

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE
ESCUELA POLITÉCNICA SUPERIOR DE ELCHE
GRADO EN INGENIERÍA ELECTRÓNICA Y
AUTOMÁTICA INDUSTRIAL



CALIBRACIÓN Y CONTROL
CINEMÁTICO DE LA ADHESIÓN DE
UN ROBOT TREPADOR

TRABAJO FIN DE GRADO

Julio - 2021

AUTOR: Marc Fabregat Jaén

DIRECTORES: Adrián Peidró Vidal

Óscar Reinoso García

Índice

1. Introducción	1
1.1. Antecedentes	1
1.2. Objetivos	5
1.3. Estructura de la memoria	5
2. Robot HyReCRO	7
2.1. Cinemática	7
2.1.1. Módulos paralelos	7
2.1.2. Robot completo	11
2.2. Hardware	17
2.3. Garras	18
3. Control de la aproximación mediante sensores de distancia	22
3.1. Obtención de la matriz de diferencia	22
3.1.1. Sensores	23
3.1.2. Filtrado y calibración	29
3.1.3. Representación en tiempo real de la posición y orientación	31
3.2. Algoritmo de aproximación mediante Jacobiana	37
3.2.1. Lectura de los sensores	38
3.2.2. Cálculo de la matriz de diferencia	39
3.2.3. Obtención del vector de diferencia	42
3.2.4. Comprobación de la norma del vector	43
3.2.5. Actualización de la configuración articular	43
3.2.6. Transformación de las coordenadas articulares intermedias	47
3.2.7. Implementación en el código original	50
3.3. Simulación de la aproximación	51
3.4. Resultados	58
4. Calibración en cadena cerrada	61
4.1. Introducción	61
4.2. Método	61
4.2.1. Jacobiana de Identificación	65
4.3. Simulación de la calibración	70
5. Conclusiones y trabajos futuros	75
5.1. Conclusiones	75
5.2. Trabajos futuros	75
A. Códigos	77
A.1. jacob	77
A.2. fk_par	82
A.3. ik_hyrecro	83
A.4. fk_hyrecro	84
A.5. df_dth	85
A.6. df_dqint	86
A.7. B_t	87

A.8. H1	89
A.9. H2	89
A.10.B_t_prima	90

1. Introducción

1.1. Antecedentes

El ámbito de este Trabajo de Fin de Grado (en adelante, TFG) es el de los robots escaladores. Es un ámbito de trabajo que se encuentra en continuo desarrollo y avance durante las últimas décadas. Las aplicaciones de los robots trepadores son muy amplias y diversas en cualquier situación donde la altura suponga un peligro para que realice la tarea una persona. Por ello, una de las aplicaciones más útiles y obvias para este tipo de robots son las tareas de inspección y mantenimiento en lugares de difícil acceso o peligrosos.

Cualquier robot escalador precisa de una forma de adherirse a la superficie por la que se está desplazando. Existen múltiples métodos de adhesión utilizados por robots trepadores. En (Schmidt and Berns, 2013) se hace una clasificación de los robots trepadores en función de la técnica utilizada:

- **Adhesión magnética:** consiste en hacer uso de imanes o electroimanes para pegarse a la superficie. Es una técnica que limita en gran medida las superficies válidas para el robot, puesto que deben ser metálicas. A cambio, se consigue una fuerza de adhesión muy fuerte en una zona muy pequeña. Ejemplos de superficies donde se usa la adhesión magnética son torres de distribución eléctrica, cascos de barcos o tanques de almacenamiento. Suele utilizarse en robots que se desplazan mediante extremidades, deslizamiento o ruedas.
- **Adhesión neumática:** se utilizan ventosas pasivas o cámaras de presión negativas entre otros. Los robots con este tipo de adhesión no son apropiados para desplazarse por superficies pequeñas, ya que para alcanzar la fuerza necesaria de pegado mediante adhesión neumática es necesaria un área de pegado mayor. La ventaja de este tipo de adhesión respecto a otros es la gran versatilidad que tienen en cuanto a superficies donde funcionan, puesto que no se necesita un material específico. Sin embargo, en el caso de las ventosas pasivas, son solo apropiadas para superficies muy lisas, como el cristal. Al ser una técnica parecida a la adhesión magnética en términos generales, también se utiliza en robots que se desplazan mediante extremidades, deslizamiento o ruedas.
- **Adhesión mecánica:** el principio se basa en el rozamiento, ya sea realizando la adhesión mediante agarre o utilizando superficies punzantes. Sus principales ventajas son el reducido consumo de energía y su pertinencia para superficies irregulares a costa de ser más lentos, tener menos maniobrabilidad y soportar menos carga que los principios expuestos anteriormente. Principalmente, la técnica se utiliza en robots con extremidades o en estructuras cilíndricas donde se pueda realizar el agarre, como árboles o tuberías.
- **Adhesión electrostática:** consiste en usar almohadillas electroadhesivas, las cuales comprenden electrodos conductores y substrato aislante, para generar

una fuerza electrostática entre el robot y la superficie. Se trata de un campo muy prometedor gracias a las ventajas que ofrece: seguridad, eficiencia energética y robustez en cuanto a superficies apropiadas. Otro principio de adhesión derivado de este es la adhesión seca, una técnica basada en la utilización de fibras microscópicas para adherirse a la superficie a nivel atómico. Las posibles aplicaciones donde esta tecnología es apropiada siguen siendo exploradas, puesto que es una técnica relativamente nueva.

