciones	15	30 (límite)

Tabla 6.3 Resultados comparación algoritmo no binario-algoritmo binario Punto objetivo (12, 14, 0) Orientación objetivo (4, 1, 0)

Tras estas dos pruebas es evidente que la precisión de la posición se ve muy afectada al introducir la condición de algoritmos binarios. No ocurre lo mismo con la precisión de orientación que en un caso es un 30% mayor que la tolerancia, pero en el otro es un 60% menor. Sin embargo, debido a la pérdida de precisión de posición se tratará a continuación de optimizar el algoritmo para que tanto la precisión de posición como la de orientación mejoren.

Para obtener mejor precisión se decide modificar el algoritmo de modo que **las 10 primeras iteraciones se aplique el algoritmo con la restricción de orientación como se aplicaba hasta ahora, pero sin restricciones de giro de ningún tipo**. De esta manera en 10 iteraciones se obtiene una configuración que servirá de guía para cuando se comiencen a aplicar las restricciones de giro. Se repetirán las dos pruebas anteriores añadiendo el nuevo algoritmo para comparar los resultados tanto visualmente como analíticamente.

Primera prueba con el algoritmo FABRIK optimizado

- Punto objetivo = (4, 19, 0)
- Orientación objetivo = (1, 2, 0) → (1/√5, 2/√5, 0)
- El resto de los parámetros permanecen igual que en la prueba anterior.

Resultados: Figura 6.13 y Tabla 6.4.

El error de orientación es el mismo que en el algoritmo binario sin optimizar. Sin embargo, el error de posición se ha reducido un 49,9%. Es evidente que las tolerancias no se cumplen, pero hay que tener en cuenta que se han establecido muy bajas para intentar alcanzar la mayor precisión posible, pero en el caso práctico no haría falta tener una tolerancia tan baja en proporción a los parámetros del robot.

	No Binario	Binario	Binario optim.
Error posición	0,0188	0,6924	0,3458
Error orientación (º)	9,9362	13,4349	13,4349
Nº Iteraciones	20	30 (límite)	30 (límite)

Tabla 6.4 Resultados comparación algoritmo no binario-algoritmo binario-algoritmo binario optimizado Punto objetivo (4, 19, 0) Orientación objetivo (1, 2, 0)

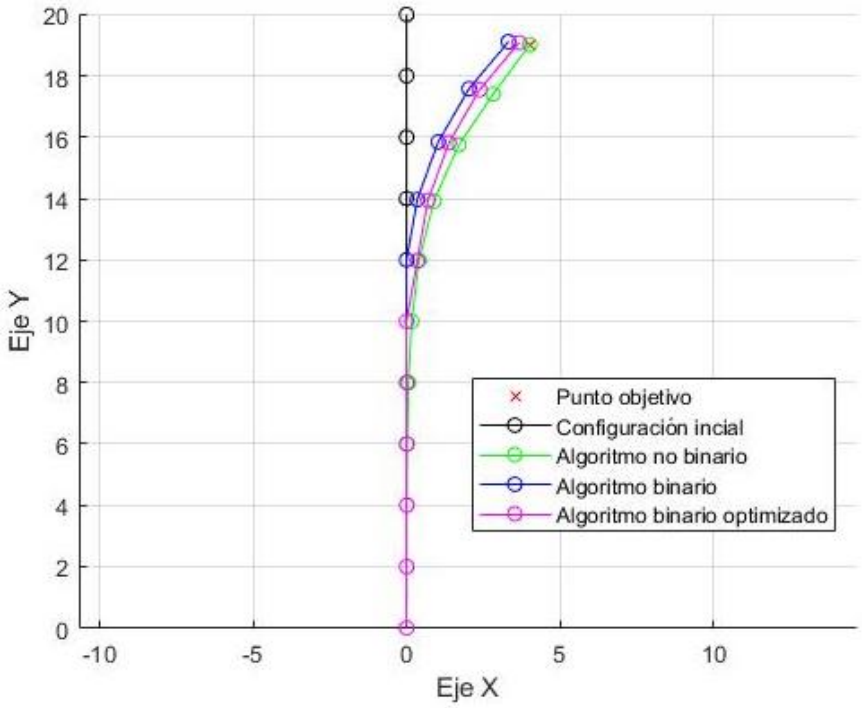


Figura 6.13 Comparación de los algoritmos FABRIK no binario (verde) con algoritmo FABRIK binario (azul). Punto objetivo (4, 19, 0) Orientación objetivo (1, 2, 0)

Segunda prueba con el algoritmo FABRIK optimizado

- Punto objetivo = (12, 14, 0)
- Orientación objetivo = (4, 1, 0) → (4/√5, 1/√5, 0)
- El resto de los parámetros permanecen igual que en la prueba anterior.

Resultados: Figura 6.14 y Tabla 6.5

Nos encontramos con una situación similar a la de la primera prueba. El error de orientación es el mismo en la versión optimizada y la no optimizada y cumple la tolerancia. El error de posición no cumple la tolerancia, pero al igual que en la primera prueba se reduce respecto al algoritmo con actuadores binarios sin optimizar, concretamente se reduce un 36,15%.

	No Binario	Binario	Binario optim.
Error posición	0,0939	0,9501	0,6066
Error orientación (º)	4,6538	4,0362	4,0362
Nº Iteraciones	15	30 (límite)	30 (límite)

Tabla 6.5 Resultados comparación algoritmo no binario-algoritmo binario-algoritmo binario optimizado Punto objetivo (12, 14, 0) Orientación objetivo (4, 1, 0)

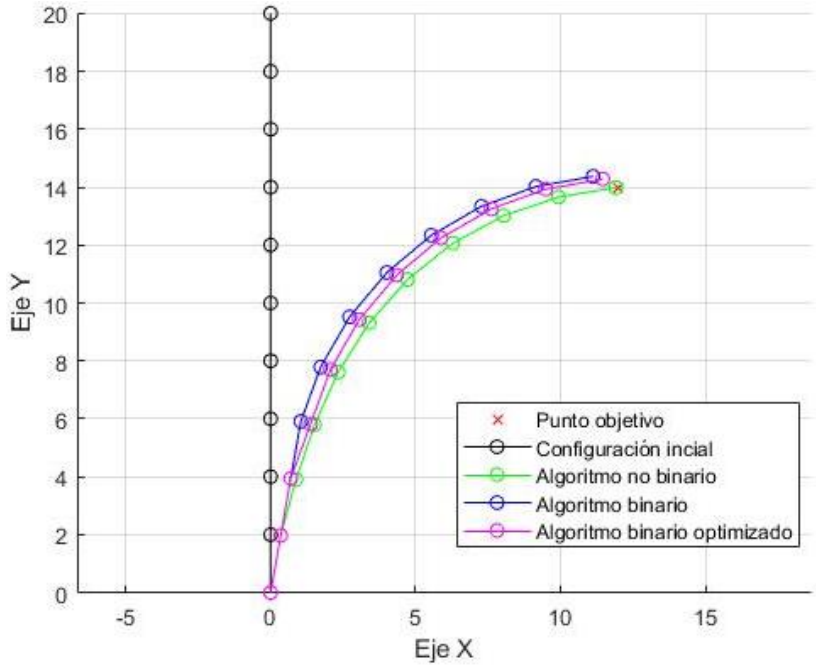


Figura 6.14 Comparación de los algoritmos FABRIK no binario (verde) con algoritmo FABRIK binario (azul). Punto objetivo (12, 14, 0) Orientación objetivo (4, 1, 0)

Tras estas dos pruebas es evidente que se ha obtenido una **mejora notable en la precisión de la posición** sin perder en la precisión de orientación que de por sí no era mala. Por lo tanto, podemos concluir que la optimización realizada es efectiva.

6.3 Restricción de los ejes de giro

Se ha introducido la condición de actuadores binarios, pero no se han limitado los ejes respecto a los que pueden girar. A continuación, se introducirá esta restricción con la que se finalizaría el proceso de optimización del algoritmo.

Se asigna un sistema de ejes locales a cada uno de los puntos del robot, excepto al último, ya que en ese no habría actuador. Al igual que cuando se estableció la condición de actuador binario, los ejes se comenzarán a utilizar a partir de la iteración 11 y solamente en el sentido de la base al extremo.

El algoritmo calcula el eje de giro ideal (caso sin restricciones de ejes) y mide el ángulo que forma el eje ideal con los dos ejes locales del punto y sus inversos (misma dirección sentido opuesto). El eje local que forme un ángulo menor se asignará como eje de giro. A partir de aquí el criterio para que el actuador gire o no es el mismo que el que se estableció al incluir la restricción de giro.

En primer lugar, se realizará una prueba en 2D, cuyo resultado se espera que sea el mismo que en el apartado anterior debido a que uno de los ejes de giro coincide con la normal al plano X-Y sobre el que se va a realizar el cálculo. La tolerancia de posición se ha aumentado respecto a los apartados anteriores debido a que se espera una reducción de la precisión al establecer La restricción de los ejes de giro.

Prueba en 2D del algoritmo FABRIK con restricciones en los ejes

- Número de módulos: 10
- Longitud de módulos: 2
- Punto objetivo = (4, 19, 0)
- Orientación objetivo = (1, 2, 0) → (1/√5, 2/√5, 0)
- Tolerancia de posición = 1
- Tolerancia de orientación = 10°
- Límite de iteraciones: 30
- α = 10°

Resultado: Figura 6.15

Se puede observar en la Figura 6.14 que solamente se ve uno de los dos algoritmos debido a que como se esperaba están superpuestos. El error de posición es 0,3458 y el error de orientación es 13,4349°.

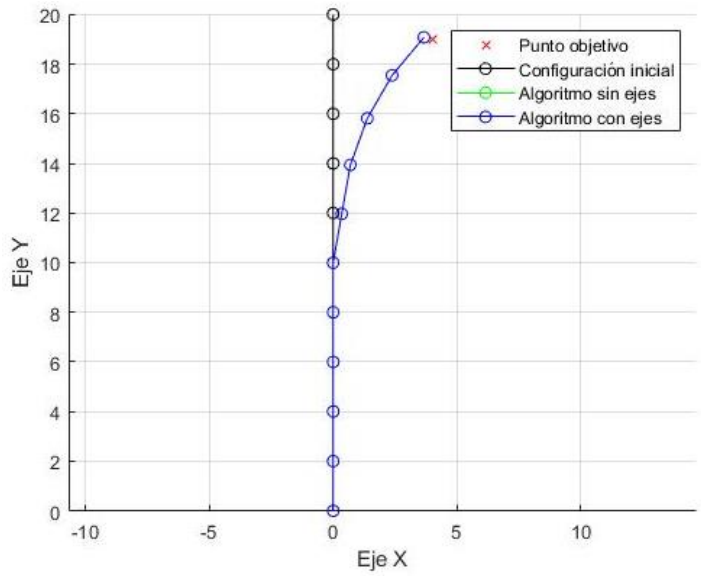


Figura 6.15 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (4, 19, 0) Orientación objetivo (1, 2, 0)

Una vez hemos comprobado que el algoritmo funciona correctamente en 2D, realizaremos tres pruebas en 3D. Se comparará además con el algoritmo sin ejes de giro para comprobar cuanta precisión se pierde. Los parámetros del robot serán los mismos que en el caso de 2D.

Primera prueba en 3D del algoritmo FABRIK con restricciones en los ejes

- Punto objetivo = (7, 18, 4)
- Orientación objetivo = (1, 1, 1) → (1/√3, 1/√3, 1/√3)
- El resto de los parámetros permanecen igual que en la prueba anterior.

Resultados: Figuras 6.16-6.18 y Tabla 6.6.

En las Figuras 6.16, 6.17 y 6.18 se muestra el resultado gráfico obtenido desde distintas perspectivas. En la Tabla 6.6 se muestran los resultados dónde podemos observar que ambos algoritmos cumplen con la tolerancia de posición siendo el error un 0,1224 mayor en el algoritmo con restricción ejes lo que supone un 26,5% mayor que el algoritmo sin restricciones. Sin embargo, aun siendo mayor el error de posición cumple la tolerancia siendo un 41,57% menor que ésta. Respecto a la tolerancia de orientación el algoritmo sin restricción de ejes no la cumple, aunque la supera por solamente un 6,4%. En cambio, el algoritmo con restricción de ejes sí que la cumple siendo un 28,56% menor que la tolerancia. Por este motivo**, el algoritmo sin ejes alcanza el límite máximo de iteraciones** mientras que el algoritmo nuevo que contiene los ejes cumple la tolerancia en el mínimo posible, ya que no debemos olvidar que se fuerza a que se realicen 10 iteraciones sin restricciones y a partir de la 11 se comienzan a aplicar.

	Algoritmo sin restricción de ejes	Algoritmo con restricción de ejes
Error posición	0,4619	0,5843
Error orientación (º)	10,6380	7,1442
Nº Iteraciones	30 (límite)	11

Tabla 6.6 Comparación algoritmo FABRIK sin restricción de ejes con algoritmo FABRIK con restricción de ejes Punto objetivo (7, 18, 4) Orientación objetivo (1, 1, 1)

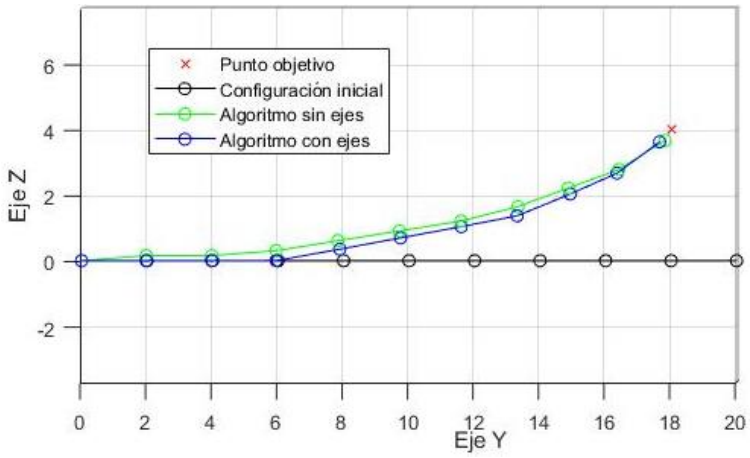


Figura 6.16 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (7, 18, 4) Orientación objetivo (1, 1, 1) Vista YZ.

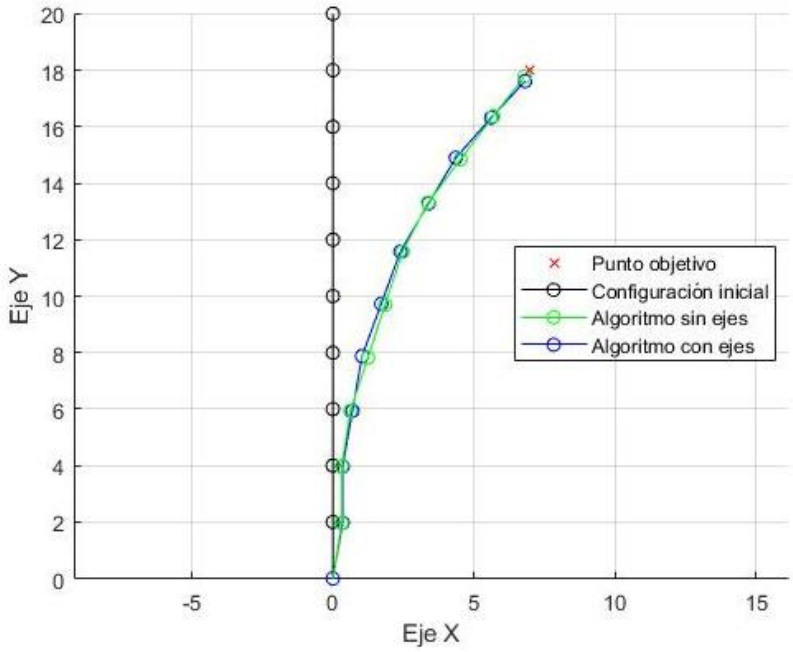


Figura 6.17 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (7, 18, 4) Orientación objetivo (1, 1, 1) Vista XY.

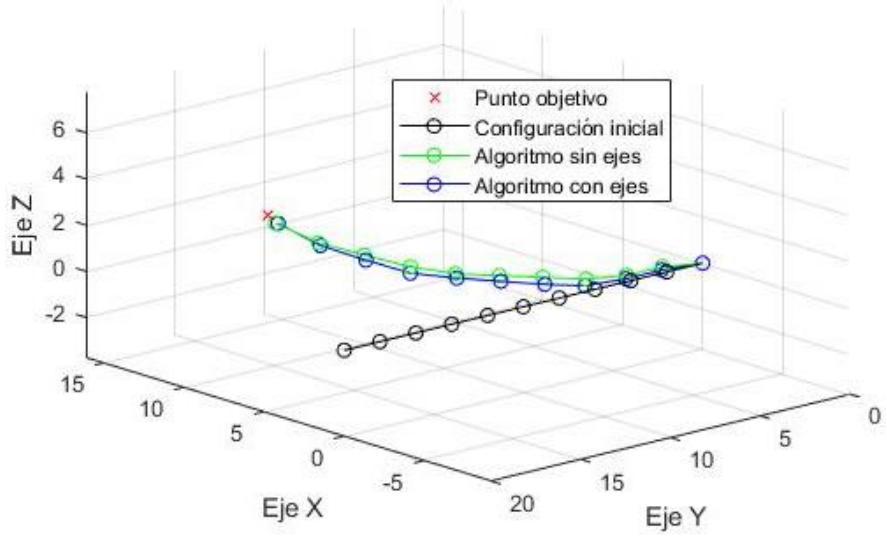


Figura 6.18 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (7, 18, 4) Orientación objetivo (1, 1, 1) Vista en perspectiva.