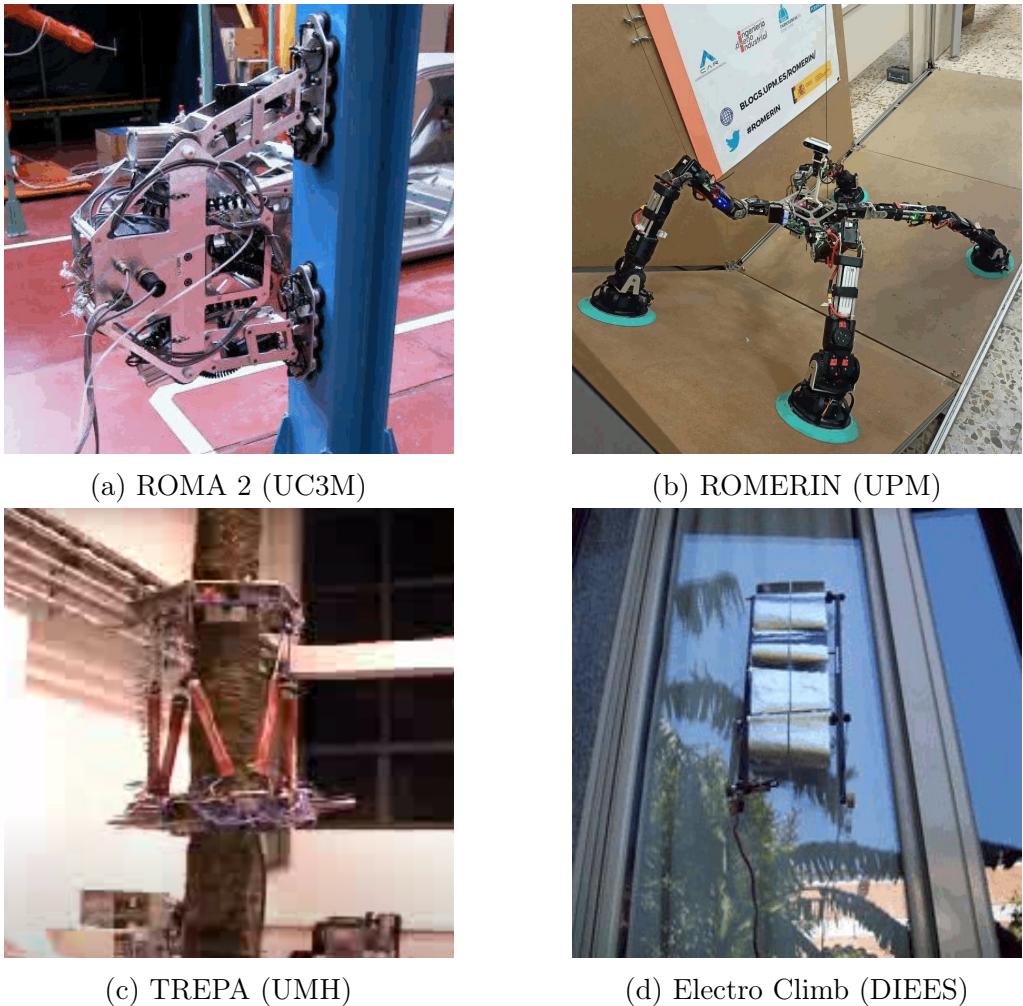


Figura 1: Ejemplos de robots trepadores con adhesión magnética (a), neumática (b), mecánica (c) y electrostática (d).

Los robots trepadores también pueden ser clasificados en función de las características del entorno para el que han sido diseñados. En concreto, este TFG cae dentro del grupo de robots escaladores de estructuras. Se tratan de robots pensados para realizar tareas en vigas interconectadas que forman estructuras tridimensionales como puentes, torres de distribución eléctrica, celosías o esqueletos de una construcción.

Los robots trepadores de estructuras pueden ser clasificados a su vez en 2 grupos (Tavakoli et al., 2011): robots de movimiento continuo o paso a paso.

Los robots de movimiento continuo diseñados para trabajar en este tipo de estructuras generalmente utilizan ruedas como forma de desplazamiento, lo cual permite que sean robots rápidos y simples.

En cambio, los robots de movimiento por pasos son más lentos, complejos y con un control difícil. La estructura de este tipo de robots consiste en 2 garras conectadas a través de una cadena cinemática de varios grados de libertad (en adelante, GDL). El movimiento se consigue mediante la adhesión de una garra y el movimiento y fijación de la garra libre. Cuando la garra libre ha completado su adhesión, los roles se intercambian y la garra previamente fija se mueve para adherirse en su nueva posición. Este tipo de configuración ofrece una maniobrabilidad mayor, lo cual permite evitar obstáculos o realizar una transición entre planos de forma más fácil.

A su vez, los robots trepadores de estructuras con movimiento paso a paso pueden dividirse dependiendo de la arquitectura de la cadena cinemática que une sus 2 garras:

- **Arquitectura serie:** consisten en una cadena cinemática abierta. Generalmente, ofrecen una espacio de trabajo mayor junto con una mayor maniobrabilidad. Un ejemplo de robot escalador con arquitectura tipo serie es el robot ROMA 2 de la UC3M (figura 1a).
- **Arquitectura paralela:** son robots formados por un mecanismo de cadena cerrada donde el efecto final se une a la base por al menos 2 cadenas cinemáticas independientes. Su principal ventaja es el elevado ratio de carga por peso del robot y su rigidez. Por ejemplo, el robot TREPA de la UMH es un robot trepador con arquitectura paralela (figura 1c).
- **Arquitectura híbrida:** son cadenas cinemáticas formadas por mecanismos paralelos conectados en serie. La naturaleza híbrida de este tipo de robots consigue reunir la alta maniobrabilidad y espacio de trabajo de los robots tipo serie, lo cual es útil para explorar estructuras tridimensionales, y la rigidez y capacidad de carga de la arquitectura paralela. El robot HyReCRo, utilizado en este TFG, es un ejemplo de este tipo de arquitectura.

El robot HyReCRo, diseñado por el Grupo de Automatización, Robótica y Visión por Computador de la UMH, es un gran ejemplo para ilustrar la gran variedad de robots trepadores que existen. Se trata de un robot escalador diseñado para la inspección y mantenimiento en estructuras tridimensionales metálicas, como torres de distribución eléctrica o esqueletos de construcciones. Tratándose de un robot escalador de estructuras, se puede clasificar dentro de los robots de movimiento por pasos, ya que está constituido por 2 garras unidas a través de una cadena cinemática. La cadena cinemática que une ambas garras es de arquitectura híbrida. Está formada por 4 módulos paralelos, donde cada pareja de 2 forma cada pata del robot. Ambas patas están unidas en serie a través de una cadera.

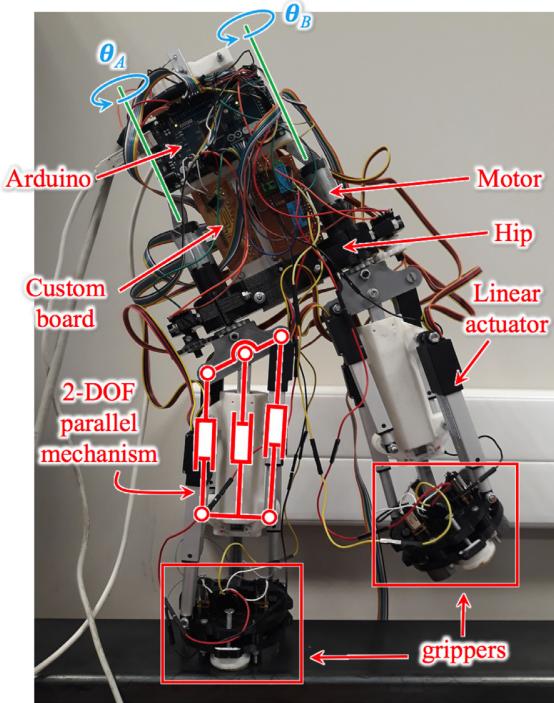


Figura 2: Prototipo del robot HyReCRO presentado en (Peidró et al., 2019).

Cada módulo paralelo se trata de un mecanismo de 2 GDL formado por 2 actuadores lineales (Peidró et al., 2015), lo cual resulta en un total de 8 actuadores lineales en el robot HyReCRO. A su vez, cada unión rotacional de la pata con la cadera está accionada por un motor. Esta configuración resulta en un robot de 10 GDL. Por lo tanto, se trata de un robot con una gran redundancia al superar los 6 GDL necesarios para una tarea en un espacio tridimensional (Peidró et al., 2016).

La técnica de adhesión que se utiliza es magnética al estar diseñado para trabajar en superficies metálicas. En (Peidró et al., 2019) se presentó un diseño de las garras utilizando imanes permanentes comutables y un prototipo funcional del robot. Sin embargo, su control lo hacía un operario de forma manual guiándose por su propia vista, lo cual no es óptimo para un robot con aplicaciones de inspección y mantenimiento en estructuras tridimensionales. Para realizar tareas en entornos de difícil acceso o peligrosos, surge la necesidad de automatizar su funcionamiento.

La automatización de la tarea de adhesión es compleja, puesto que debe realizarse manteniendo una orientación paralela a la superficie de pegado, con una distancia mínima y con suficiente presión, para que la adhesión se produzca de forma segura y sin forzar los actuadores o la estructura del robot. En este TFG se perseguirá automatizar la última etapa de este proceso mediante un algoritmo que, interpretando las medidas tomadas por varios sensores de distancia, realice el ajuste y acercamiento de la garra de forma fina.

El otro objetivo abordado es la calibración del prototipo para identificar los errores geométricos en su estructura debido a defectos de fabricación y mejorar su precisión. Identificando los errores, es posible ajustar el modelo cinemático del

robot para corregirlos. Ambos objetivos están fuertemente relacionados, puesto que una mejora en la precisión del robot derivará en una aproximación y un agarre más preciso.

1.2. Objetivos

El objetivo general de este TFG es la optimización del robot híbrido redundante escalador HyReCRo. Teniendo la premisa original en mente, se proponen 2 objetivos para la optimización del robot:

- Control del acercamiento para la adhesión.
- Calibración del robot.

En relación al primer punto, se persigue automatizar la tarea rutinaria que supone la aproximación de la garra libre a la superficie a la que se va a realizar el pegado por medio de sensores de distancia. El proceso de acercamiento para el pegado es una tarea compleja de realizar manualmente de forma precisa, ya que demanda que la garra esté completamente paralela, a una distancia mínima de la superficie y que aplique la suficiente presión para no forzar los actuadores del robot ni dañar los imanes que permiten la adhesión. La propuesta de este TFG consiste en la adición de 3 sensores de distancia para determinar en todo momento la posición y orientación de la garra respecto a la superficie para realizar un control cinemático realimentado de la adhesión.

La calibración cinemática del robot es el segundo objetivo abordado. Mediante esta técnica, se busca ajustar el modelo cinemático del robot con el fin de mejorar su precisión, resultando en un agarre y movimientos más precisos. Sin embargo, la calibración en cadena abierta convencional supone un coste muy elevado, por lo que se optará por realizar una calibración cinemática en cadena cerrada del robot. De esta forma, no es necesario equipo adicional y de forma autónoma se podrán identificar los errores a corregir a partir de la calibración y ajustar el modelo cinemático del robot para corregir los errores que se han identificado en la calibración.

Ambos objetivos están estrechamente relacionados, puesto que una mejora en la precisión del robot, fruto de la calibración realizada, supondrá una mejora en el control de la aproximación de la garra para la adhesión.

1.3. Estructura de la memoria

Este TFG se ha estructurado de la siguiente forma:

En el capítulo 2 se expone el robot HyReCRo y el prototipo utilizado en este TFG. Primero, se habla tanto de la cinemática de los 4 módulos paralelos que

componen el robot, como de la cinemática del equivalente serie del robot completo. Seguidamente, se describe el hardware que compone el prototipo del robot HyRe-CRo utilizado. Y para acabar, se muestra el diseño de las garras que permiten la adhesión del robot.

El capítulo 3 es el primer capítulo de desarrollo del trabajo. En él, se diseña un algoritmo que permite el acercamiento de la garra libre del robot para su adhesión a una superficie utilizando sensores de distancia. Además, se programa una herramienta que permite visualizar en tiempo real la posición y orientación de la garra respecto a la superficie cercana. Finalmente, se realizan 2 ejemplos de simulación del algoritmo y un experimento con el prototipo.

El segundo objetivo del TFG se desarrolla en el capítulo 4. Se expone un método de calibración en cadena cerrada con el fin de mejorar la precisión del robot y se realiza una simulación del mismo.

Por último, en el capítulo 5, se analizan los resultados obtenidos y se formulan las conclusiones del trabajo. Finalmente, derivados de las conclusiones obtenidas, se proponen trabajos futuros para desarrollar, tanto a corto como a largo plazo.

2. Robot HyReCRo

La premisa original de este TFG consistía en la optimización de un prototipo del robot HyReCRo. Por lo cual, todos los objetivos perseguidos, las soluciones propuestas y su programación se harán sobre el robot HyReCRo o tomando como referencia su modelo geométrico y cinemático.

El acrónimo HyReCRo (*Hybrid Redundant Climbing Robot*) viene dado por las características principales del robot en cuanto a su estructura, cinemática y función respectivamente. El robot fue diseñado por el Grupo de Automatización, Robótica y Visión por Computador de la UMH, y su cinemática, diseño, geometría, espacio de trabajo, singularidades y garras entre otros temas han sido ampliamente expuestos en múltiples artículos, algunos de los cuales se citarán seguidamente.

En este capítulo se discutirá la cinemática y estructura mecánica, tanto de los módulos paralelos (Peidró et al., 2015) como del robot completo (Peidró et al., 2016). Luego, se indicará el hardware del que se compone el robot. Y finalmente, se expondrá el diseño de las garras que permiten la adherencia del robot durante su escalada (Peidró et al., 2019).

2.1. Cinemática

El robot está formado por 2 patas idénticas unidas a su cadera a través de 2 articulaciones rotacionales. A su vez, cada pata está compuesta por 2 módulos paralelos unidos a un mismo cuerpo central. Cada uno de los 2 módulos paralelos que forman cada una de las patas robot consiste en 2 actuadores lineales que unen una plataforma con el cuerpo común a 2 módulos paralelos diferentes. Se trata de un robot de arquitectura híbrida al combinar mecanismos paralelos conectados en serie.

Esta configuración da como resultado un robot de 10 GDL. Para realizar una tarea en un espacio tridimensional son necesarios 6 GDL, por lo que se trata de un robot redundante. La principal ventaja de los robots redundantes es la capacidad de poder realizar movimientos en las articulaciones sin cambiar la posición u orientación del efecto final. La propiedad de redundancia permite un gran abanico de posibilidades al robot: desde sortear obstáculos hasta permitir la calibración del robot sin necesitar medios externos, como se explicará en el capítulo 4.

2.1.1. Módulos paralelos

Como ya se ha explicado brevemente, cada una de las patas del robot está formada por 2 módulos paralelos unidos a un mismo cuerpo central. Los 2 actuadores lineales que forman cada módulo unen la plataforma con el cuerpo central.

Como se aprecia en la figura 3, los módulos paralelos son mecanismos planares

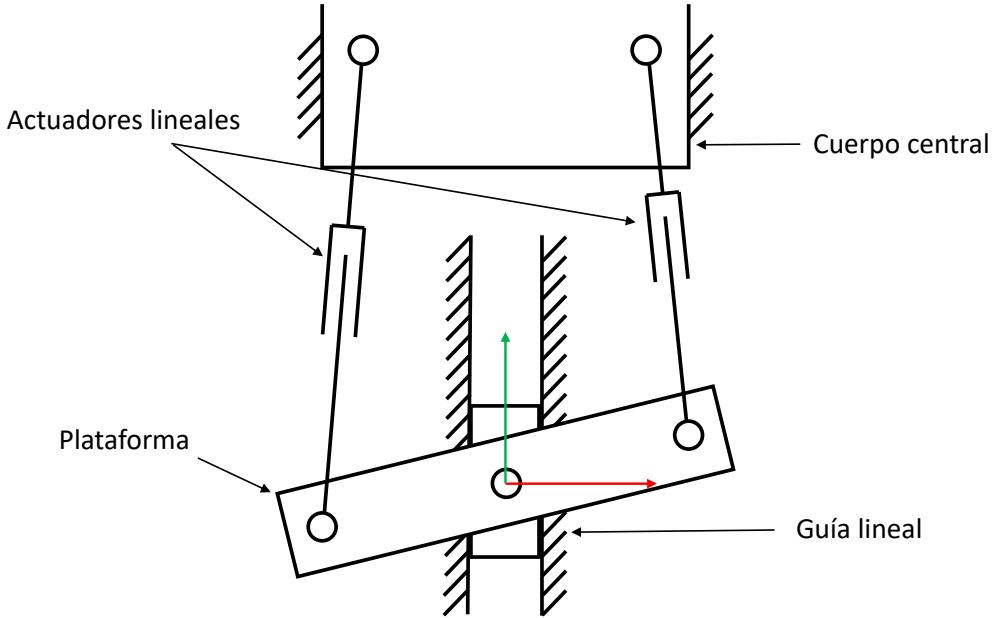


Figura 3: Representación del mecanismo de 2 GDL de los módulos paralelos.