Segunda prueba en 3D del algoritmo FABRIK con restricciones en los ejes

- Punto objetivo = (9, 16, 2)
- Orientación objetivo = (4, 1, 1) → (4/√18, 1/√18, 1/√18)
- El resto de los parámetros permanecen igual que en la prueba anterior.

Resultados: Figuras 6.19-6.21 y Tabla 6.7.

Los resultados se muestran en las Figuras 6.18, 6.19, 6.20 y en la Tabla 6.7. En este caso ocurre al contrario que en la prueba anterior. El algoritmo sin restricción de ejes cumple ambas tolerancias necesitando el mínimo de iteraciones. En cambio, el algoritmo con los ejes supera la tolerancia de posición por un 86,86% y la de orientación por un 63,88%. Como es lógico al no cumplir las tolerancias el número de iteraciones alcanza el límite. Los errores se deben a que al no poder girar con ejes libres el algoritmo debe elegir respecto a cuál girar, lo que dependiendo de dónde se encuentre el punto objetivo puede resultar más o menos favorable para alcanzarlo.

	Algoritmo sin restricciones ejes	Algoritmo con restricción ejes
Error posición	0,0720	1,8661
Error orientación (º)	6,4889	16,3884
Nº Iteraciones	11	30 (límite)

Tabla 6.7 Comparación algoritmo FABRIK sin restricción de ejes con algoritmo FABRIK con restricción de ejes Punto objetivo (9, 16, 2) Orientación objetivo (4, 1, 1)

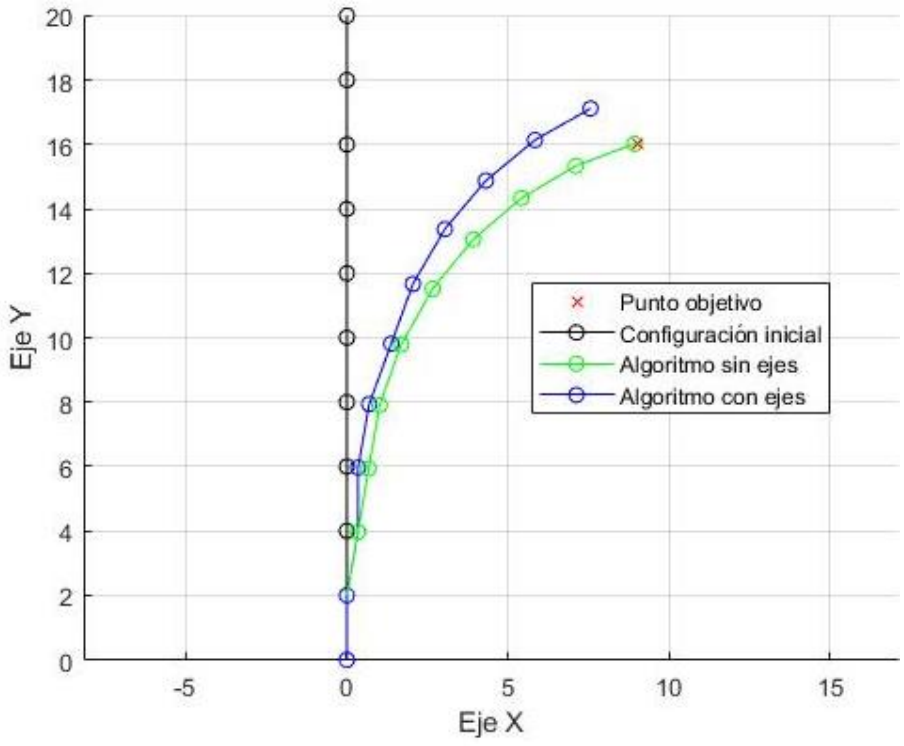


Figura 6.19 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (9, 16, 2) Orientación objetivo (4, 1, 1) Vista XY.

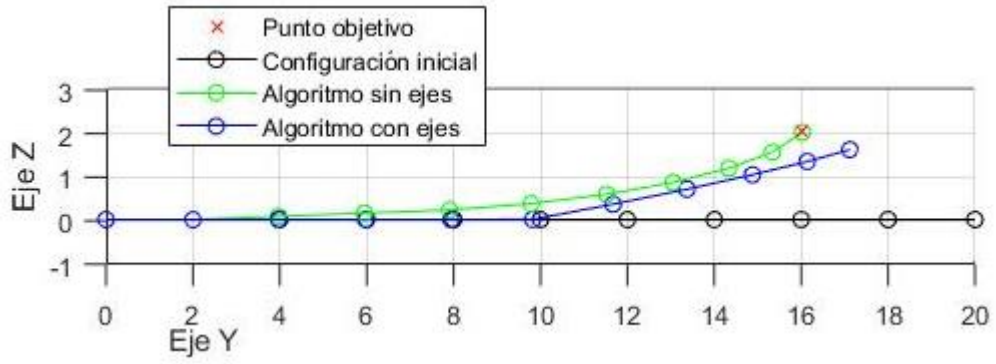


Figura 6.20 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (9, 16, 2) Orientación objetivo (4, 1, 1) Vista YZ.

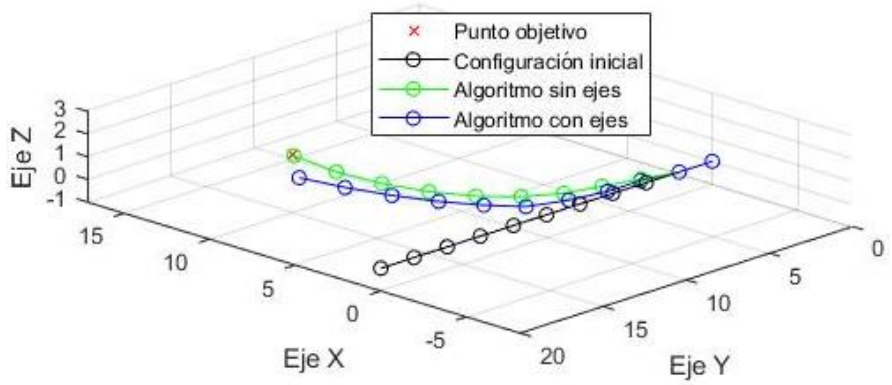


Figura 6.21 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (9, 16, 2) Orientación objetivo (4, 1, 1) Vista en perspectiva.

Tercera prueba en 3D del algoritmo FABRIK con restricciones en los ejes

- Punto objetivo = (3, 19, 4)
- Orientación objetivo = (1, 1, 1) → (1/√11, 3/√11, 1/√11)
- El resto de los parámetros permanecen igual que en la prueba anterior.

Resultados: Figuras 6.22-6.24 y Tabla 6.8.

Los resultados se muestran en las Figuras 6.22, 6.23, 6.24 y la Tabla 6.8. Al igual que en la prueba anterior el algoritmo sin las restricciones de los ejes cumple las tolerancias en el mínimo de iteraciones. En este caso, el algoritmo con restricción de ejes sí que cumple la tolerancia de posición siendo el error obtenido un 28,71% menor que la tolerancia establecida. El error de orientación es un 19,78% mayor que la tolerancia. Por este motivo se alcanza el límite de iteraciones.

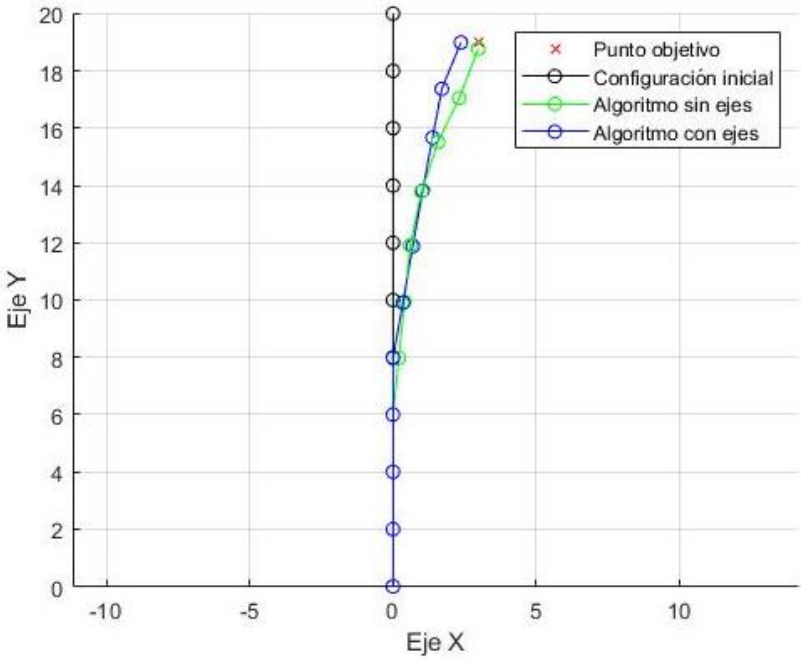


Figura 6.22 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (3, 19, 4) Orientación objetivo (1, 3, 1) Vista XY.

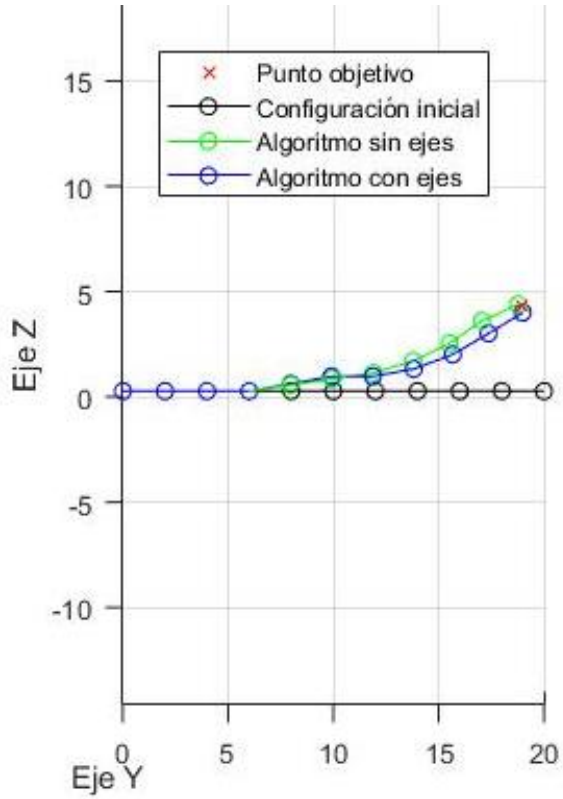


Figura 6.23 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (3, 19, 4) Orientación objetivo (1, 3, 1) Vista YZ.

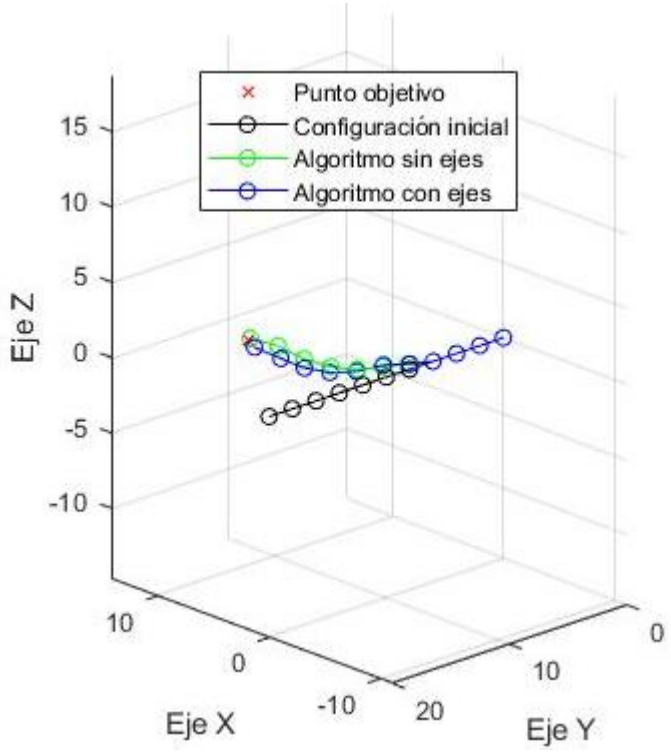


Figura 6.24 Comparación algoritmo FABRIK sin restricción de ejes (verde) con algoritmo FABRIK con restricción de ejes (azul) Punto objetivo (3, 19, 4) Orientación objetivo (1, 3, 1) Vista en perspectiva.

	Algoritmo sin ejes	Algoritmo con ejes
Error posición	0,2308	0,7129
Error orientación (º)	5,6353	11,9778
Nº Iteraciones	11	30 (límite)

Tabla 6.8 Comparación algoritmo FABRIK sin restricción de ejes con algoritmo FABRIK con restricción de ejes Punto objetivo (3, 19, 4) Orientación objetivo (1, 3, 1)

Tras las pruebas realizadas podemos concluir que a pesar de que se ha disminuido la precisión, **la restricción del sistema de ejes funciona correctamente** limitando a solo dos ejes de giro para cada módulo. Además, **los errores son elevados respecto a las tolerancias establecidas que se fijaron bajas precisamente para obligar al algoritmo a ejecutar un número elevado de iteraciones y comprobar su precisión**. Por lo tanto, la optimización estaría completada y se podría comenzar con las pruebas de seguimiento de trayectoria, la cuales son nuestro objetivo final.

CAPÍTULO 7

7. Seguimiento de trayectorias

Para realizar el seguimiento de trayectorias se ha diseñado un nuevo programa en el que se ejecutará la función en la que se encuentra el algoritmo FABRIK mejorado del apartado anterior. En este programa se introducen inicialmente los diferentes parámetros del robot: número de módulos, longitud de módulos y ángulo α. También se definen las tolerancias de posición y orientación, el número máximo de iteraciones para cada posición y el avance, que es la distancia que avanza la base del robot entre punto y punto.

En el mismo programa también se dará a elegir 4 tipos de trayectorias a seguir: línea recta, curva, espiral y un intestino grueso simplificado. Según cual se seleccione se tendrán que introducir los diferentes parámetros para definir la trayectoria. Con los parámetros introducidos se generan tres vectores. Cada vector contiene una de las componentes para definir los puntos de la trayectoria, de modo que, si tomamos la primera componente del vector 'xi', el vector 'yi' y el vector 'zi' obtendríamos las coordenadas del primer punto de la trayectoria.

Para realizar el seguimiento de trayectoria se utilizarán 5 módulos sobre los que se aplicará el algoritmo. En el caso real, el resto de los módulos seguirían a los 5 primeros mediante un algoritmo 'Follow The Leader'. La longitud de los módulos será 10 mm, el radio 5mm y el ángulo α será 10º. Estos parámetros se han estimado en función a las características de los endoscopios y las limitaciones que presentan los actuadores.

La tolerancia de posición será 8 mm y la de orientación 20º. Se ha establecido 8mm como tolerancia de posición debido a que el intestino grueso en su parte más estrecha (recto) tiene un diámetro de 30 mm, por lo tanto, teniendo en cuenta que nuestro robot tiene un diámetro de 10mm aproximadamente, si se cumple la tolerancia, el margen de seguridad será de 2 mm a cada lado en el punto más crítico. La tolerancia de orientación se ha establecido relativamente alta porque el objetivo principal cuando el robot recorre el intestino es que no se aleje de la trayectoria marcada para evitar tocar las paredes siendo la orientación del último módulo algo secundario. Todas las trayectorias se iniciarán en el punto (0, 0, 0), dónde se encontrará inicialmente la cabeza del robot.

Se comenzará a emplear el error global de los módulos cuyas referencias serán las tolerancias de posición y de orientación. Se considera que el punto importante de cada módulo es el más próximo al extremo. Por este motivo se agruparán en parejas los puntos y módulos de extremo a base quedando el primer punto (base) solo. Debido a que se le da más importancia a la posición que a la orientación el factor multiplicador del error de posición será 0,7 y el factor del error de orientación 0,3.

7.1 Resultados obtenidos del seguimiento de trayectorias

Durante la simulación se representa el robot en rojo. Los puntos representan la base de cada módulo. Las circunferencias representan la orientación de los módulos y se encontrarían en el extremo del módulo (donde se encuentra el actuador). Por lo tanto, las circunferencias no están asociadas con el punto que se encuentra en su centro, que pertenecería al siguiente módulo.

7.1.1 Trayectoria recta

En primer lugar, se seguirá la trayectoria recta. Simplemente se realiza para comprobar que el programa funciona correctamente y no hay fallos. Al ser una trayectoria recta los actuadores se mantendrán en la configuración 0. La trayectoria a seguir será una recta de 70 mm de longitud con origen en el punto (0, 0, 0) que se extiende a lo largo del eje x en sentido positivo. La distancia entre puntos será la misma que el avance. La Figura 7.1 muestra la situación final tras recorrer la trayectoria. En las Figuras 7.2, 7.3 y 7.4 se muestran los errores de posición de los puntos y de orientación de los módulos respectivamente.