de 2 GDL. Los 2 actuadores lineales están unidos al cuerpo central y a la plataforma mediante articulaciones rotacionales. Además, el cuerpo y la plataforma están unidos a través de una guía que restringe el movimiento de la plataforma respecto al cuerpo central únicamente a movimientos lineales en su eje.

De esta forma, la posición y orientación de la plataforma se puede modificar mediante los actuadores lineales (u_{ij} y v_{ij}) y se puede describir con 2 parámetros, posición en el eje y (y_{ij}) y giro respecto al eje z (φ_{ij}).

El cuerpo central, que actúa como base del módulo paralelo, tiene una longitud b desde el eje y del sistema de referencia hasta cada articulación rotacional. La plataforma móvil del módulo tiene una longitud p desde el eje de la guía hasta cada articulación.

Cinemática directa

En la figura 4 se expone la cinemática del módulo paralelo número i ($i \in \{1, 2\}$) de la pata j ($j \in \{A, B\}$).

La cinemática directa de los módulos consiste en calcular la posición y_{ij} y la orientación φ_{ij} en función de las longitudes de los actuadores u_{ij} y v_{ij} . A partir de la figura, se pueden extraer las relaciones:

$$(p \cos \varphi_{ij} - b)^2 + (y_{ij} - p \sin \varphi_{ij})^2 = u_{ij}^2 \quad (1)$$

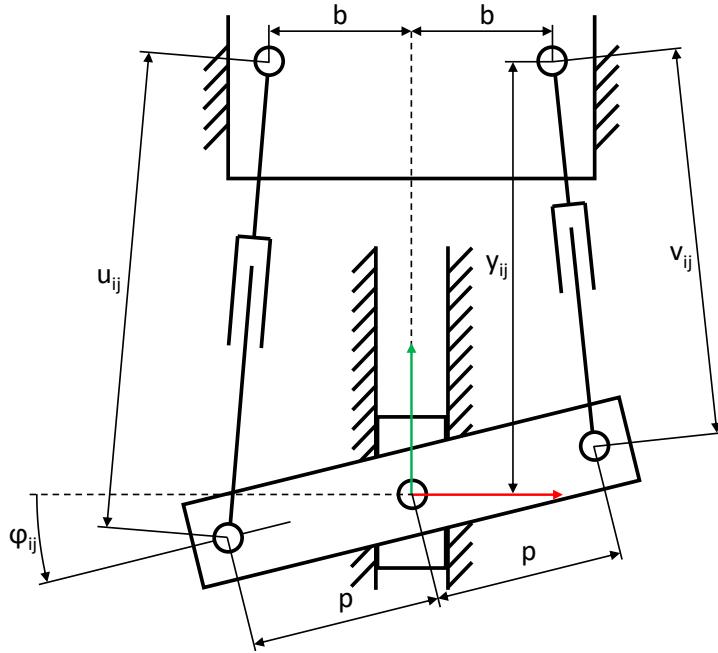


Figura 4: Cinemática de los módulos paralelos.

$$(p \cos \varphi_{ij} - b)^2 + (y_{ij} + p \sin \varphi_{ij})^2 = v_{ij}^2 \quad (2)$$

Combinar estas ecuaciones genera un nuevo sistema equivalente. Sumando las ecuaciones (1) y (2) se obtiene la ecuación (3), mientras que restar la ecuación (2) a (1) resulta en la ecuación (4):

$$4 b p \cos \varphi_{ij} = 2 y_{ij}^2 + 2 b^2 + 2 p^2 - u_{ij}^2 - v_{ij}^2 \quad (3)$$

$$4 y_{ij} p \sin \varphi_{ij} = v_{ij}^2 - u_{ij}^2 \quad (4)$$

De la ecuación (3) se puede despejar $\cos \varphi_{ij}$:

$$\cos \varphi_{ij} = \frac{2 y_{ij}^2 + 2 b^2 + 2 p^2 - u_{ij}^2 - v_{ij}^2}{4 b p} \quad (5)$$

Por otro lado, elevando al cuadrado la ecuación (4) se obtiene:

$$16 y_{ij}^2 p^2 (1 - \cos^2 \varphi_{ij}) = (v_{ij}^2 - u_{ij}^2)^2 \quad (6)$$

Finalmente, sustituir (5) en (6):

$$\Upsilon_{ij}^3 + k_2^{ij}\Upsilon_{ij}^2 + k_1^{ij}\Upsilon_{ij} + k_0^{ij} = 0 \quad (7)$$

donde:

$$\Upsilon_{ij} = y_{ij}^2 \quad (8)$$

$$k_2^{ij} = 2b^2 + 2p^2 - u_{ij}^2 - v_{ij}^2 \quad (9)$$

$$k_1^{ij} = \left[(b+p)^2 - \frac{u_{ij}^2 + v_{ij}^2}{2} \right] \left[(b-p)^2 - \frac{u_{ij}^2 + v_{ij}^2}{2} \right] \quad (10)$$

$$k_0^{ij} = \frac{b^2(u_{ij} + v_{ij})^2(u_{ij} - v_{ij})^2}{4} \quad (11)$$

La ecuación (7) es una función de Υ_{ij} que siempre tiene 3 raíces, de las cuales 2 pueden ser complejas. Para la raíz real positiva de mayor valor se calcula y_{ij} con:

$$y_{ij} = \pm\sqrt{\Upsilon_{ij}} \quad (12)$$

De las 2 soluciones se escoge la positiva y mediante la ecuación (5) se calcula el valor de $\cos \varphi_{ij}$. Dependiendo de la raíz del polinomio de la ecuación (7) utilizada y del signo escogido en (12) habrán 4 soluciones posibles, correspondientes a 4 regiones diferentes expuestas en (Peidró et al., 2016). Los valores escogidos corresponden a la región R_1 del artículo citado.