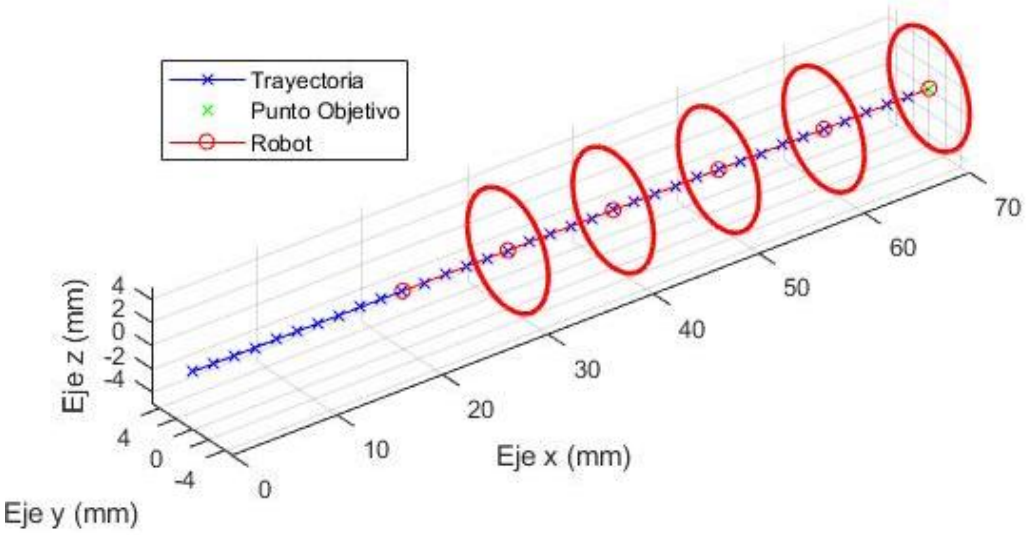


Figura 7.1 Resultado de seguimiento de una trayectoria recta de 70mm a lo largo del eje X

Como se esperaba los errores de posición y orientación son nulos en todas las iteraciones y en consecuencia el error global también lo es. Esto se debe a que al ser una trayectoria recta los actuadores no deben cambiar la posición porque la configuración inicial es válida para toda la trayectoria.

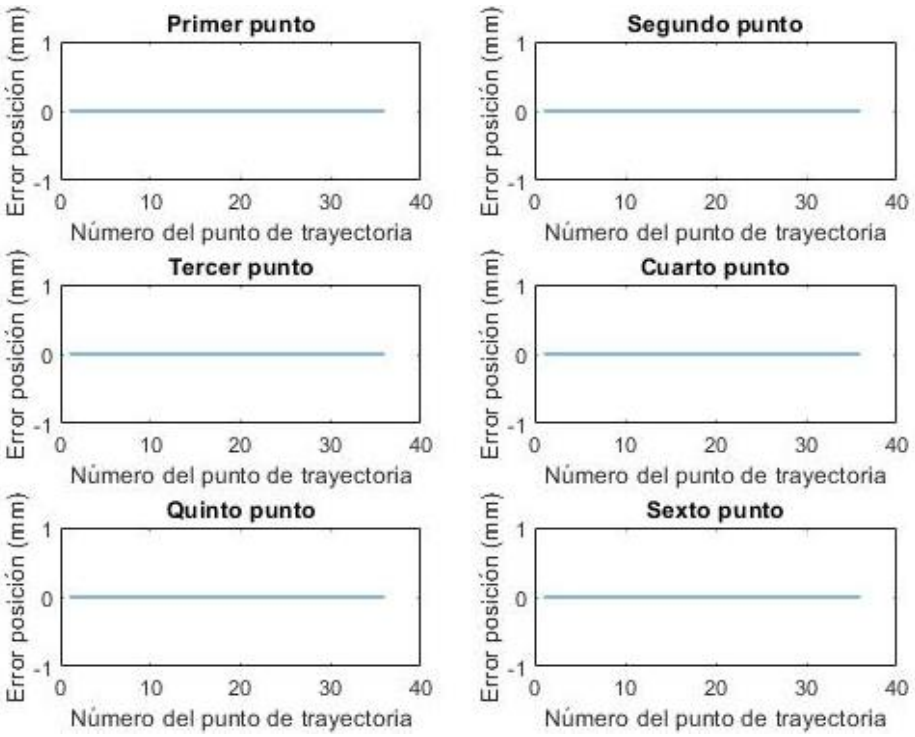


Figura 7.2 Errores de posición de los puntos en una trayectoria recta de 70mm a lo largo del eje X

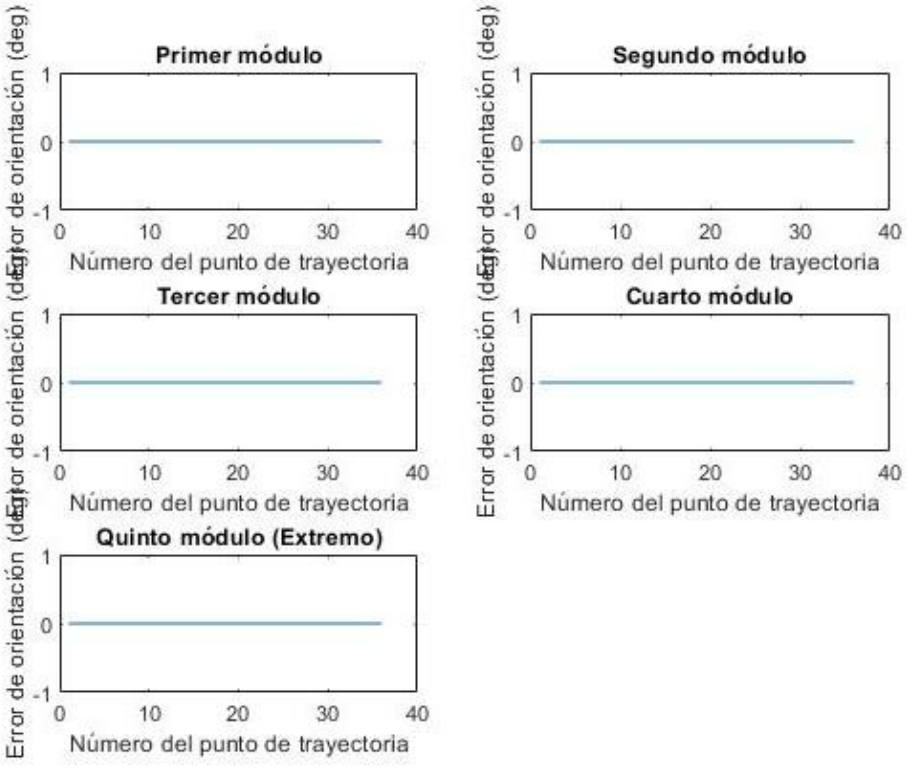


Figura 7.3 Errores de orientación de los módulos en una trayectoria recta de 70mm a lo largo del eje X

No se muestra la gráfica del error global porque evidentemente si los errores de posición y orientación son nulos también lo será el error global y, por lo tanto, la gráfica no aportaría ninguna información adicional.

7.1.2 Trayectorias en forma de curva

Una vez se ha comprobado que el programa creado funciona correctamente se tratará de seguir una trayectoria en 2D. Hay que tener en cuenta que debido a las limitaciones físicas del robot hay un radio mínimo para la curva a partir del cual si se disminuye más el radio el robot no sería capaz de seguir la trayectoria. Este radio viene determinado por la (Ec. 7.1) donde α es el ángulo que pueden girar los actuadores y se muestra una representación gráfica en la Figura 7.4 [Tappe, S. 2015].

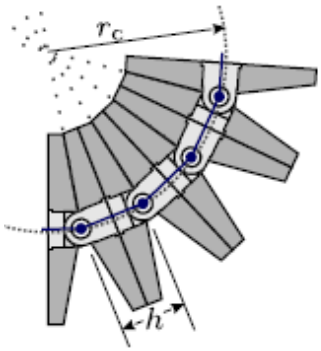


Figura 7.4 Representación gráfica del radio mínimo de giro

$$r_c = \frac{h}{2\tan\frac{\alpha}{2}} \qquad (\boldsymbol{Ec.7.1})$$

Para los parámetros del robot que se han definido, el radio de giro mínimo es 57,15 mm. Conociendo el radio mínimo de giro podemos realizar la simulación de una trayectoria curva en 2D.

- Trayectoria curva 1

La curva tendrá un radio de giro de 70 mm, se girará 90° y tanto previa como posteriormente a la curva habrá una recta de 30 mm de longitud.

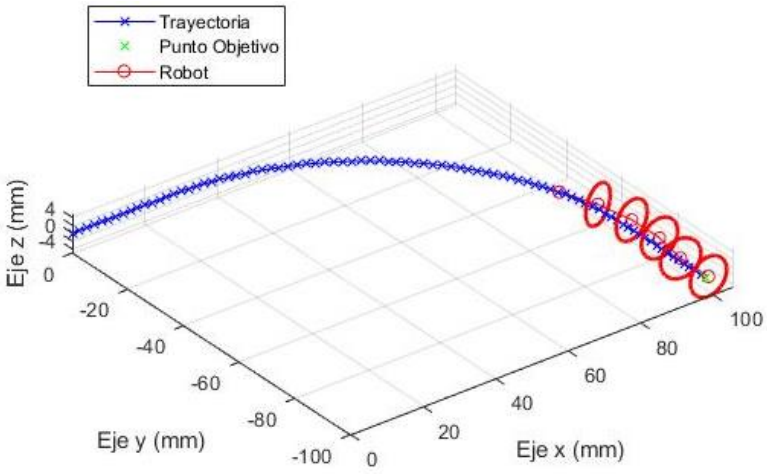


Figura 7.5 Resultado de seguimiento de una trayectoria curva de radio 70mm y giro de 90°

En la Figura 7.5 se muestra el resultado obtenido al final el recorrido donde se puede observar que el robot sí alcanza el punto final manteniéndose correctamente en la trayectoria. En las Figuras 7.6, 7.7 y 7.8 se muestran los errores de posición, orientación y global.

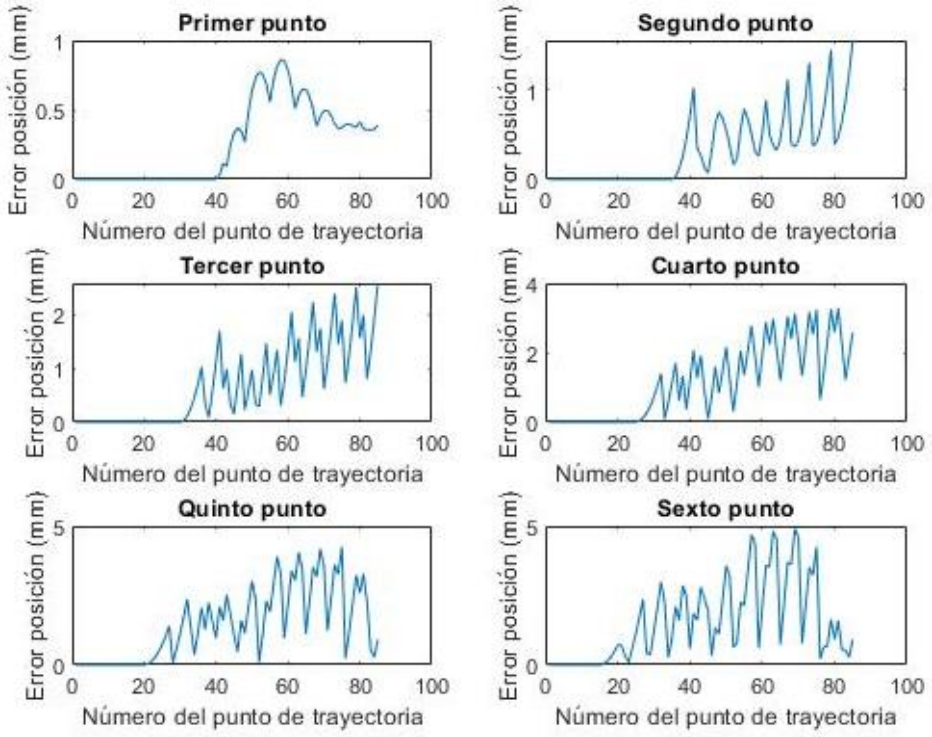


Figura 7.6 Errores de posición de los puntos en una trayectoria curva de radio 70mm y giro de 90º

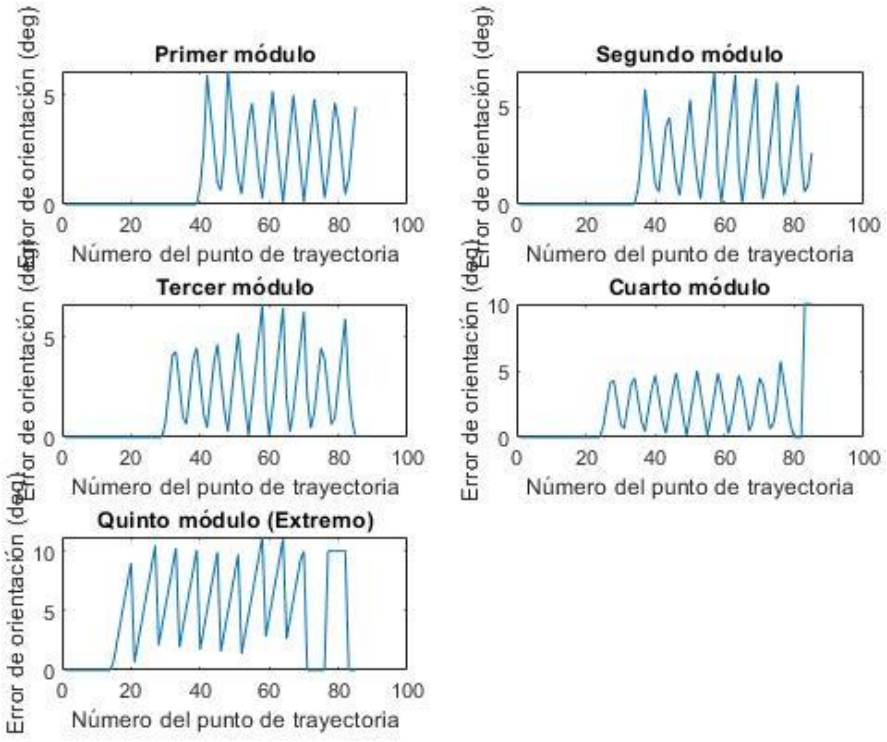


Figura 7.7 Errores de orientación de los módulos en una trayectoria curva de radio 70mm y giro de 90º

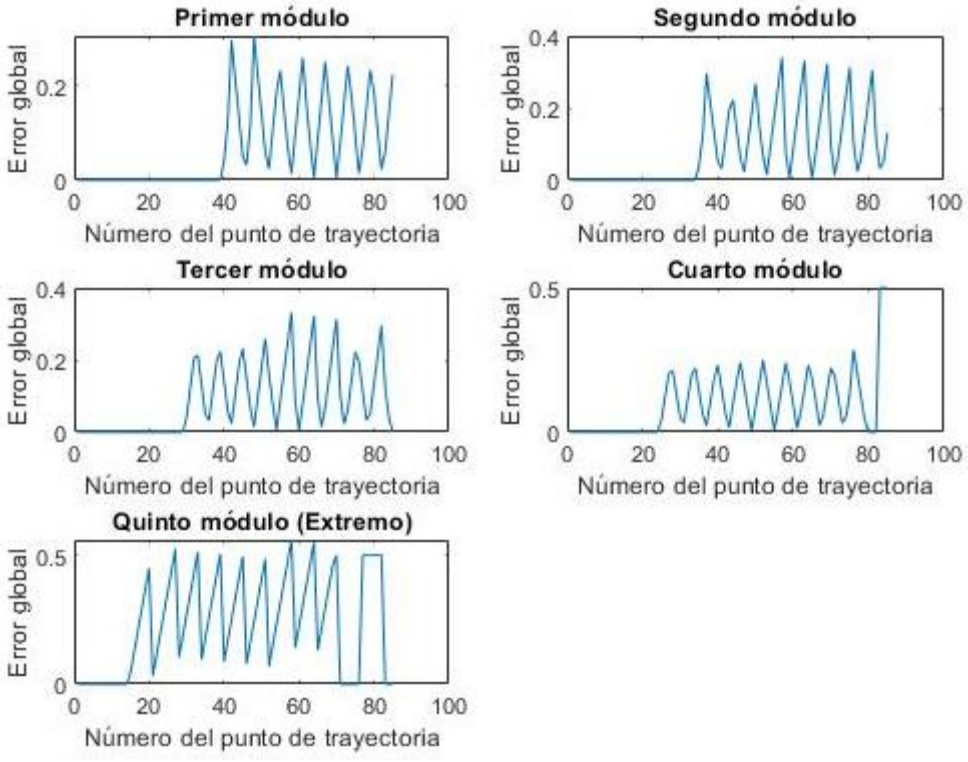


Figura 7.8 Errores globales de los módulos en una trayectoria curva de radio 70mm y giro de 90°

Los errores de posición se mantienen por debajo de la tolerancia en todos los puntos del robot. Como es de esperar cuanto más nos aproximamos al extremo del robot mayor es el error debido a que se van acumulando los pequeños errores de los puntos anteriores. Los errores de orientación también se mantienen por debajo de la tolerancia para todos los módulos. En la gráfica de los errores de posición se observa que dependiendo de cuál sea el actuador que cambie el error disminuye en mayor o menor cantidad. Los errores del último punto son de 0,9099 mm y 0°. Evidentemente al mantenerse ambos errores por debajo de la tolerancia en todo momento los errores globales son en todo momento inferiores a 1.

- Trayectoria curva 2

La segunda trayectoria curva será un giro de 360° de radio 70 mm. El objetivo es observar el comportamiento del robot al realizar un giro sostenido durante un periodo largo. Las rectas de entrada y salida de la curva medirán 30 mm ambas.

La Figura 7.9 muestra la trayectoria seguida y la posición del robot al finalizar el recorrido. El giro se realiza en en sentido horario por lo tanto el punto final de la trayectoria es el (60, 0, 0) y a simple vista parece que se alcanza correctamente. En las Figuras 7.10, 7.11 y 7.12 se muestran los errores.

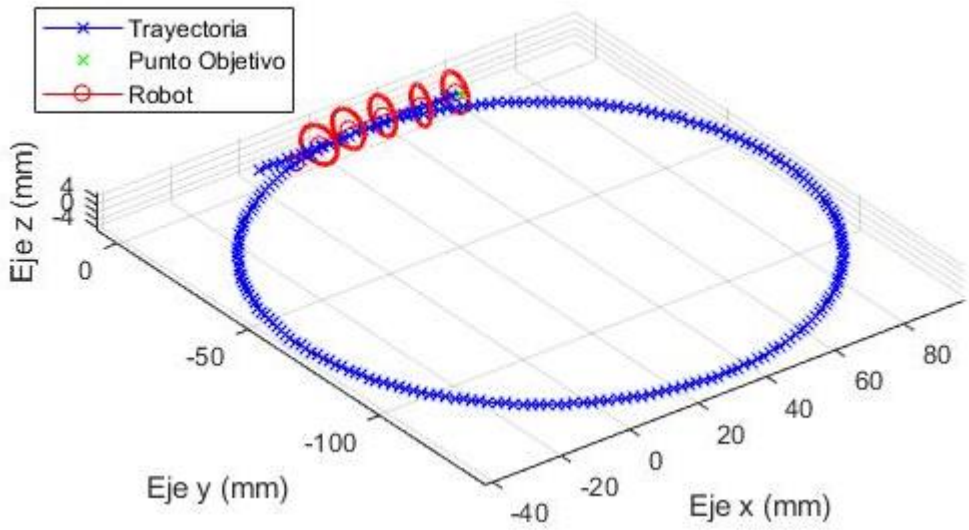


Figura 7.9 Resultado de seguimiento de una trayectoria curva de radio 70mm y giro de 360º

Al igual que en la prueba del giro de 90º, en la prueba de 360º se cumplen tanto las tolerancias de posición como las de orientación. El error máximo de posición tiene un valor de 4,995 mm y tiene lugar en el último punto del robot al alcanzar el punto 124 de la trayectoria. Por otro lado, los errores de orientación se mantienen alejados de la tolerancia marcada rondando el error máximo los 10º en el último módulo. Vuelve a repetirse el patrón de dientes de sierra y la acumulación de los errores de posición de los primeros puntos hacia los últimos. Los errores en el último punto de la trayectoria son de 0,9813 mm y 0º.

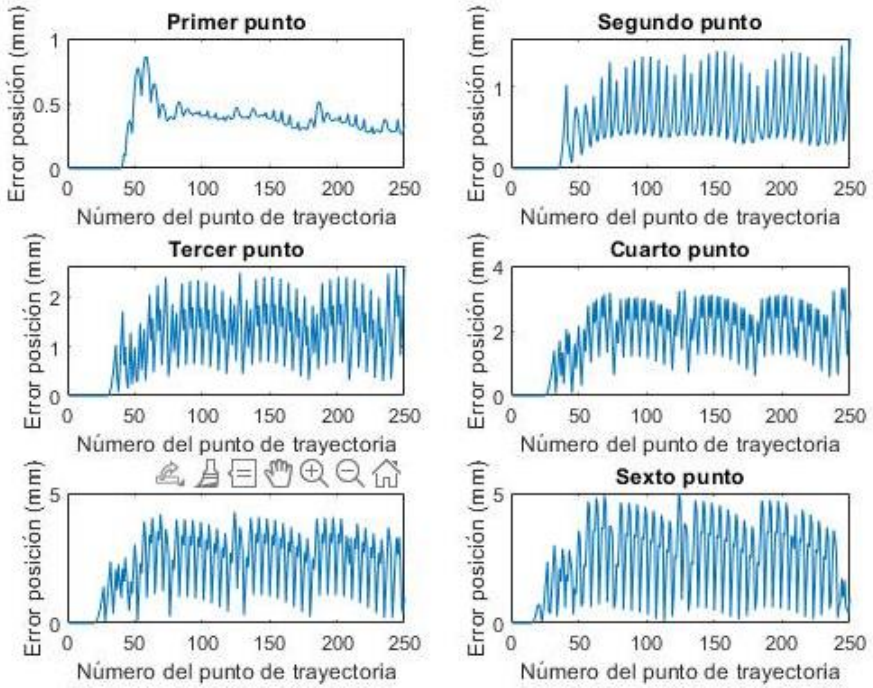


Figura 7.10 Errores de posición de los puntos en una trayectoria curva de radio 70mm y giro de 360º

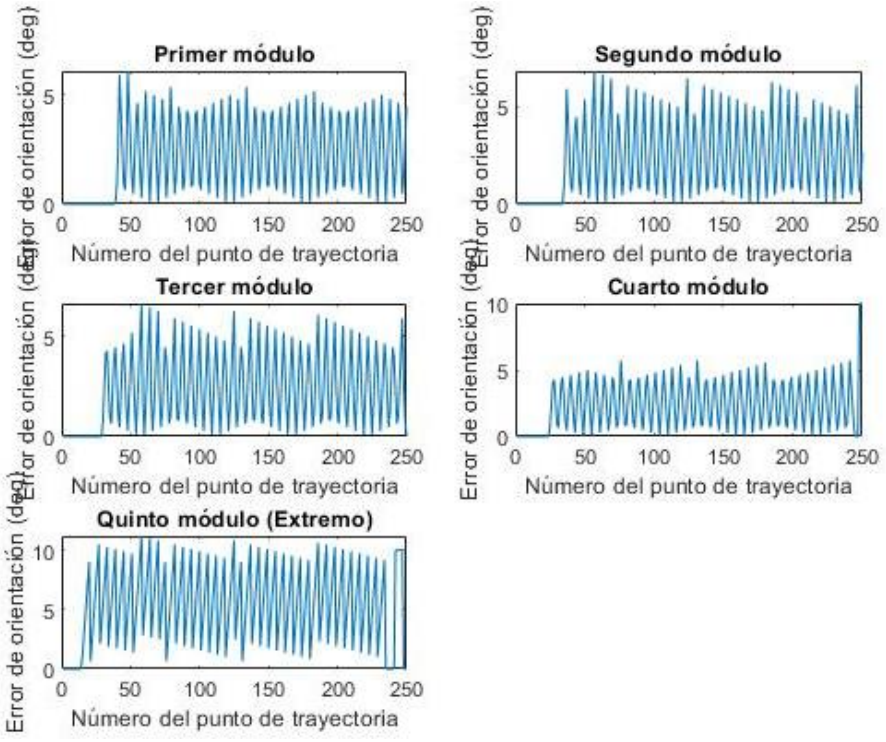


Figura 7.11 Errores de orientación de los módulos en una trayectoria curva de radio 70mm y giro de 360º

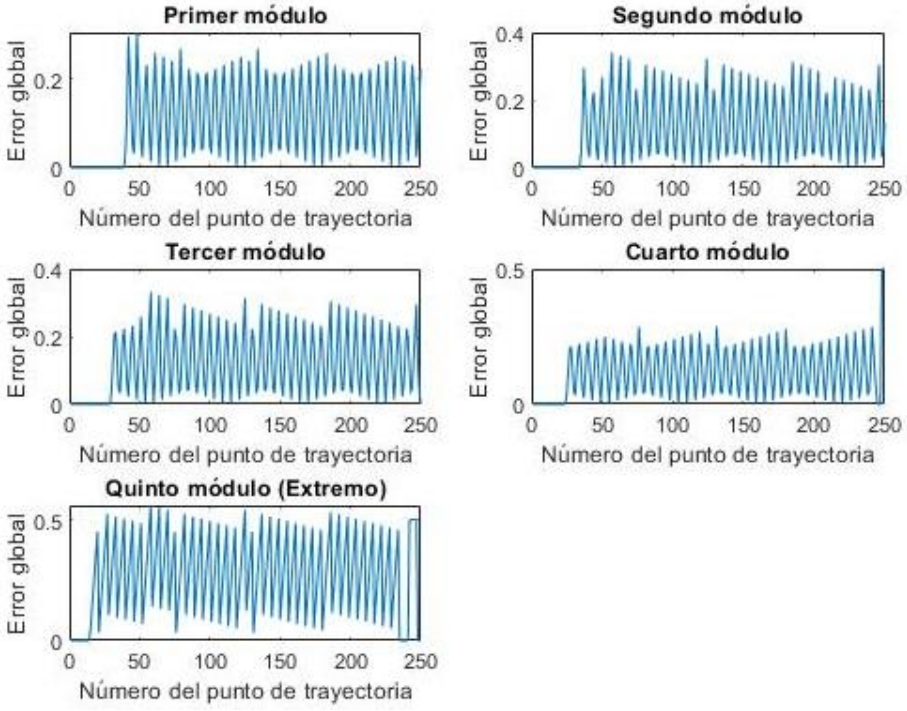


Figura 7.12 Errores globales de los módulos en una trayectoria curva de radio 70mm y giro de 360º

Estas dos pruebas muestran que el algoritmo funciona correctamente al realizar un giro de radio próximo al límite físico del robot. El robot es capaz de seguir la trayectoria cumpliendo las tolerancias en todo momento. Además, se alcanza una precisión alta en el último punto al llegar al final de la trayectoria.

7.1.3 Trayectoria en espiral

Aunque el caso del intestino grueso sería seguir una trayectoria casi sobre un plano resulta interesante que el algoritmo funcione correctamente en casos tridimensionales. Para realizar las pruebas en 3D se tratará de seguir una trayectoria con forma de espiral.

La trayectoria a seguir es una espiral de radio 80 mm, paso 40 mm y contará con dos espiras, Figura 7.13. Se ha aumentado en 10 mm el radio respecto a las curvas de las pruebas anteriores debido a que al tener que decidir si los actuadores hacen un giro con el objetivo de ascender o de girar lateralmente la capacidad de giro se ve reducida y por lo tanto el radio de giro debe ser mayor.

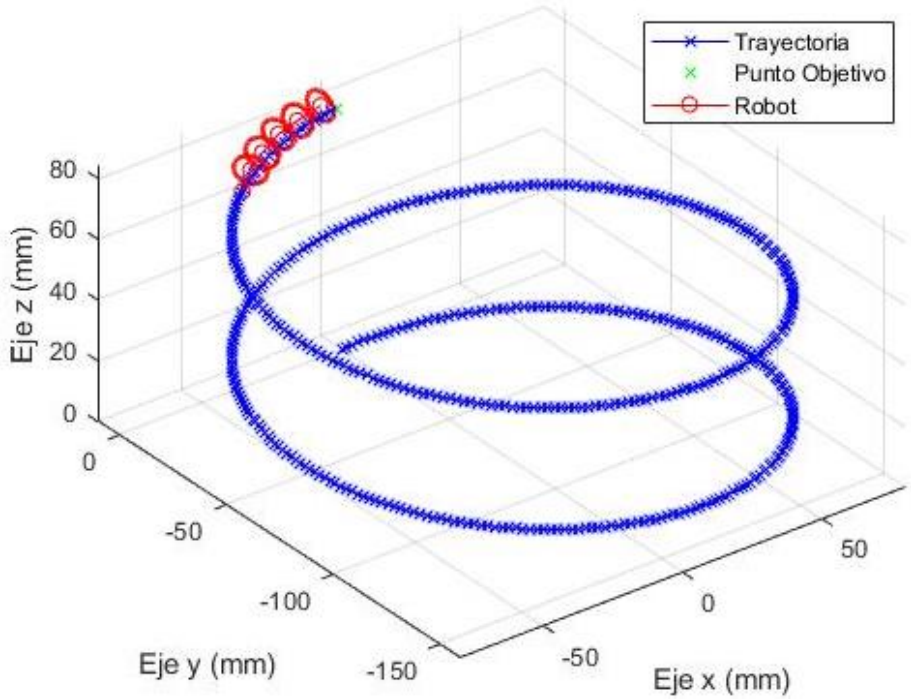


Figura 7.13 Resultado de seguimiento de una trayectoria espiral de radio 80mm, paso 40 mm y 2 espiras

Las Figuras 7.14, 7.15 y 7.16 muestran los errores. En este caso el error de posición sí que supera la tolerancia. En el quinto punto se supera en dos ocasiones con valores de 5,064 y 5,101 mm. En el último punto el error de posición supera la tolerancia en alguna ocasión al inicio de la trayectoria hasta convertirse en un error casi constante en torno a 5,1 y 5,2 mm con algún pico que supera los 6 mm siendo el error máximo de posición de 6,596 mm en el punto 325 de la trayectoria. Respecto a la orientación sí que se cumple la tolerancia. Los valores se mantienen alrededor de los 10° y se alcanza un valor máximo de 17,94° también en el punto 325 de la trayectoria. Los errores del último punto son 5,247 mm y 9,108°. Son valores similares a los que se han reproducido en el resto de los puntos de la trayectoria. Es

evidente que los errores de posición se encuentran ligeramente por encima del valor de tolerancia establecido, sin embargo, esto se debe a las limitaciones físicas del robot y no al funcionamiento del algoritmo. Una vez más al no superarse las tolerancias los errores globales serán inferiores a 1.

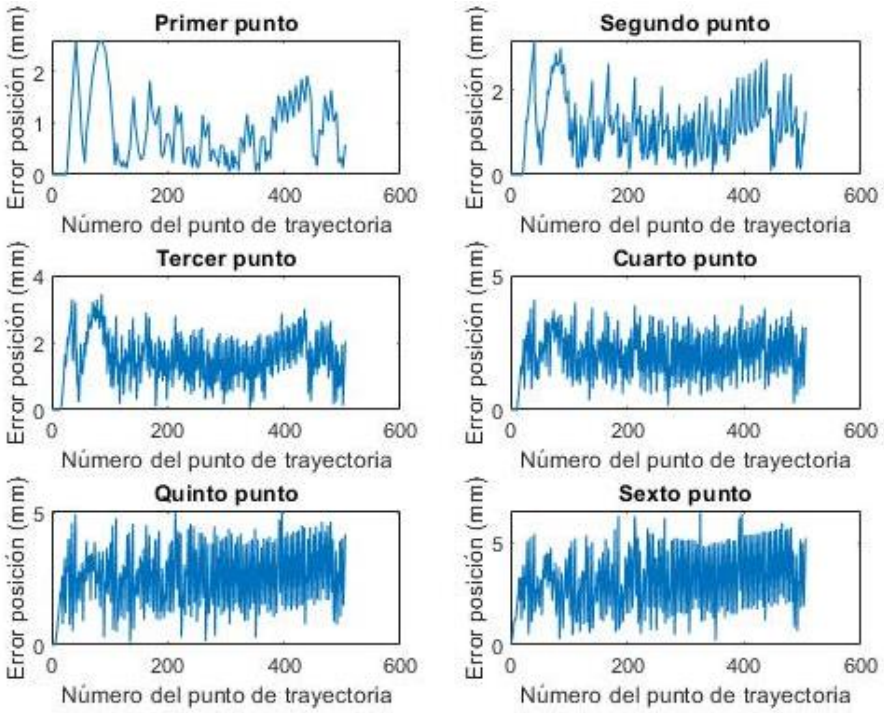


Figura 7.14 Errores de posición de los puntos en una trayectoria espiral de radio 80mm, paso 40 mm y 2 espiras

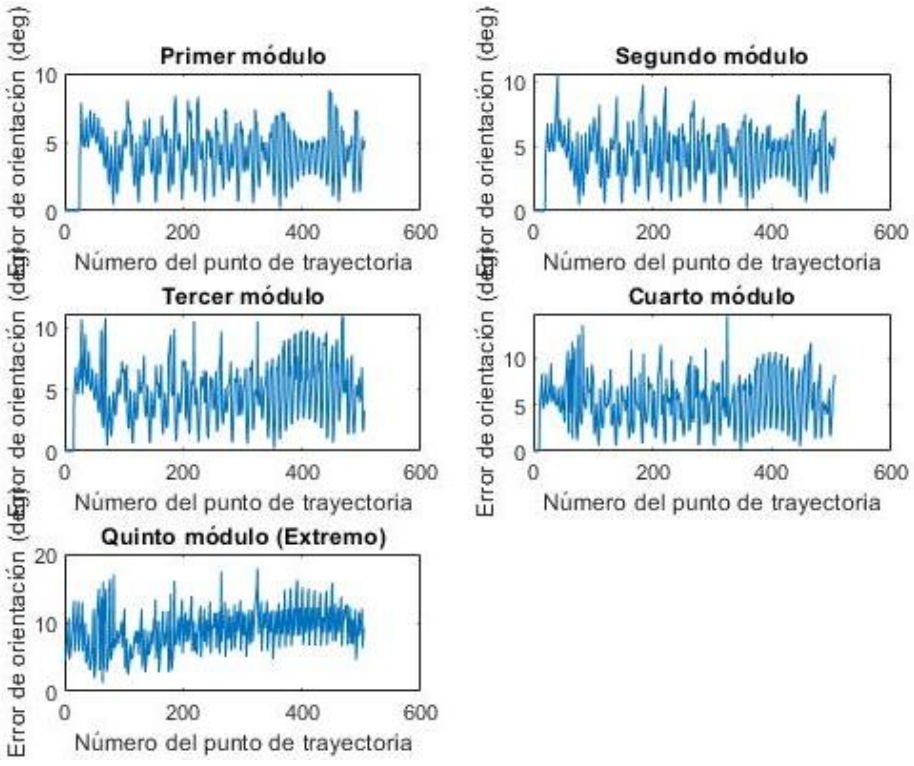


Figura 7.15 Errores de orientación de los módulos en una trayectoria espiral de radio 80mm, paso 40 mm y 2 espiras