A partir de la ecuación (4) se puede despejar $\sin \varphi_{ij}$:

$$\sin \varphi_{ij} = \frac{v_{ij}^2 - u_{ij}^2}{4 y_{ij} p} \quad (13)$$

Conociendo $\sin \varphi_{ij}$ y $\cos \varphi_{ij}$ se puede calcular el valor de φ_{ij} inequívocamente con la función atan2:

$$\varphi_{ij} = \text{atan2}(\sin \varphi_{ij}, \cos \varphi_{ij}) \quad (14)$$

La función atan2, a diferencia de la función trigonométrica arctan convencional, devuelve el ángulo a partir de 2 parámetros de entrada. La función arctan es incapaz de determinar el cuadrante del ángulo únicamente a partir de su valor tangente. Para conocer el cuadrante concreto del ángulo, se utiliza la función atan2, que puede definirse de la siguiente manera:

$$\text{atan2}(y, x) = \begin{cases} 2 \arctan \left(\frac{y}{\sqrt{x^2 + y^2} + x} \right) & \text{si } x > 0 \text{ o } y \neq 0, \\ \pi & \text{si } x < 0 \text{ o } y = 0, \\ \text{indefinido} & \text{si } x = 0 \text{ o } y. \end{cases} \quad (15)$$

De esta forma, quedan completamente definidos los valores de y_{ij} y φ_{ij} a partir de los valores de las longitudes u_{ij} y v_{ij} .

Cinemática inversa

Por otro lado, la cinemática inversa consiste en expresar los valores de los actuadores lineales u_{ij} y v_{ij} en función de la posición y orientación de la plataforma y_{ij} y φ_{ij} . En este caso, se puede obtener la relación sencillamente a partir de las ecuaciones (1) y (2):

$$u_{ij} = \sqrt{(p \cos \varphi_{ij} - b)^2 + (y_{ij} - p \sin \varphi_{ij})^2} \quad (16)$$

$$v_{ij} = \sqrt{(p \cos \varphi_{ij} - b)^2 + (y_{ij} + p \sin \varphi_{ij})^2} \quad (17)$$

2.1.2. Robot completo

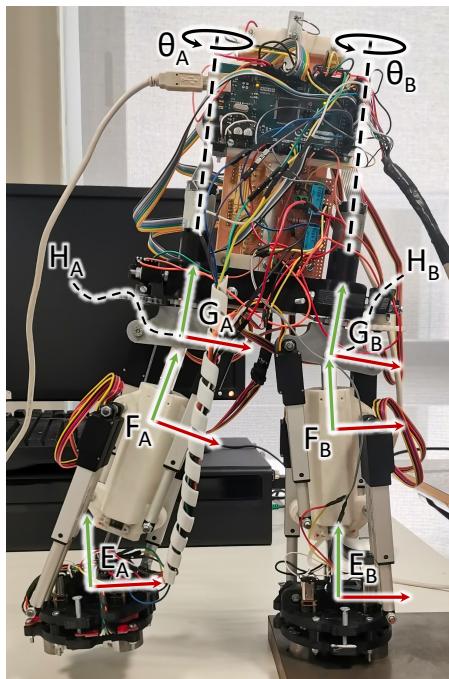


Figura 5: Prototipo del robot con los sistemas de referencia identificados.

El robot completo está formado por 2 patas, cada una unida a la cadera a través de una articulación rotacional (θ_A y θ_B). Cada pata está a su vez formada por 2 módulos paralelos conectados en serie a través de un cuerpo central, que actúa como base de ambos. La plataforma del módulo $1j$ actúa como pie del robot y es donde se fijan las garras que se discutirán posteriormente. La plataforma del módulo $2j$ corresponde a la parte superior de la pata y se conecta a la cadera a través la articulación θ_j .

La cadera está dimensionada en función del parámetro t , distancia entre ejes de las articulaciones rotacionales de la cadera. Los cuerpos centrales también tienen sus dimensiones determinadas por el valor de la variable geométrica h , que indica la longitud entre las bases de ambos módulos paralelos de cada pata.

Cinemática directa

La cinemática directa del robot completo consiste en calcular la posición y orientación del efecto final (pie B) relativas a la base (pie A) en función de las rotaciones de la cadera (θ_A y θ_B) y de las rotaciones y traslaciones de los módulos paralelos (y_{ij} y φ_{ij}), también llamadas variables articulares intermedias. La pose¹ del robot se expresa en una matriz de transformación ${}^A\mathbf{T}_B$. Las matrices de transformación son matrices homogéneas de dimensiones 4×4 , donde se indica la posición y orientación del sistema de referencia B relativo a A . La submatriz 3×3 correspondiente a las 3 primeras filas y columnas de la matriz de transformación indica la rotación, mientras que la posición o traslación viene dada por los 3 primeros elementos de la cuarta columna de la matriz ${}^A\mathbf{T}_B$.

A partir de la figura 6 se puede relacionar la posición y orientación de la cadera en relación al pie de la pata j :

$$\begin{aligned} {}^{E_j}\mathbf{T}_{H_j} &= {}^{E_j}\mathbf{T}_{F_j} {}^{F_j}\mathbf{T}_{G_j} {}^{G_j}\mathbf{T}_{H_j} \\ &= \begin{bmatrix} \cos \varphi_{1j} & \sin \varphi_{1j} & 0 & y_{1j} \sin \varphi_{1j} \\ -\sin \varphi_{1j} & \cos \varphi_{1j} & 0 & y_{1j} \cos \varphi_{1j} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \varphi_{2j} & -\sin \varphi_{2j} & 0 & 0 \\ \sin \varphi_{2j} & \cos \varphi_{2j} & 0 & y_{2j} - h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta_j & 0 & \sin \theta_j & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta_j & 0 & \cos \theta_j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta_j \cos \Phi_j & \sin \Phi_j & \sin \theta_j \cos \Phi_j & y_j \sin \varphi_{1j} \\ -\cos \theta_j \sin \Phi_j & \cos \Phi_j & -\sin \theta_j \sin \Phi_j & y_j \cos \varphi_{1j} \\ -\sin \theta_j & 0 & \cos \theta_j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{18}$$

donde $y_j = y_{1j} + y_{2j} - h$ y $\Phi_j = \varphi_{1j} - \varphi_{2j}$.

¹Término utilizado en robótica y visión por computador. Determina la posición y orientación relativas a un sistema de referencia.