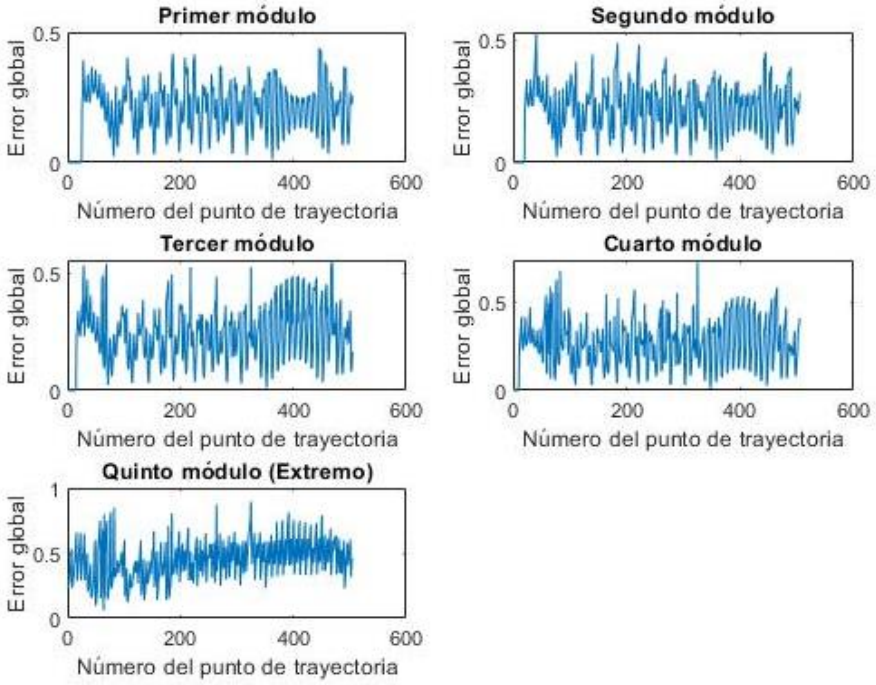


Figura 7.16 Errores globales de los módulos en una trayectoria espiral de radio 80mm, paso 40 mm y 2 espiras

7.1.4 Trayectoria en forma de intestino

La última trayectoria seguida es una simplificación del intestino grueso en 2D. Las dimensiones de cada tramo se muestran en la Figura 7.17 y son: colon ascendente 200 mm, colon transverso 240 mm, colon descendente 200 mm, colon sigmoideo 110 mm y recto 50 mm. Las uniones entre los tramos son giros de 90º y radio 60 mm. Los radios de giro se han tenido en cuenta al crear la trayectoria de manera que los tramos rectos no miden lo mismo que los tramos descritos previamente, pues parte de la longitud de los tramos forman parte de las curvas.

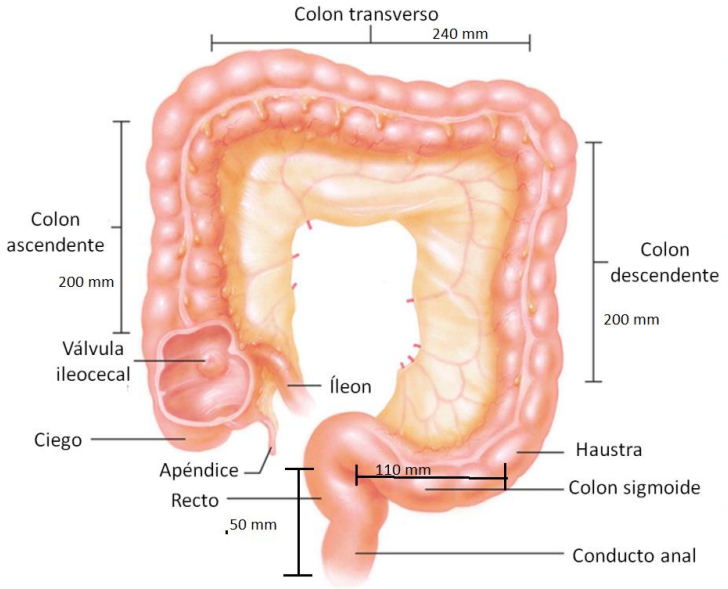


Figura 7.17 Dimensiones del intestino grueso

La trayectoria se muestra en azul en la Figura 7.18. En esa misma imagen también se observa en rojo la posición del robot al finalizar el seguimiento.

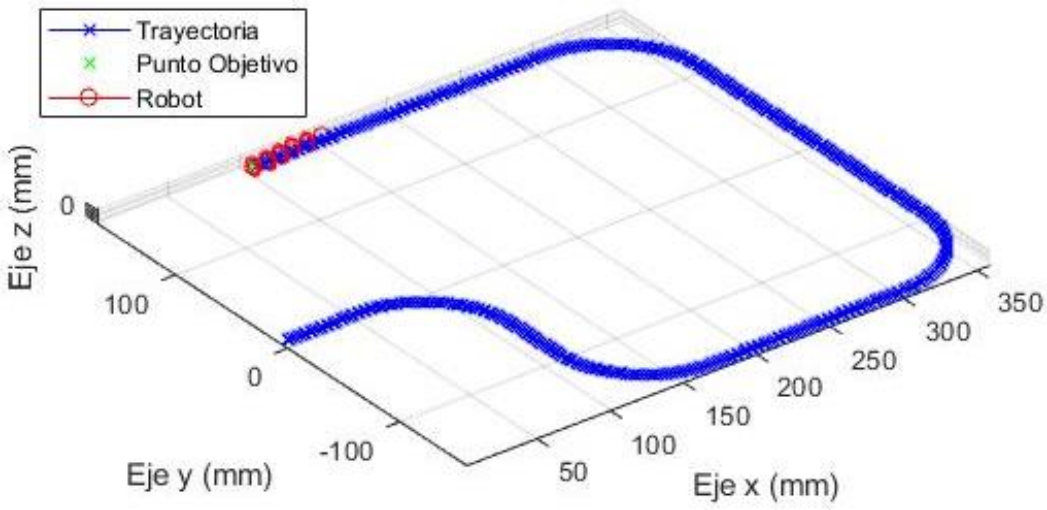


Figura 7.18 Resultado de seguimiento de una trayectoria con forma de intestino grueso

En las Figuras 7.19, 7.20, 7.21 se observan los errores de posición, orientación y global que se van produciendo a medida que se recorre la trayectoria.

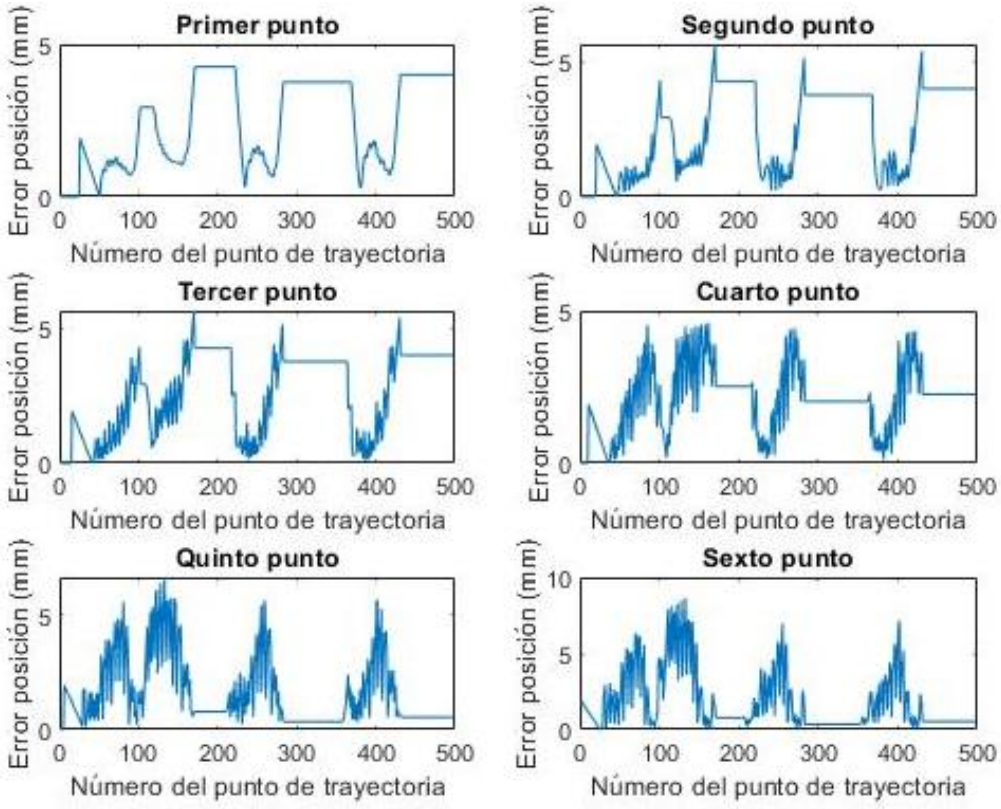


Figura 7.19 Errores de posición de los puntos en una trayectoria con forma de intestino grueso

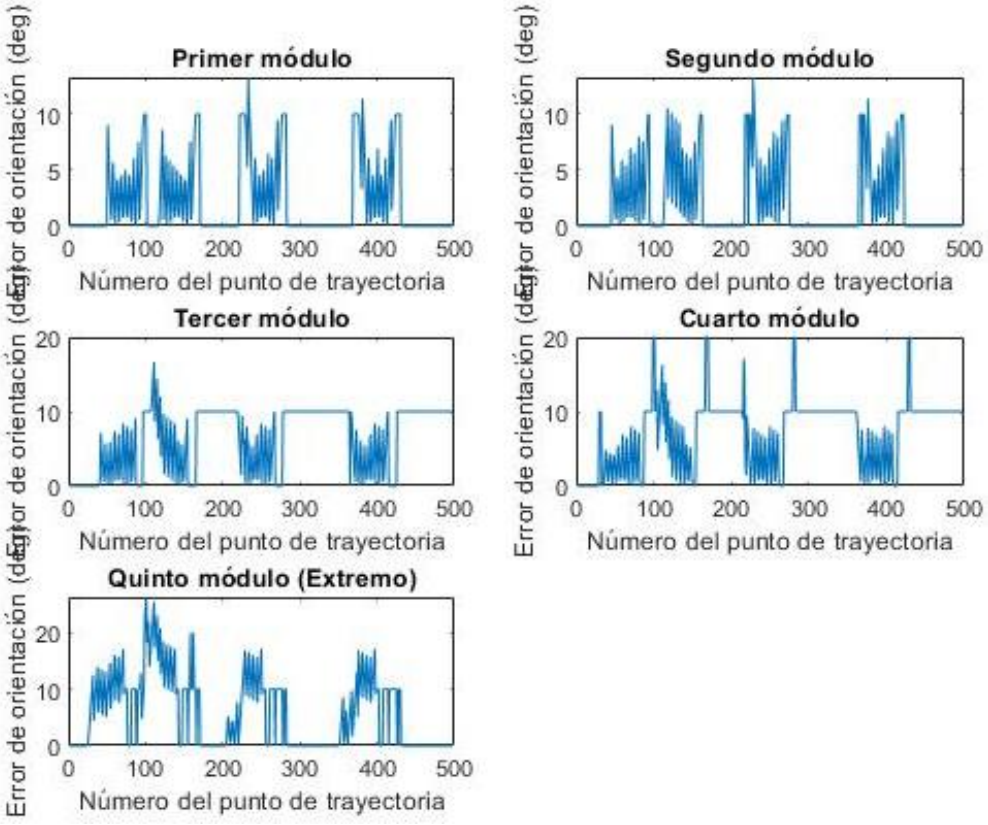


Figura 7.20 Errores de orientación de los módulos en una trayectoria con forma de intestino grueso

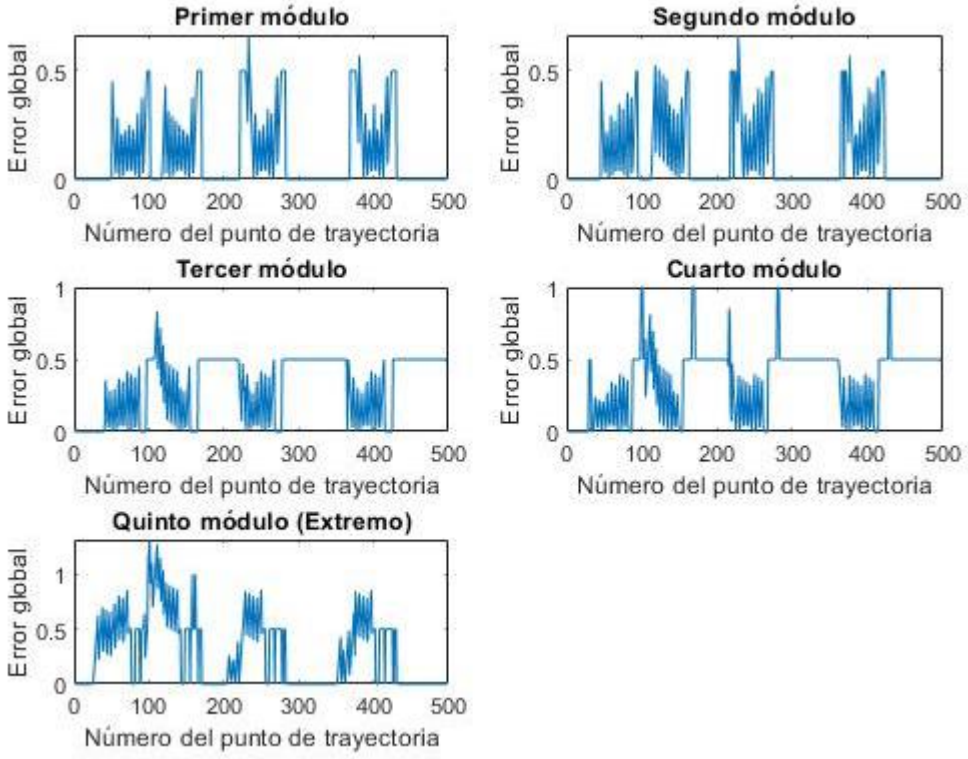


Figura 7.21 Errores globales de los módulos en una trayectoria con forma de intestino grueso

En los tramos rectos los errores son constantes y el valor máximo que alcanza está en torno a 4 mm. Los puntos con errores más elevados en las rectas son

los más próximos a la base. Se deben a que al realizar los giros el robot sale abriéndose hacia el exterior de la curva y este pequeño desplazamiento se corrige en el extremo del robot, que busca acercarse lo máximo al punto objetivo. Al lograrlo los módulos de la base no tienen la necesidad de corregir la posición. El momento más crítico tiene lugar al realizar el segundo giro, ya que se alcanzan errores de posición alrededor de los 8 mm en el punto 6, siendo el mayor error de 8,655 mm en el punto 133 de la trayectoria. Este error corresponde al último punto del robot. Este error supera la tolerancia por 0,655 mm por lo tanto no es especialmente significativo. Además, en el tramo que se produce este error, la anchura del intestino es de aproximadamente 45 mm. Por lo tanto, no supone un problema ya el error que la tolerancia se estableció para el punto más estrecho del intestino. Los errores de posición en los giros disminuyen a medida que el robot realiza nuevos giros.

En cuanto a los errores de orientación, el primer giro vuelve a ser el punto más crítico superándose la tolerancia al obtener un error de 23,18º en el último módulo. A partir de este punto se cumple la tolerancia en todo momento aumentando los errores en los giros y disminuyendo en las rectas.

Cabe destacar que el error de orientación solamente es relevante al finalizar la trayectoria. Durante el resto del recorrido interesa que no sea muy alto porque significa que el robot va a seguir con más facilidad la trayectoria, pero no es fundamental que sea muy bajo.

El error global en el segundo giro sí que supera el valor 1 ya que, aunque no sea de forma exagerada, se superan las tolerancias de posición y orientación. Sin embargo, como se acaba de explicar, por el punto en el que se encuentra el robot superar el error global límite no es crítico.

7.2 Interpretación de los resultados

En el apartado anterior se realizaba un análisis del resultado obtenido en cada trayectoria bajo las gráficas obtenidas. Sin embargo, no se explicó a qué se debe la forma de dientes de sierra dibujada en las gráficas.