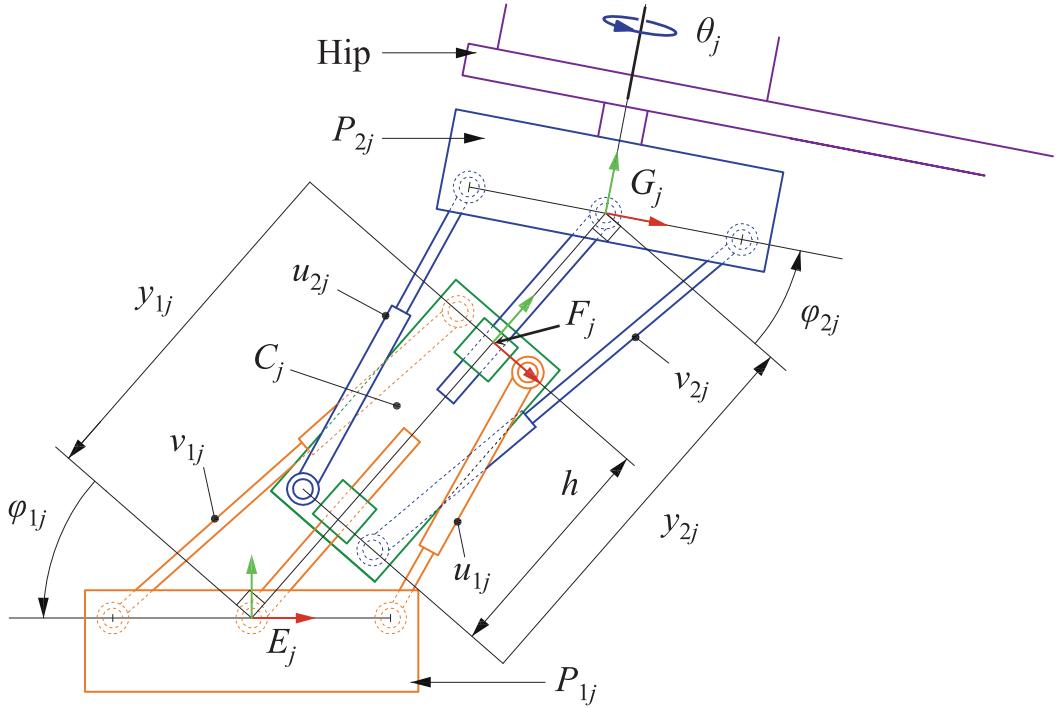


Figura 6: Cinemática de la pata j . Figura tomada de (Peidró et al., 2016) con permiso.

La relación entre los sistemas de referencia de la cadera corresponde a una traslación de distancia t en su eje \mathbf{x} :

$${}^{H_A}\mathbf{T}_{H_B} = \begin{bmatrix} \mathbf{I}_{3 \times 3} & [t, 0, 0]^T \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \quad (19)$$

La matriz de transformación del pie respecto a la cadera de la pata j se puede obtener invirtiendo (18):

$${}^{H_j}\mathbf{T}_{E_j} = {}^{E_j}\mathbf{T}_{H_j}^{-1} \quad (20)$$

De esta forma, se puede obtener la matriz de transformación que expresa la pose del pie B relativo al pie A con el producto de las 3 matrices de transformación calculadas:

$${}^{E_A}\mathbf{T}_{E_B} = {}^{E_A}\mathbf{T}_{H_A} {}^{H_A}\mathbf{T}_{H_B} {}^{H_B}\mathbf{T}_{E_B} \quad (21)$$

Resultando en una matriz de transformación homogénea donde se recoge la orientación en la submatriz de rotación ${}^{E_A}\mathbf{R}_{E_B}$ y la traslación en el vector posición $\mathbf{p} = [p_x, p_y, p_z]^T$:

$${}^{E_A}\mathbf{T}_{E_B} = \begin{bmatrix} {}^{E_A}\mathbf{R}_{E_B} & \mathbf{p} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \quad (22)$$

El producto de la ecuación (21) resulta en el siguiente vector posición:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = y_A \begin{bmatrix} \sin \varphi_{1A} \\ \cos \varphi_{1A} \\ 0 \end{bmatrix} + y_B \begin{bmatrix} -\cos \Theta \cos \Phi_A \sin \varphi_{2B} - \sin \Phi_A \cos \varphi_{2B} \\ \cos \Theta \sin \Phi_A \sin \varphi_{2B} - \cos \Phi_A \cos \varphi_{2B} \\ \sin \Theta \sin \varphi_{2B} \end{bmatrix} + t \begin{bmatrix} \cos \theta_A \cos \Phi_A \\ -\cos \theta_A \sin \Phi_A \\ -\sin \theta_A \end{bmatrix} \quad (23)$$

donde $\Theta = \theta_A - \theta_B$.

Mientras que la matriz de rotación equivale a:

$${}^{E_A}\mathbf{R}_{E_B} = \begin{bmatrix} \sin \Phi_A \sin \Phi_B + \cos \Theta \cos \Phi_A \cos \Phi_B & \sin \Phi_A \cos \Phi_B - \cos \Theta \cos \Phi_A \sin \Phi_B & \sin \Theta \cos \Phi_A \\ \cos \Phi_A \sin \Phi_B - \cos \Theta \sin \Phi_A \cos \Phi_B & \cos \Phi_A \cos \Phi_B + \cos \Theta \sin \Phi_A \sin \Phi_B & -\sin \Theta \sin \Phi_A \\ -\sin \Theta \cos \Phi_B & \sin \Theta \sin \Phi_B & \cos \Theta \end{bmatrix} \quad (24)$$

Cinemática inversa

La cinemática inversa del robot consiste en, a partir de una pose conocida, calcular las coordenadas articulares que consiguen esa pose concreta. En el caso de robots no redundantes, existen finitas soluciones. Para el robot HyReCRO, en cambio, al ser redundante, existirán infinitas configuraciones articulares que consigan una misma pose.

A partir de la matriz de rotación (24) y suponiendo que los ejes **z** de los pies no son paralelos ni antiparalelos ($\mathbf{R}_{33} \neq \pm 1$) se puede calcular directamente el valor de Θ a partir de \mathbf{R}_{33} :

$$\Theta = \pm \text{arc cos } \mathbf{R}_{33} \quad (25)$$

De las 2 soluciones posibles soluciones se utilizará únicamente la positiva.

A partir de los elementos restantes de la tercera columna de la matriz de rotación y conociendo el valor de Θ , se puede extraer:

$$\cos \Phi_A = \frac{\mathbf{R}_{13}}{\sin \Theta} \quad (26)$$

$$\sin \Phi_A = -\frac{\mathbf{R}_{23}}{\sin \Theta} \quad (27)$$

Conociendo $\sin \Phi_A$ y $\cos \Phi_A$ se puede calcular inequívocamente el valor de Φ_A mediante la función atan2, definida en (15):

$$\Phi_A = \text{atan2}(\sin \Phi_A, \cos \Phi_A) \quad (28)$$

De la misma forma, a partir de los elementos restantes de la tercera fila de la matriz de rotación, conociendo el valor de Θ y mediante la función `atan2`, se puede calcular el valor de Φ_B :

$$\cos \Phi_B = -\frac{\mathbf{R}_{31}}{\sin \Theta} \quad (29)$$

$$\sin \Phi_B = \frac{\mathbf{R}_{32}}{\sin \Theta} \quad (30)$$

$$\Phi_B = \text{atan2}(\sin \Phi_B, \cos \Phi_B) \quad (31)$$

Como ya se ha comentado, al ser un robot redundante, existirán infinitas configuraciones articulares que resulten en una misma pose. Por lo tanto, habrá que dar valores a ciertas coordenadas articulares para obtener una solución única. Concretamente, el equivalente serie del robot HyReCRo que se está utilizando en el modelo tiene 8 GDL, por lo que habrá que dar valores a 2 variables para obtener una solución para una tarea que requiere 6 GDL. Se le darán valores a las variables articulares intermedias φ_{2B} e y_B . De esta forma, se extrae directamente el valor de θ_A a partir del elemento p_z del vector de posición (23):

$$\theta_A = \begin{cases} \arcsin\left(\frac{y_B \sin \Theta \sin \varphi_{2B} - p_z}{t}\right) \\ \pi - \arcsin\left(\frac{y_B \sin \Theta \sin \varphi_{2B} - p_z}{t}\right) \end{cases} \quad (32)$$

Una vez más, se obtienen 2 valores de θ_A en diferentes cuadrantes. Se utilizará para seguir con los cálculos el valor correspondiente al primer o cuarto cuadrante. De esta forma, se está siguiendo 1 de las 4 posibles ramas de soluciones que se bifurcan en las ecuaciones (25) y (32).

Conociendo θ_A y Θ se puede obtener θ_B :

$$\theta_B = \theta_A - \Theta \quad (33)$$

A partir de los elementos p_x y p_y del vector posición se puede obtener el siguiente sistema despejando el primer sumando de la ecuación (23):

$$\left. \begin{aligned} y_A \sin \varphi_{1A} &= p_x + y_B (\cos \Theta \cos \Phi_A \sin \varphi_{2B} + \sin \Phi_A \cos \varphi_{2B}) - t \cos \theta_A \cos \Phi_A \\ y_A \cos \varphi_{1A} &= p_y + y_B (\cos \Phi_A \cos \varphi_{2B} - \cos \Theta \sin \Phi_A \sin \varphi_{2B}) + t \cos \theta_A \sin \Phi_A \end{aligned} \right\} \quad (34)$$

Para facilitar la legibilidad, se sustituirá el lado derecho de las ecuaciones (34) por las variables K_1 y K_2 :

$$\left. \begin{array}{l} y_A \sin \varphi_{1A} = K_1 \\ y_A \cos \varphi_{1A} = K_2 \end{array} \right\} \quad (35)$$

De donde se puede extraer el valor de φ_{1A} mediante la función atan2:

$$\varphi_{1A} = \text{atan2}(K_1, K_2) \quad (36)$$

Y de donde también se puede calcular y_A :

$$y_A = \sqrt{K_1^2 + K_2^2} \quad (37)$$

Finalmente, se pueden calcular los ángulos φ_{ij} restantes:

$$\varphi_{2A} = \varphi_{1A} - \Phi_A \quad (38)$$

$$\varphi_{1B} = \Phi_B + \varphi_{2B} \quad (39)$$

Esta solución a la cinemática inversa, sin embargo, solo es válida cuando los ejes **z** de los pies no son paralelos ni antiparalelos; es decir, cuando $\mathbf{R}_{33} \neq \pm 1$.

En el caso de que los ejes **z** de los pies sean paralelos o antiparalelos ($\mathbf{R}_{33} = \pm 1$), los elementos restantes de la tercera fila y columna de la matriz **R** serán nulos, por lo que el método expuesto no será válido. A continuación, se propone otra solución a la cinemática inversa cuando $\mathbf{R}_{33} = \pm 1$.

El valor de Θ se puede obtener de la misma forma:

$$\Theta = \text{arc cos } \mathbf{R}_{33} \quad (40)$$

De aquí en adelante los 2 métodos difieren. En este caso, además de dar valores a φ_{2B} e y_B , será necesario dar valor a la variable φ_{1B} para obtener una solución única. De esta forma, se puede calcular la variable Φ_B :

$$\Phi_B = \varphi_{1B} - \varphi_{2B} \quad (41)$$

Al ser los elementos restantes de la tercera fila y columna de la matriz **R** nulos, se deberán usar los elementos de las 2 primeras filas y columnas para obtener el

valor de Φ_A . Concretamente, a partir de los elementos \mathbf{R}_{12} y \mathbf{R}_{22} y mediante las razones trigonométricas de la suma y diferencia de ángulos se puede obtener:

$$\sin(\Phi_A - \mathbf{R}_{33}\Phi_B) = \sin\Phi_A \cos\Phi_B - \mathbf{R}_{33} \cos\Phi_A \sin\Phi_B = \mathbf{R}_{12} \quad (42)$$

$$\cos(\Phi_A - \mathbf{R}_{33}\Phi_B) = \cos\Phi_A \cos\Phi_B + \mathbf{R}_{33} \sin\Phi_A \sin\Phi_B = \mathbf{R}_{22} \quad (43)$$

Cabe recordar que, al ser los ejes **z** de los pies paralelos o antiparalelos, \mathbf{R}_{33} siempre será 1 o -1 , por lo que solo afectará en el signo de las ecuaciones anteriores y siempre se cumplirá la propiedad.

Una vez más, se puede usar la función atan2 definida en (15) y despejar Φ_A :

$$\Phi_A = \text{atan2}(\mathbf{R}_{12}, \mathbf{R}_{22}) + \mathbf{R}_{33} \Phi_B \quad (44)$$

El resto de variables articulares se calculan de la misma forma que en el método utilizado cuando los pies no son paralelos ni antiparalelos. Se calcula θ_A como en la ecuación (32). θ_B se obtiene de (33). Se obtiene el sistema de (34) y se calcula φ_{1A} en (36) y y_A en (37). Y finalmente, se calcula φ_{2A} de (38).

2.2. Hardware

En esta sección se discutirá el hardware del cual se compone el prototipo del robot HyReCRo con el que se ha trabajado.

Dependiendo de las variables geométricas expuestas en la sección anterior, el espacio de trabajo del robot se verá afectado. El prototipo ha sido diseñado siguiendo las siguientes variables geométricas: $t = 110$, $h = 70$, $b = 25$, $p = 31.5$ (valores en mm).

El robot está compuesto casi en su totalidad por partes impresas en 3D usando plástico PLA. Las partes restantes están fabricadas en aluminio.

La cadera del robot, dimensionada en función de la variable t , es el eslabón que une las 2 patas del robot a través de 2 articulaciones rotacionales (θ_A y θ_B) accionadas por motores DC. Los motores utilizados son del fabricante Maxon, modelo A-Max 22. Además, cada motor lleva incorporada una reductora con un factor de reducción de 590:1 con el fin generar el par necesario para mover las patas.

Cada pata está compuesta por 2 módulos paralelos conectados en serie con base común. El cuerpo central que hace de base de ambos módulos está dimensionado por la longitud h . Cada módulo paralelo, cuya base y plataforma están dimensionadas por las variables b y p respectivamente, está formado por 2 actuadores lineales. Los actuadores utilizados son del fabricante Actuonix, modelo L12-50-210-12-P.

Para hacer el control del robot se ha utilizado un controlador Arduino Mega 2560. El control que se hace sobre los actuadores, pasa a través de una placa diseñada para amplificar y filtrar las señales entre el Arduino y los actuadores. Tanto la placa de Arduino como la diseñada se encuentran montadas en la cadera del robot.

El control manual del robot se hace a través de un mando con una cruceta y 6 botones. A través de la cruceta se controla la translación que hace el efecto final a través del plano formado por los ejes **x** e **y** de la base (H_j en la figura 5). Mientras que a través de 2 de los 6 botones se controla el giro que se hace en el eje **z**. Estos 3 movimientos (traslación en los ejes **x** e **y** y giro en el eje **z**) se computan siguiendo la relación $\Delta t = J \Delta q$.

El control de los giros θ_A y θ_B de la cadera se realiza de forma independiente utilizando los 4 botones restantes.

El control de los motores que accionan los 3 imanes de cada garra se realiza mediante botones montados directamente en la cadera. Junto a ellos hay otro botón que cambia el sistema de referencia del robot dependiendo de la garra fija que actúa como base.

2.3. Garras

La función principal del robot HyReCRo es la escalada de estructuras metálicas tridimensionales con el fin de realizar tareas de mantenimiento e inspección. En cada paso de la escalada, el robot debe adherir el pie libre a la superficie y liberar el pie fijo para que estos intercambien los roles de base y efecto final. Para ello, se hace uso de garras instaladas en cada pie con la capacidad de aguantar los grandes pares de fuerza que genera el peso del robot. En esta sección se expondrá el diseño de las garras que aloja el prototipo del robot HyReCRo.