El robot a medida que avanza va cambiando su configuración para adaptarse a la trayectoria. Sin embargo, debido a la condición por la que los actuadores deben ser binarios no cambiarán la configuración hasta que el ángulo formado entre el ángulo ideal girado sea mayor que $\frac{\alpha}{2}$. Esto provoca que los errores de posición y orientación vayan aumentando hasta que el actuador cambie de posición. Entonces el cambio de posición acercará repentinamente el punto del robot al punto de la trayectoria reduciéndose significativamente los errores de posición, orientación y

en consecuencia el global. Este fenómeno se muestra en las Figuras 7.22 y 7.23 que corresponden al seguimiento de la trayectoria curva 1, Apartado 7.1.2.1. Este fenómeno aparece en todas las gráficas y en todas se debe a este motivo.

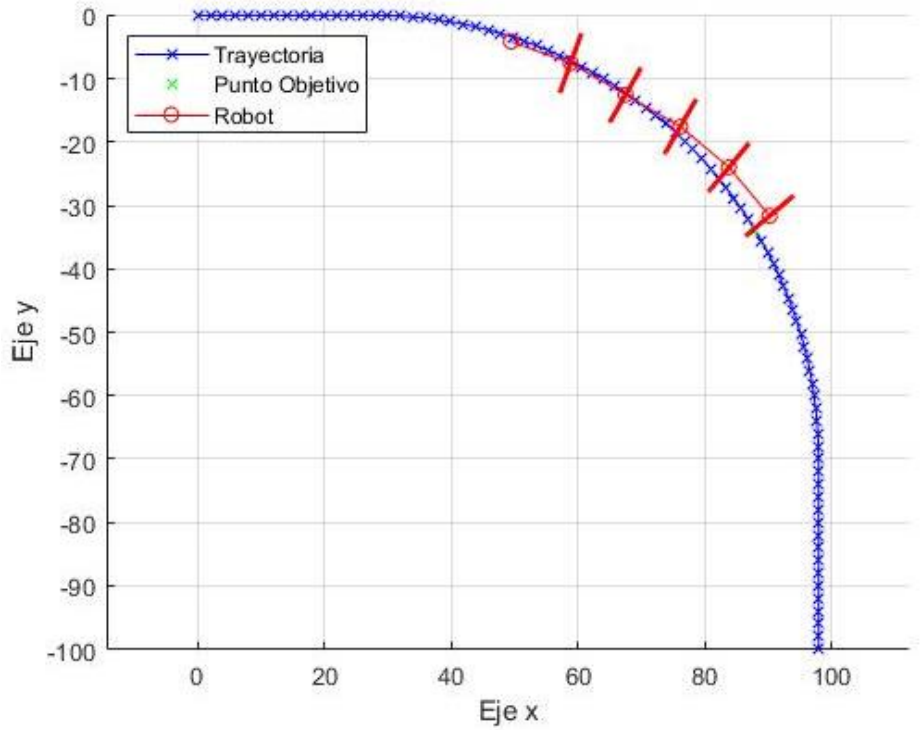


Figura 7.22 Configuración del robot en el punto 50 de la trayectoria. Error de posición del último punto alto (pico en la gráfica)

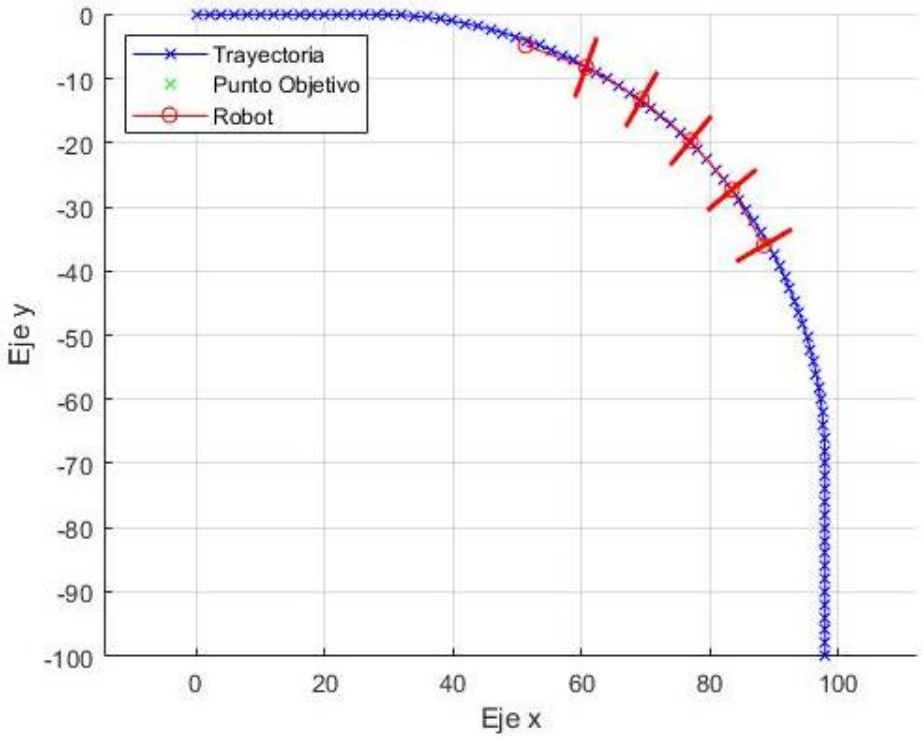


Figura 7.23 Configuración del robot en el punto 51 de la trayectoria. Error de posición del último punto bajo (valle en la gráfica)

CAPÍTULO 8

8. Conclusiones y líneas futuras

8.1 Conclusiones

Tras realizar del Trabajo de Fin de Grado se han obtenido las siguientes conclusiones:

- Se ha realizado un profundo y minucioso estudio de la robótica aplicada al campo de la endoscopia, proporcionándonos información para abordar el proyecto.

- Se ha realizado un estudio de dos de los algoritmos más prometedores para representar la cinemática inversa de robots hiper-redundantes (FABRIK y CCD). Tras los resultados obtenidos se ha decidido que el algoritmo FABRIK es el más adecuado para representar la cinemática inversa.

- Se ha mejorado el algoritmo FABRIK, aplicando las restricciones y modificaciones correspondientes para que represente correctamente el movimiento del robot y además tenga una buena precisión tanto en posición como en orientación.

- Se ha desarrollado un programa de seguimiento de trayectorias sobre el que se ha aplicado el algoritmo FABRIK mejorado. Tras realizar varias simulaciones se ha comprobado que el algoritmo funciona correctamente, siempre y cuando se respeten las limitaciones físicas del robot.

En líneas generales, se ha completado el objetivo principal marcado que era desarrollar un algoritmo que permitiera seguir trayectorias.

8.2 Líneas futuras

Este Trabajo de Fin de Grado forma parte de un proyecto de investigación. Alguno de los puntos sobre los que se puede trabajar en el futuro:

- Análisis de la transición entre configuraciones: Estudiar las diferentes estrategias posibles para que la transición sea lo más suave posible, junto con el menor coste computacional.

- Búsqueda de algoritmos alternativos: Investigar nuevos algoritmos que puedan ser empleados para el cálculo de la cinemática inversa y el

seguimiento de trayectorias. Comparar los nuevos algoritmos con el desarrollado en este trabajo para comprobar cual obtiene mayor precisión dentro de un coste computacional razonable.

- Estudio sobre materiales: Se debe realizar un estudio de los materiales que compondrán los módulos, así como del recubrimiento del robot. Se debe tener en cuenta que el exterior del robot estará en contacto con el cuerpo humano, por lo que deben emplearse materiales biocompatibles.

- Estudio dinámico: Se debe realizar un estudio sobre las fuerzas y momentos a los que estará sometido el robot en el intestino y comprobar si la fuerza que realizan los actuadores es suficiente para vencer la resistencia que recibe.

- Búsqueda de nuevos actuadores: Se ha realizado un estudio sobre actuadores de nitinol, pero se ha comprobado que se produce una acumulación de calor excesiva en los compartimentos donde se encuentran los actuadores. Se ha encontrado una alternativa de actuadores llamados Peano-HASEL cuyo funcionamiento se basa en la aplicación de un voltaje controlado sobre una cápsula de gel dieléctrico. Para comprobar la viabilidad de este tipo de actuadores, se debe construir un prototipo sobre el que realizar diversas pruebas que simulen los diferentes requerimientos a los que los actuadores estarán sometidos en el robot real.

BIBLIOGRAFÍA

Bibliografía

TFG, libros y artículos

Abovitz, R. (2001). *Digital surgery: the future of medicine and human-robot symbiotic interaction*, Industrial Robot: An International Journal, Vol. 28, Iss 5, pp. 401 – 406

Álvarez, J. (2019). *ESTRATEGIAS DE SEGUIMIENTO DE TRAYECTORIAS Y DETERMINACIÓN DE VELOCIDADES DE UN MANIPULADOR BHR.* Trabajo de Fin de Grado de la Escuela de Ingenierías Industriales de Valladolid, Universidad de Valladolid.

Aristidou, A., Lasenby, J. (2011). *FABRIK: a fast, iterative solver for the Inverse Kinematics problema*, Eselvier, Graphical Models, Vol 73, Iss 5, pp. 243-260.

Barrientos, A., Peñín,L.P., Balaguer, C., Aracil, R. (1997*). FUNDAMENTOS DE ROBÓTICA*. Ed. McGraw-Hill. Madrid

Cabezas, G. (2019). *MODELADO DINÁMICO DE UN ROBOT BHRM MEDIANTE LA HERRAMIENTA SIMSCAPE DE MATLAB.* Trabajo de Fin de Grado de la Escuela de Ingenierías Industriales de Valladolid, Universidad de Valladolid.

Cervantes, A., Chirivella, I., García-Granero, E. (2003). *Cáncer de colon y recto: Conceptos actuales en la patogenia, diagnóstico precoz, estudio de extensión, pronóstico y tratamiento*. Revista médica Universidad de Navarra, Vol 47, Nº1, pp. 64 – 68.

Garzón Oviedo, M. A. (2011). *Estrategias Bio-Inspiradas para Locomoción de Robots Ápodos.*Trabajo Fin de Máster de la Escuela Técnica Superior de Ingenieros Industriales, Universidad Politécnica de Madrid.

Hannan, M. W., Ian D. Walker, I. D. (2002). *Kinematics and the Implementation of an Elephant's Trunk Manipulator and Other Continuum Style Robots*. Department of Electrical and Computer Engineering. Clemson University.

Hontiyuelo, E. (2018). *PLANIFICACIÓN DE MOVIMIENTOS DE UN ROBOT ÁPODO, MODULAR E HIPERREDUNDANTE APLICANDO ALGORITMOS DE SEGUIMIENTO*. Trabajo de Fin de Grado de la Escuela de Ingenierías Industriales de Valladolid, Universidad de Valladolid.

Kenwright, B. (2012). *Inverse Kinematics – Cyclic Coordinate Descent (CCD)*. Journal of Graphics Tools, 16:4, pp. 177-217

Muñoz, S. (2013). *Análisis de la cinemática inversa y minimización de errores mediante algoritmos de optimización para un robot hiper-redundante.* Máster en Investigación en Ingeniería de Procesos y Sistemas, Universidad de Valladolid.

Paljug, E., Ohm, T., Hayatj, S. (1995). *The JPL Serpentine Robot: a 12 DOF System for Inspection.* Jet Propulsion Laboratory, Vol. 3, pp. 3143-3148.

Song, W., Hu, G. (2011*). A Fast Inverse Kinematics Algorithm for Joint Animation.* Procedia Engineering. Vol. 24, pp. 350-354.

S. Tappe, J. Kotlarski, T. Ortmaier, M. Dörbaum, A. Mertens and B. Ponick (2015), *The kinematic synthesis of a spatial, hyper-redundant system based on binary electromagnetic actuators, 2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, Queenstown, 2015, pp. 211-216, doi: 10.1109/ICARA.2015.7081149.

Thibodeau, G.A., Patton, K. T. (2007). *Anatomía y fisiología.* Elsevier Imprint, 6ª Edición

Trivedi, D., Rahn, C. D., Kierb, W. M., Walkerc, I.D. (2008). *Soft robotics: Biological inspiration, state of the art, and future research*. Applied Bionics and Biomechanics.

Williams, R. L., Mayhew, J. B. (1997). *OBSTACLE-FREE CONTROL OF THE HYPER REDUNDANT NASA INSPECTION MANIPULATOR*. Department of Mechanical Engineering. Ohio University.

Yttersad Pettersen, K., & Transeth, A. A. (2006). *Developments in Snake Robot Modeling and Locomotion.* Trondheim, Noruega: Department of Engineering Cybernetics Norwegian University of Science and Technology.

Xie, H., Wang, C., Li, S., Hu, L., Yang, H. (2019) *A geometric approach for follow-the- leader motion of serpentine manipulator.* International Journal of Advanced Robotic Systems.

Páginas web

AECC. Cáncer de colon (29/10/2020)
https://www.aecc.es/es/todo-sobre-cancer/tipos-cancer/cancer-colon

Fundación Española del Aparato Digestivo (FEAD). Enfermedades y síntomas (28/04/2020)
https://www.saludigestivo.es/enfermedades-digestivas-y-sintomas/#estomago

Anexo

Anexo: Scripts de MATLAB

Se mostrarán los scripts de Matlab utilizados y se explicará parte por parte que función realiza cada uno.

1. Seguimiento_Trayectoria.m

   Este script es el programa que se ejecuta para realizar el seguimiento de trayectorias. Dentro de este programa se encuentran varias funciones de Matlab que se explicarán en los próximos apartados.

   En primer lugar, se encuentra un clear y un clc para eliminar variables de simulaciones anteriores y la ventana de comandos.

```
clear
clc
```

   A continuación, se introducen los datos del robot. P será la matriz con las coordenadas de los puntos del robot. En primer lugar, se crea la matriz llena de ceros con la dimensión necesaria, para rellenar las posiciones con las coordenadas iniciales del robot empleando un bucle for.

```
%% Datos del robot

Origen=[0 0 0];       %Punto en el que se encuentra la base del
robot
h=10;                 %Altura de cada módulo en mm
Radio=5;             %Radio del robot
nh=5;                 %Número de módulos del robot con los que
trabaja el algoritmo
alpha=10;             %Ángulo máximo de giro entre módulos en
grados
P=zeros(nh+1,3);
P(nh+1,:)=Origen;

for i=nh:-1:1

    P(i,:)=P(i+1,:) - [h 0 0];     %Se genera la posición inicial
del robot

end
```

   Para poder realizar más adelante los giros restringiendo los ejes, se deben crear en función de la posición inicial.

```
%% Creamos los ejes locales de los módulos
%%Debido a la disposición inicial que hemos dado al robot(a lo
largo del
%%eje X) los ejes de giro serán 'Y' y 'Z'

for i=1:nh
```

```
    EjeY(i,:)=P(i,:)+[0,1,0];
    EjeZ(i,:)=P(i,:)+[0,0,1];
end
```

Se establecen los diferentes parámetros que dependen del problema. Si se emplea el programa para otra función se pueden modificar los parámetros para que se ajusten más al nuevo problema.

```
%% Datos del problema

tol_pos=8 ;                 %Tolerancia del error de posición
Target_ori=[0,-1,0];          %Orientación objetivo en el último
punto
tol_ori=20;                   %Tolerancia de orientación
kmax=30;                     %Número máximo de iteraciones
f=5;                         %Es el número por el que se dividirá
la altura de cada módulo para definir el avance.
feed=h/f;                    %Avance, es decir
distanctipo_trayectoria que avanza el feeder entre cada punto.
```

Se pregunta por la trayectoria que se quiere seguir y según cual se decida se introducirán los datos para definirla.

```
%% Datos de la trayectoria
tipo_trayectoria=0;
while tipo_trayectoria < 1 | tipo_trayectoria>4
tipo_trayectoria=input('Seleccione el tipo de trayectoria
(1=recta  2=espiral 3=curva 4=intestino plano ): '); %Se
selecciona el tipo de trayectoria que seguirá el robot
switch tipo_trayectoria
    case 1,
        PA = input('Indique la longitud de la recta en mm:');
        PB = input('Indique el ángulo que forma la recta con el
eje x en grados:'); %La proyección de la trayectoria en el plano
xy, antihorario
        PC = input('Indique el ángulo que forma la recta con el
plano xy en grados:'); %Arriba positivo, abajo negativo.
        PD = 0;
    case 2,
        PA = input('Indique el radio de la espira en mm:');
        PB = input('Indique el paso en mm:');
        PC = input ('Indique el número de espiras:');
        PD = 0;
    case 3,
        PA = input('Indique el rádio de giro en mm:');
        PB = input('Indique el angulo girado en grados:');
        PC = input('Indique la distancia de entrada en mm:');
        PD = input('Indique la distancia de salida en mm:');
    case 4,
        PA = 0;
        PB = 0;
        PC = 0;
        PD = 0;
end
end
```

Se introducen los parámetros solicitados para definir la trayectoria en la función Generador_Trayectorias (está función se explicará más adelante). La función devuelve tres vectores, cada uno contiene una componente de las coordenadas de los puntos de la trayectoria. Los tres vectores se agrupan en una matriz de forma que cada fila de la matriz corresponde a las coordenadas x, y, z de un punto de la trayectoria.

```
%% Generación de la trayectoria

[xi, yi, zi] =
Generador_Trayectorias(tipo_trayectoria,PA,PB,PC,PD,feed);

%La función da 3 vectores que contienen las coordenadas de una
serie de
%puntos equiespaciados una distancia que se ha decidido que
valga igual que el feed, definen la trayectoria

Trayectoria=[xi' yi' zi'];


%% Ejecución del algoritmo y representación simultánea
```

Se calculan las orientaciones entre los puntos de las trayectorias para poder introducirlas como orientación objetivo del extremo del robot.

```
for i=1:length(xi)-1

    orientacion(i,:)=(Trayectoria(i+1,:)-
Trayectoria(i,:))/norm(Trayectoria(i+1,:)-Trayectoria(i,:));
%Orientación a seguir para que se pueda seguir la trayectoria de
manera natural

end

i=0;
%Variable que indica por que punto de la trayectoria nos
llegamos
```

Se inicia el bucle while donde se realiza el cálculo de las posiciones del robot a lo largo de la trayectoria. La variable 'i' indica el número de punto de la trayectoria en el que nos encontramos. El bucle while se ejecutará hasta que se alcance que 'i' es igual que el número de elementos en el vector 'xi'. Según se inicia el bucle se suma 1 a 'i' y se asigna el primer punto objetivo (Target_pos') y la primera orientación objetivo ('Ori'). A medida que el robot vaya avanzando 'i' irá sumando y con el se irán cambiando el punto objetivo y la orientación objetivo.

```
while i<length(xi)

    i=i+1
    Target_pos=Trayectoria(i,:);    %Punto objetivo en esa
iteración
```

```
    if i == length(xi)                  %Cuando llegamos al
último punto se asigna la orientación objetivo introducida en
los datos del problema

        Ori=orientacion(end,:);

    else                                %Si no nos encontramos
en el último punto la orientación será la definida por la unión
de los puntos de la trayectoria, calculada en la línea 80

        Ori=orientacion(i,:);

    end
```

Se ejecuta el algoritmo que hemos creado en el que se deben introducir las variables que aparecen entre paréntesis y devuelve las variables que están entre corchetes. 'P' es la matriz que contiene las nuevas coordenadas de los puntos del robot. Se almacenan los errores de posición y orientación para poder representarlos en gráficas al finalizar el programa.

```
[P,error_pos,error_ori,EjeY,EjeZ,k]=FABRIK_7_ACTUADORES_BINARIOS
_CON_EJES_RESTRINGIDOS(Target_pos,Ori,EjeY,EjeZ,P,h,nh,tol_pos,t
ol_ori,alpha,kmax);
    error_pos_acum(i)=error_pos;    %Se almacenan los errores de
posición
    error_ori_acum(i)=error_ori;    %Se almacenan los errores de
orientación
    i_acum(i)=i;

    error_pos_puntos(i,nh+1)=error_pos;

    for j=1:nh

        ori_modulos(j,:)=(P(j+1,:)-P(j,:))/norm(P(j+1,:)-
P(j,:));

    end

    for j=1:nh

        if (i-(nh+1-j)*f)<1

           error_pos_puntos(i,j)=0;

        else

           error_pos_puntos(i,j)=real(norm(P(j,:)-
Trayectoria((i-(nh+1-j)*f),:)));

        end

    end

    error_ori_modulos(i,nh)=error_ori;
```

```
    for j=1:(nh-1)

        if (i-(nh+1-j)*f)<1

           error_ori_modulos(i,j)=0;

        else

            ori_modulo=(P(j+1,:)-P(j,:));

error_ori_modulos(i,j)=real(acosd(dot(ori_modulo,orientacion((i-
(nh+1-j)*f),:))/(norm(ori_modulo)*norm(orientacion((i-(nh+1-
j)*f),:))));

        end

    end
```

Se realiza la representación gráfica del robot y la trayectoria. A medida que avanza y se calculan las nuevas configuraciones se van representando, de modo que se vea como recorre el robot la trayectoria.

```
%A continuación se hace la representación de la configuración
obtenida al aplicar el algoritmo
    plot3(Trayectoria(:,1),Trayectoria(:,2),Trayectoria(:,3),'-
xb')
    hold on
    plot3(Target_pos(1),Target_pos(2),Target_pos(3),'xg')
%Punto objetivo de la iteración
    plot3(P(:,1),P(:,2),P(:,3),'-or')  %Representación de la
configuración del robot de la iteración
    for j=1:nh

        circulo(P(j+1,:),ori_modulos(j,:),Radio)

    end
    xlabel('Eje x')
    ylabel('Eje y')
    zlabel('Eje z')
    legend('Trayectoria','Punto
Objetivo','Robot','Location','northwest')
    grid on
    view(0,90)
    axis equal
    drawnow
    pause
    hold off

    ori_base=(P(2,:)-P(1,:))/norm(P(2,:)-P(1,:));
%Orientación de la base del robot
```

Se realiza el avance del robot desplazando los puntos una distancia igual al feed definido previamente en la dirección que coincida con la orientación del módulo de la base. También se desplazan los ejes que se están utilizando para limitar los giros. Para evitar que los ejes no coincidan con los puntos y se realicen los giros

de forma incorrecta se resetean de tal forma que el eje del primer punto se mantenga y el resto de los ejes se coloquen como si todos los puntos del robot estuvieran en la configuración 0, de forma que el robot esté recto.

```
    P=P+feed*ori_base;     %Realizamos el avance
    EjeY=EjeY+feed*ori_base;
    EjeZ=EjeZ+feed*ori_base;

    Dir_Y=(EjeY(1,:)-P(1,:))/norm(EjeY(1,:)-P(1,:));  %Se
calcula la dirección del eje Y del primer módulo
    Dir_Z=(EjeZ(1,:)-P(1,:))/norm(EjeZ(1,:)-P(1,:));  %Se
calcula la dirección del eje Z del primer módulo

    for j=1:nh    %Se generan unos nuevos ejes a lo largo de la
dirección del módulo 1

        EjeY(j+1,:)=P(1,:)+ori_base*h*j+Dir_Y;
        EjeZ(j+1,:)=P(1,:)+ori_base*h*j+Dir_Z;

    end

end
```

El bucle while finaliza aquí. Una vez se tienen los errores de posición y orientación almacenados se calcula el error global tomando como referencia las tolerancias establecidas. Debido a que hay 6 puntos y 5 módulos el punto de la base no se utilizará para calcular el error global de tal forma que el último módulo esté asociado con el último punto, el penúltimo módulo con el penúltimo punto y así sucesivamente.

```
%Error global
for i=1:nh


error_global(:,i)=(error_pos_puntos(:,i+1)/tol_pos)+(error_ori_m
odulos(:,i)/tol_ori);

end
```

Se representan en gráficas los errores de posición a lo largo de la trayectoria de los 6 puntos del robot.

```
%Representación de los errores de posición de los puntos

f1=figure('Name','Errores de posición de los puntos')
hold on
subplot(3,2,1)
plot(i_acum,error_pos_puntos(:,1))
xlabel('Número del punto de trayectoria')
ylabel('Error posición (mm)')
title('Primer punto')

subplot(3,2,2)
plot(i_acum,error_pos_puntos(:,2))
xlabel('Número del punto de trayectoria')
```

```
ylabel('Error posición (mm)')
title('Segundo punto')

subplot(3,2,3)
plot(i_acum,error_pos_puntos(:,3))
xlabel('Número del punto de trayectoria')
ylabel('Error posición (mm)')
title('Tercer punto')

subplot(3,2,4)
plot(i_acum,error_pos_puntos(:,4))
xlabel('Número del punto de trayectoria')
ylabel('Error posición (mm)')
title('Cuarto punto')

subplot(3,2,5)
plot(i_acum,error_pos_puntos(:,5))
xlabel('Número del punto de trayectoria')
ylabel('Error posición (mm)')
title('Quinto punto')

subplot(3,2,6)
plot(i_acum,error_pos_puntos(:,end))
xlabel('Número del punto de trayectoria')
ylabel('Error posición (mm)')
title('Sexto punto')
```

Se representan los errores de orientación a lo largo de la trayectoria de los módulos del robot.

```
%Representación de los errores de orientación de los módulos

f2=figure('Name','Errores del orientación de los módulos')
subplot(3,2,1)
plot(i_acum,error_ori_modulos(:,1))
xlabel('Número del punto de trayectoria')
ylabel('Error de orientación (deg)')
title('Primer módulo')

subplot(3,2,2)
plot(i_acum,error_ori_modulos(:,2))
xlabel('Número del punto de trayectoria')
ylabel('Error de orientación (deg)')
title('Segundo módulo')

subplot(3,2,3)
plot(i_acum,error_ori_modulos(:,3))
xlabel('Número del punto de trayectoria')
ylabel('Error de orientación (deg)')
title('Tercer módulo')

subplot(3,2,4)
plot(i_acum,error_ori_modulos(:,4))
xlabel('Número del punto de trayectoria')
ylabel('Error de orientación (deg)')
title('Cuarto módulo')

subplot(3,2,5)
```

```
plot(i_acum,error_ori_modulos(:,5))
xlabel('Número del punto de trayectoria')
ylabel('Error de orientación (deg)')
title('Quinto módulo (Extremo)')
hold off
```

Se representan los errores globales de los módulos a lo largo de la trayectoria.

```
%Representación de los errores globales de los módulos

f3=figure('Name','Errores globales de los módulos')
subplot(3,2,1)
plot(i_acum,error_global(:,1))
xlabel('Número del punto de trayectoria')
ylabel('Error global')
title('Primer módulo')

subplot(3,2,2)
plot(i_acum,error_global(:,2))
xlabel('Número del punto de trayectoria')
ylabel('Error global')
title('Segundo módulo')

subplot(3,2,3)
plot(i_acum,error_global(:,3))
xlabel('Número del punto de trayectoria')
ylabel('Error global')
title('Tercer módulo')

subplot(3,2,4)
plot(i_acum,error_global(:,4))
xlabel('Número del punto de trayectoria')
ylabel('Error global')
title('Cuarto módulo')

subplot(3,2,5)
plot(i_acum,error_global(:,5))
xlabel('Número del punto de trayectoria')
ylabel('Error global')
title('Quinto módulo (Extremo)')
hold off
```

Por último, se calcularon las medias y medianas de los errores, aunque debido a la forma de sierra de las gráficas el resultado no resulta útil, pues, aunque el error medio no sea elevado pueden existir varios puntos con errores grandes que no podrían admitirse como válidos.

```
for j=1:nh+1

    error_pos_mediana_aux=error_pos_puntos((1+(nh+1-
j)*5):end,j);   %Se eliminan los primeros ceros para no
adulterar el resultado
    mediana_pos(j)=median(error_pos_mediana_aux);
    media_pos(j)=mean(error_pos_mediana_aux);

end
```

```
for j=1:nh

    error_ori_mediana_aux=error_ori_modulos((1+(nh+1-
j)*5):end,j); %Se eliminan los primeros ceros para no adulterar
el resultado
    mediana_ori(j)=median(error_ori_mediana_aux);            %Se
calcula la mediana del error de orientación
    media_ori(j)=mean(error_ori_mediana_aux);                %Se
calcula la media del error de orientación

    error_global_mediana_aux=error_global((1+(nh+1-j)*5):end,j);
%Se eliminan los primeros ceros para no adulterar el resultado
    mediana_global(j)=median(error_global_mediana_aux);
%Se calcula la mediana del error global
    media_global(j)=mean(error_global_mediana_aux);
%Se calcula la media del error global

end

n_puntos=[1:(nh+1)];
n_modulos=[1:nh];

fprintf('Mediana del error de posición \n')

for i=1:(nh+1)

    fprintf('Punto %2.0f   %2.4f \n',i, mediana_pos(i))

end

fprintf('Mediana del error de orientación \n')

for i=1:nh

fprintf('Módulo %2.0f   %2.4f \n',i, mediana_ori(i))

end

fprintf('Mediana del error de global \n')

for i=1:nh

fprintf('Módulo %2.0f   %2.4f \n',i, mediana_global(i))

end

fprintf('Media del error de posición \n')

for i=1:(nh+1)

    fprintf('Punto %2.0f   %2.4f \n',i, media_pos(i))

end
fprintf('Media del error de orientación \n')

for i=1:nh
```

```
fprintf('Módulo %2.0f    %2.4f \n',i, media_ori(i))

end

fprintf('Media del error de global \n')

for i=1:nh

fprintf('Módulo %2.0f    %2.4f \n',i, media_global(i))

end
```

2. FABRIK_7_ACTUADORES_BINARIOS_CON_EJES_RESTRINGIDOS.m

Este script contiene la función del algoritmo de FABRIK mejorado. Es utilizado en el script Seguimiento_Trayectoria.m, pero puede se utilizado en cualquier otro script siempre que se introduzcan las variables necesarias.

Esta función solicita como entradas las variables Target_pos, Target_ori, Eje1, Eje2, Posicion_inicial, h, nh, tol_pos, tol_ori, Alpha y kmax, que se definieron en el script Seguimiento_Trayectoria.m y devuelve las variables P, error_pos, error_ori, Eje1, Eje2 y k. Los ejes no tienen se han numerado en lugar de llamarlos según las letras del eje debido a que se pueden utilizar otros ejes globales diferentes por lo que las letras de los ejes no coincidan.

```
function
[P,error_pos,error_ori,Eje1,Eje2,k]=FABRIK_7_ACTUADORES_BINARIOS
_CON_EJES_RESTRINGIDOS(Target_pos,Target_ori,Eje1,Eje2,Posicion_
inicial,h,nh,tol_pos,tol_ori,alpha,kmax)

%%En esta variante del algoritmo FABRIK en las 10 primeras
iteraciones se
%%ejecuta el algoritmo completo sin ninguna restricción de
ningún tipo
%%excepto la orientación.
%%A partir de la 11 iteración se aplican restricciones de giro
en el
%%recorrido de la base al extremo. Y restricciones de
orientación en el
%%recorrido del extremo a la base.
%%Se aplica restricción a los ejes respecto a los que pueden
girar los
%%módulos

P=Posicion_inicial;
Target_ori=Target_ori/norm(Target_ori);    %Vector dirección ori

%Generamos una matriz de las orientaciones de cada módulo
orientacion=zeros(nh,3);

for i=1:nh
    orientacion(i,:)=(P(i+1,:)-P(i,:))/norm(P(i+1,:)-P(i,:));
end
```

```
P_anterior_base=P(1,:)-h*orientacion(1,:);    %Punto auxiliar
para medir el ángulo del primer módulo respecto la base
P_base=P(1,:);                                %Se guarda el
primer punto que corresponde a la base para no perderlo al
realizar el recorrido del extremo a la base.
error_pos=norm(Target_pos-P(end,:));
ori_end=(P(end,:)-P(end-1,:))/norm(P(end,:)-P(end-1,:));
%Cambiar el error de orientacion a un ángulo
error_ori=acosd(dot(Target_ori,ori_end)/(norm(Target_ori)*norm(o
ri_end)));   %Ángulo entre la orientación del último módulo y la
deseada    %Si la distancia entre el obtetivo y el extremo es
mayor que la tolerancia se aplica el algoritmo FABRIK

'k' será la variable que utilizaremos para contar el número de
veces que se ha ejecutado el bucle.

k=0;
```

El bucle while es donde se realizan los cálculos de FABRIK. Se ejecutará mientras haya una de las tolerancias que no se cumple o bien no se hay realizado mínimo 11 iteraciones. Hay que recordar que al mejorar el algoritmo se decidió que las 10 primeras iteraciones no tuvieran restricciones de giro con el objetivo de mejorar la precisión.

```
while error_pos>tol_pos  |  error_ori>tol_ori | k<11    %Si la
distancia entre el objetivo y el extremo es mayor que la
tolerancia se aplica el algoritmo FABRIK

        %% Recorremos el robot del extremo a la base
           P(end,:)=Target_pos;
```

En esta primera parte se calculan los puntos del extremo a la base. No cambia entre las 10 primeras iteraciones y las siguientes.

```
           %El recorrido del extremo a la base se realiza sin
rectricciones
           %de giro

           for i=nh:-1:1

               if i==nh

                   P(end-1,:)=P(end,:)-h*Target_ori;

               else

                   ri=norm(P(i+1,:)-P(i,:));
                   landai=h/ri;
                   P(i,:)=(1-landai)*P(i+1,:)+landai*P(i,:);
%Punto que se obtiene sin restricciones

               end

           end
```

Aquí finaliza el recorrido del extremo a la base y comienza el recorrido en el sentido contrario. Cada vez que se llega a esta parte los ejes vuelven a ser los que se han introducido en la función porque si no los ejes no coincidirían con los puntos a los que corresponden.

Se comprueba si estamos en las 10 primeras iteraciones o no. Si estamos en las 10 primeras se ejecuta FABRIK sin ninguna restricción de giro.

```
%% Recorremos el robot de la base al extremo
           P(1,:)=P_base;
           Eje1_aux=Eje1;
           Eje2_aux=Eje2;
           Pos_ini=Posicion_inicial;
           for i=1:nh

               if k<10
                   ri=norm(P(i+1,:)-P(i,:));
                   landai=h/ri;
                   P(i+1,:)=(1-landai)*P(i,:)+landai*P(i+1,:);
%Punto que se obtiene sin restricciones
```

En caso de que no estemos en las 10 primeras iteraciones las coordenadas de los ejes se pasan a coordenadas locales de cada punto del robot. Además, se obtienen otros dos ejes en el sentido contrario a los que ya tenemos. Se hace esto porque el programa se ha escrito para que el ángulo que giren los actuadores sea 0 ó α, de modo que por ejemplo, en lugar de girar -α respecto al eje (1, 0, 0) se girará α respecto al eje (-1, 0, 0).

```
               else

                   for j=i:nh

                       Eje1_aux(j,:);
                       Eje1_aux(j,:)=Eje1_aux(j,:)-P(i,:);
%Ejes locales X de cada módulo respecto al punto de giro
                       Eje2_aux(j,:)=Eje2_aux(j,:)-P(i,:);
%Ejes locales Z de cada módulo respecto al punto de giro

                   end
                   for j=i:nh

                       Pos_ini(j+1,:)=Pos_ini(j+1,:)-P(i,:);
%Se emplea esta variable para obtener las coordenadas locales de
los ejes de giro.


                   end

                   Eje1_giro=Eje1_aux(i,:)/norm(Eje1_aux(i,:));
                   Eje2_giro=Eje2_aux(i,:)/norm(Eje2_aux(i,:));
                   Eje3_giro=-Eje1_giro;    %Eje local X en
sentido opuesto
```

```
                    Eje4_giro=-Eje2_giro;  %Eje local Z en
sentido opuesto

Se ejecuta el algoritmo sin restricciones para obtener el que
sería el punto ideal del robot (Pos_aux).

                    ri=norm(P(i+1,:)-P(i,:));
                    landai=h/ri;
                    Pos_aux=(1-landai)*P(i,:)+landai*P(i+1,:);
%Punto obtenido sin restricciones

                    if i==1                         %Para calcular
el punto P2 se emplea el punto auxiliar que correspondería a P0

                        v1=P_anterior_base-P(i,:);  %Vector fijo
respecto al que medimos el angulo
                        v2=Pos_aux-P(i,:);   %Vector creado con
el punto a mover

                    else

                        v1=P(i-1,:)-P(i,:);   %Vector fijo
respecto al que medimos el angulo
                        v2=Pos_aux-P(i,:);       %Vector que
creado con el punto a mover

                    end
```

Se calculan el eje de rotación y el ángulo rotado ideales. A continuación, se calculan los ángulos que forman los ejes de los actuadores con el eje ideal. El eje de los actuadores que forme un ángulo menor se seleccionará como eje de giro para ese punto.

```
                    vr=cross(v2,v1)/norm(cross(v2,v1));    %Eje
de rotación
                    theta=180-
acosd(dot(v2,v1)/(norm(v2)*norm(v1)));   %Ángulo rotado sin
restricciones
Se mide el ángulo
                    %Se crea un vector con los ángulos que forma
cada eje
                    %con el eje de giro ideal vr

Angulo_giro_Ejes(1)=acosd(dot(vr,Eje1_giro))/(norm(vr)*norm(Eje1
_giro));

Angulo_giro_Ejes(2)=acosd(dot(vr,Eje2_giro))/(norm(vr)*norm(Eje2
_giro));

Angulo_giro_Ejes(3)=acosd(dot(vr,Eje3_giro))/(norm(vr)*norm(Eje3
_giro));

Angulo_giro_Ejes(4)=acosd(dot(vr,Eje4_giro))/(norm(vr)*norm(Eje4
_giro));
```

```
                    [Angulo,Indice]=min(Angulo_giro_Ejes);
%Se obtiene el menor ángulo formado por los ejes y la posición
de dicho ángulo en el vector

                    switch Indice  %Se asigna al vector vr el
vector que menos ángulo forma con el vector vr ideal

                        case 1

                            vr=Eje1_giro;

                        case 2

                            vr=Eje2_giro;

                        case 3

                            vr=Eje3_giro;

                        case 4

                            vr=Eje4_giro;

                    end
```

Una vez se ha seleccionado el eje de giro se comprueba si el ángulo de giro ideal es mayor que la mitad del ángulo que giran los actuadores. Si es mayor se asigna a theta (ángulo que va a girar el actuador) el valor Alpha. En caso de que theta sea menor que α/2 el valor de theta será 0. Esto se debe a que los actuadores son binarios y por lo tanto solo pueden valer 0 ó α.

```
                    %%
                    if theta>alpha/2

                        theta=alpha;    %Si el ángulo recomendado
es más de la mitad del ángulo que puede realizar el actuador el
actuador se activará

                    else

                        theta=0;     %Si el ángulo recomendado es
menor que la mitad del ángulo que gira el actuador el actuador
no no girará

                    end
```

Se realizan las rotaciones empleando cuaternios duales. Se rotarán los puntos del robot, también los ejes y la variable Pos_ini para poder calcular los ejes locales a medida que se calculan los nuevos puntos.

```
%Se realiza la rotación del punto
                    DQP=DualQuaternion([1 0 0 0 0 -v1]);
%Cuaternio del punto nuevo sin girar
```

```
                    DQR=DualQuaternion([cosd(theta/2)
sind(theta/2)*vr 0 0 0 0]);  %Cuaternio de giro
                    DQ=DQR;
                    DQP_nuevo=DQ*DQP*DQ.c_qd;
                    P_aux=DQP_nuevo.double;
                    P(i+1,:)=P_aux(6:8)+P(i,:); %Nuevo punto con
la condición de ángulo máximo

                    for j=i:nh    %Giramos los ejes 1 locales
afectados por el giro del módulo i
                        DQP=DualQuaternion([1 0 0 0 0
Eje1_aux(j,:)]);   %Cuaternio del punto nuevo sin girar
                        DQR=DualQuaternion([cosd(theta/2)
sind(theta/2)*vr 0 0 0 0]);  %Cuaternio de giro
                        DQ=DQR;
                        DQP_nuevo=DQ*DQP*DQ.c_qd;
                        Eje1_girado_aux=DQP_nuevo.double;
                        Eje1_aux(j,:)=Eje1_girado_aux(6:8)+P(i,:);
                    end

                    for j=i:nh    %Giramos los ejes 2 locales
afectados por el giro del módulo i
                        DQP=DualQuaternion([1 0 0 0 0
Eje2_aux(j,:)]);   %Cuaternio del punto nuevo sin girar
                        DQR=DualQuaternion([cosd(theta/2)
sind(theta/2)*vr 0 0 0 0]);  %Cuaternio de giro
                        DQ=DQR;
                        DQP_nuevo=DQ*DQP*DQ.c_qd;
                        Eje2_girado_aux=DQP_nuevo.double;
                        Eje2_aux(j,:)=Eje2_girado_aux(6:8)+P(i,:);
                    end

                    for j=i:nh    %Giramos los ejes X locales
afectados por el giro del módulo i
                        DQP=DualQuaternion([1 0 0 0 0
Pos_ini(j+1,:)]);   %Cuaternio del punto nuevo sin girar
                        DQR=DualQuaternion([cosd(theta/2)
sind(theta/2)*vr 0 0 0 0]);  %Cuaternio de giro
                        DQ=DQR;
                        DQP_nuevo=DQ*DQP*DQ.c_qd;
                        Pos_ini_aux=DQP_nuevo.double;
                        Pos_ini(j+1,:)=Pos_ini_aux(6:8)+P(i,:);
                    end

                end

            end
```

Aquí se finaliza el recorrido de la base al extremo. Se calculan los errores de posición y orientación y se suma un al contador de iteraciones 'k'. Se comprueba si se ha alcanzado el límite de iteraciones, en caso de ser así se finaliza el bucle while automáticamente. Si no se ha alcanzado el límite de iteraciones se comprueban las condiciones del bucle while. Si no se satisfacen se repetirá otra vez, en caso de que se hayan alcanzado se finalizará la función.

```
            error_pos=norm(Target_pos-P(nh+1,:));
```

```
            ori_end=(P(end,:)-P(end-1,:))/norm(P(end,:)-P(end-
1,:)); %Cambiar el error de orientacion a un ángulo

error_ori=acosd(dot(Target_ori,ori_end)/(norm(Target_ori)*norm(o
ri_end)));   %Ángulo entre la orientación del último módulo y la
deseada
            k=k+1;

            if k>=kmax

                break

            end

end
Eje1=Eje1_aux;      %Se asignan las posiciones de los nuevos ejes
a la variable antigua. SI solo se introduce un punto no realiza
ninguna función,
Eje2=Eje2_aux;      %pero si se sigue una trayectoria es
necesario para conocer la posición de los ejes en el espacio a
medida que el robot avanza.
```

3. Generador_Trayectorias.m

Esta función se emplea también en el programa Seguimiento_Trayectoria.m. Se introducen el tipo de trayectoria que se quiere seguir y los parámetros necesarios para definirla y devuelve tres vectores que contienen las coordenadas de los puntos de la trayectoria.

```
function [ xi, yi, zi] = Generador_Trayectorias( Tipo, PA, PB,
PC, PD, h1)
%   Genera la trayectoria a partir del tipo y los datos
introducidos.
%   Trayectorias
        %1:Recta
        %2:Espiral
        %3:Curva
        %4:Intestino plano
        %5:Curva pruebas
switch Tipo
    case 1,
        L=PA; %Longitud de la trayectoria
        N=(fix(L/(h1))); %Número de componentes del vector
%número de puntos de la trayectoria % función fix -> parte
entera de un número
        Ang1=PB*2*pi/360; %Ángulo de la trayectoria respecto al
eje x (rad)
        Ang2=PC*2*pi/360; %Ángulo de la trayectoria respecto al
plano xy (rad)
        xi = 0:(L/N)*cos(Ang1)*cos(Ang2):L*cos(Ang1)*cos(Ang2);
%Componentes x de cada punto del vector de puntos
        yi = 0:(L/N)*sin(Ang1)*cos(Ang2):L*sin(Ang1)*cos(Ang2);
%Componentes y de cada punto del vector de puntos
        zi = 0:(L/N)*sin(Ang2):L*sin(Ang2); %Componentes z de
cada punto del vector de puntos
        if Ang1==0
            yi=zeros(1,N+1); %Vector de 1 fila por N+1 columnas
de 0. La recta estaría contenida en el plano xz
```

```
        end
        if Ang2==0
            zi=zeros(1,N+1); %Vector de 1 fila por N+1 columnas
de 0. La recta estaría contenida en el plano xy
        end
    case 2,
        R=PA; %Radio de la espiral
        P=PB; %Paso de la espiral
        ang=atan(P/(2*pi*R)); %pendiente de la espiral
        n=PC; %Número de espiras
        N=fix((n*2*pi*R/cos(ang))/(h1))+1; %Número de puntos del
vector
        xi=zeros(1,N); %inicialización Componentes x de cada
punto del vector de puntos
        yi=zeros(1,N); %inicialización Componentes y de cada
punto del vector de puntos
        zi=zeros(1,N); %inicialización Componentes z de cada
punto del vector de puntos
        for i=2:N;
        xi(i)=R*sin(n*2*pi*((i-1)/(N))); %Componentes x de cada
punto del vector de puntos
        yi(i)=R*cos(n*2*pi*((i-1)/(N)))-R; %Componentes y de
cada punto del vector de puntos
        zi(i)=P*n*((i-1)/N); %Componentes z de cada punto del
vector de puntos
        end
    case 3,
        R=PA; %Radio de la curva
        ang=PB*2*pi/360; %Ángulo girado
        d1=PC; %Distancia en recta entrada
        d2=PD; %Distancia en recta salida
        %N=fix((d2+d1+ang*R/(2*pi))/(h1))+1;%Número de puntos
del vector
        N=fix((d2+d1+ang*R)/(h1))+1%Número de puntos del vector
ZZ
        xi=zeros(1,N); %inicialización Componentes x de cada
punto del vector de puntos
        yi=zeros(1,N); %inicialización Componentes y de cada
punto del vector de puntos
        zi=zeros(1,N); %inicialización Componentes z de cada
punto del vector de puntos
        N1=round(N*(d1/(d2+d1+ang*R)));%Número de puntos del
vector tramo 1
        N2=round(N*(ang*R/(d2+d1+ang*R)));%Número de puntos del
vector tramo 2
        N3=N-(N1+N2);%Número de puntos del vector tramo 3
        for i=1:N1;
        xi(i)=d1*(i-1)/N1; %Componentes x de cada punto del
vector de puntos tramo 1 -> Recto
        yi(i)=0; %Componentes y de cada punto del vector de
puntos tramo 1
        zi(i)=0; %Componentes z de cada punto del vector de
puntos tramo 1
        end
        for i=(N1+1):(N1+N2);
        xi(i)=xi(N1)+R*sin(ang*((i-N1)/(N2))); %Componentes x de
cada punto del vector de puntos tramo 2 -> Curva
        yi(i)=R*cos(ang*((i-N1)/(N2)))-R; %Componentes y de cada
punto del vector de puntos tramo 2
        zi(i)=0; %Componentes z de cada punto del vector de
puntos tramo 2
```

```
        end
        for i=(N1+N2+1):(N);
        xi(i)=xi(N1+N2)+cos(ang)*d2*(i-N1-N2)/N3; %Componentes x
de cada punto del vector de puntos tramo 3
        yi(i)=yi(N1+N2)-sin(ang)*d2*(i-N1-N2)/N3; %Componentes y
de cada punto del vector de puntos tramo 3
        zi(i)=0; %Componentes z de cada punto del vector de
puntos tramo 3
        end
    case 4 % Trayectoria intestino plano
        R1=70;
        R2=70;
        Lrecto=40;
        Lcsig=155;
        Lcdesc=320;
        Lctras=380;
        Lcasc=300;
N=fix((Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4)*2*pi*R2+(1/4)*2*pi
*R1)/(h1));

N1=round(((Lrecto)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4)*2*pi*
R2+(1/4)*2*pi*R1))*N); %La función round, redondea al entero más
cercano en este caso

N2=round((((1/4)*2*pi*R1)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4
)*2*pi*R2+(1/4)*2*pi*R1))*N);

N3=round(((Lcsig)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4)*2*pi*R
2+(1/4)*2*pi*R1))*N);

N4=round((((1/4)*2*pi*R2)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4
)*2*pi*R2+(1/4)*2*pi*R1))*N);

N5=round(((Lcdesc)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4)*2*pi*
R2+(1/4)*2*pi*R1))*N);

N6=round((((1/4)*2*pi*R2)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4
)*2*pi*R2+(1/4)*2*pi*R1))*N);

N7=round(((Lctras)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4)*2*pi*
R2+(1/4)*2*pi*R1))*N);

N8=round((((1/4)*2*pi*R2)/(Lrecto+Lcsig+Lcdesc+Lctras+Lcasc+(3/4
)*2*pi*R2+(1/4)*2*pi*R1))*N);
        N9=N-N1-N2-N3-N4-N5-N6-N7-N8+10;
        for i=1:N1
            xi(i)=Lrecto*i/N1;
            yi(i)=0;
            zi(i)=0;
        end
        for i=(N1+1):(N1+N2)
            xi(i)=xi(N1)+R1*sin((pi/2)*((i-N1)/(N2)));
            yi(i)=R1*cos((pi/2)*((i-N1)/(N2)))-R1;
            zi(i)=0;
        end
        for i=(N1+N2+1):(N1+N2+N3)
            xi(i)=xi(N1+N2);
            yi(i)=yi(N1+N2)-Lcsig*((i-(N1+N2))/N3);
            zi(i)=0;
        end
        for i=(N1+N2+N3+1):(N1+N2+N3+N4)
```

```
            xi(i)=xi(N1+N2+N3)+R2*sin((3*pi/2)+(pi/2)*((i-N1-N2-
N3)/(N4)))+R2;
            yi(i)=yi(N1+N2+N3)-R2*cos((3*pi/2)+(pi/2)*((i-N1-N2-
N3)/(N4)));
            zi(i)=0;
        end
        for i=(N1+N2+N3+N4+1):(N1+N2+N3+N4+N5)
            xi(i)=xi(N1+N2+N3+N4)+Lcdesc*((i-(N1+N2+N3+N4))/N5);
            yi(i)=yi(N1+N2+N3+N4);
            zi(i)=0;
        end
        for i=(N1+N2+N3+N4+N5+1):(N1+N2+N3+N4+N5+N6)
            xi(i)=xi(N1+N2+N3+N4+N5)+R2*sin((pi/2)*((i-N1-N2-N3-
N4-N5)/(N6)));
            yi(i)=yi(N1+N2+N3+N4+N5)-R2*cos((pi/2)*((i-N1-N2-N3-
N4-N5)/(N6)))+R2;
            zi(i)=0;
        end
        for i=(N1+N2+N3+N4+N5+N6+1):(N1+N2+N3+N4+N5+N6+N7)
            xi(i)=xi(N1+N2+N3+N4+N5+N6);
            yi(i)=yi(N1+N2+N3+N4+N5+N6)+Lctras*((i-
(N1+N2+N3+N4+N5+N6))/N7);
            zi(i)=0;
        end
        for i=(N1+N2+N3+N4+N5+N6+N7+1):(N1+N2+N3+N4+N5+N6+N7+N8)
            xi(i)=xi(N1+N2+N3+N4+N5+N6+N7)-
R2*sin((3*pi/2)+(pi/2)*((i-N1-N2-N3-N4-N5-N6-N7)/(N8)))-R2;

yi(i)=yi(N1+N2+N3+N4+N5+N6+N7)+R2*cos((3*pi/2)+(pi/2)*((i-N1-N2-
N3-N4-N5-N6-N7)/(N8)));
            zi(i)=0;
        end
        for i=(N1+N2+N3+N4+N5+N6+N7+N8):(N+10)
            xi(i)=xi(N1+N2+N3+N4+N5+N6+N7+N8)-Lcdesc*((i-
(N1+N2+N3+N4+N5+N6+N7+N8))/N9);
            yi(i)=yi(N1+N2+N3+N4+N5+N6+N7+N8);
            zi(i)=0;
        end
 end
```