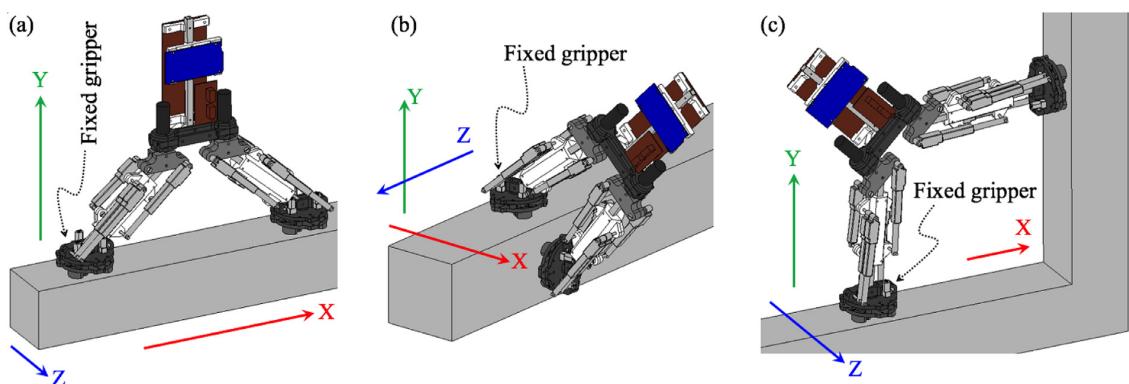


Figura 7: Posturas básicas durante el movimiento: desplazamiento longitudinal (a), transición exterior entre superficies (b), transición interior entre superficies (c). Figura tomada de (Peidró et al., 2019) con permiso.

Las garras diseñadas deben ser capaces de aguantar los esfuerzos generados en todas las posibles posturas del robot. Para ello, se consideran 3 posturas básicas

usadas en la exploración de estructuras (figura 7). A su vez, en cada postura, se consideran 6 casos diferentes dependiendo del eje y sentido en el que actúa la gravedad. Esto resulta en un total de 18 escenarios considerados para el diseño de las garras.

El diseño presentado en (Peidró et al., 2019) consiste en 3 imanes permanentes commutables² distribuidos en una circunferencia con un desfase de 120° entre ellos. 2 objetivos son perseguidos mediante el diseño: evitar el desprendimiento y el deslizamiento de la garra.

El desprendimiento de la garra ocurre cuando uno de los 3 imanes se desprende a causa de que la posición **p** del Punto de Momento Cero sale de la envoltura convexa del área de contacto de la garra con la superficie. El desarrollo de las ecuaciones y consideración de los peores casos de desprendimiento del robot en los 18 escenarios, así como los parámetros escogidos para dimensionar el diseño, se discuten en los capítulos 4 y 5 de (Peidró et al., 2019). Como solución al problema del desprendimiento se propuso el siguiente diseño, donde en una vista explosionada se puede apreciar la pareja de imanes commutables en su alojamiento, los motores y piezas que los accionan y la estructura donde se sitúan los 3 imanes:

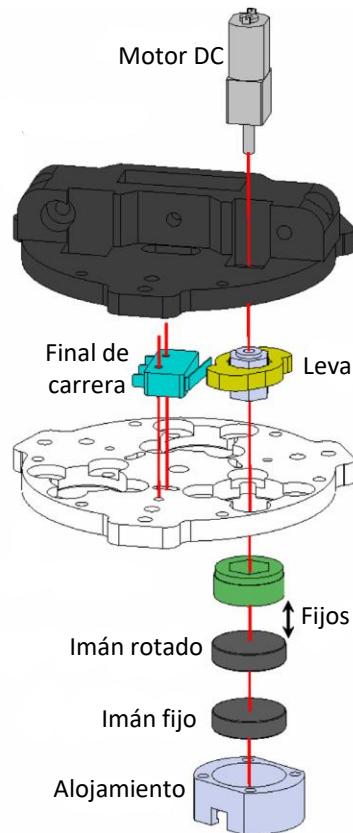


Figura 8: Vista explosionada del diseño de la garra. Figura tomada de (Peidró et al., 2019) con permiso.

El accionamiento de los imanes, tanto para activar el campo magnético exterior

²Imanes que permiten que el campo magnético exterior pueda ser activado o no. Su activación se hace mediante la rotación de los polos de un imán relativo a otro.

como para desactivarlo, se hace mediante el circuito de la figura 9. Cuando el circuito recibe una señal de entrada, acciona los 3 motores que giran los imanes 180° hasta que cada leva acciona su final de carrera. Cuando se recibe un pulso de entrada enviado desde el Arduino, se accionan los 3 motores que giran los imanes. Cuando cada uno de los imanes ha girado 180°, cada leva acciona su final de carrera para conseguir que cada imán termine su rotación independientemente de los otros.

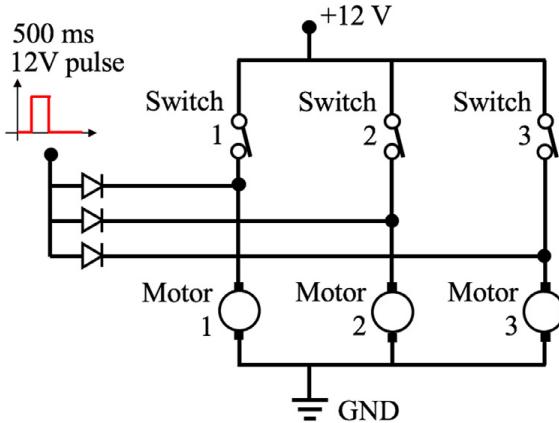


Figura 9: Circuito que acciona los imanes. Figura tomada de (Peidró et al., 2019) con permiso.

Esta solución, sin embargo, no previene el deslizamiento de la garra cuando la fuerza de gravedad actúa en los ejes **x** o **z** de la garra fija. Para solucionar el problema del deslizamiento, se diseñaron unos accesorios de fricción para las garras que evitan el deslizamiento de esta.

En el capítulo 6 de (Peidró et al., 2019) se discutió cuál es el coeficiente de fricción necesario para que la garra fija no se deslice a causa de la fuerza de la gravedad. Se determinó que para los 18 escenarios propuestos, la fuerza de la gravedad causa más deslizamiento en la garra cuando actúa en el eje **z**. Concretamente, cuando el robot está realizando un movimiento de desplazamiento longitudinal y la gravedad actúa en el sentido negativo del eje **z**, el coeficiente de fricción necesario para que no ocurra el deslizamiento es 0.3392.

El accesorio de fricción diseñado consiste en 3 almohadillas circulares de Vytaflex® 30 montadas en una estructura que permite que se puedan añadir y quitar de la garra con facilidad. Las almohadillas están distribuidas de la misma forma que los imanes, desfasadas 120° entre ellas. El accesorio genera un coeficiente de fricción de 0.438, mayor al necesario de 0.3392.

La altura del accesorio de fricción puede ser regulada mediante 3 tornillos para optimizar la fricción como se muestra en la siguiente figura:

Finalmente, se muestra una imagen de la garra completa con el accesorio de fricción añadido, tal y como se presentó en (Peidró et al., 2019), para su comparación con el nuevo diseño propuesto en el siguiente capítulo:

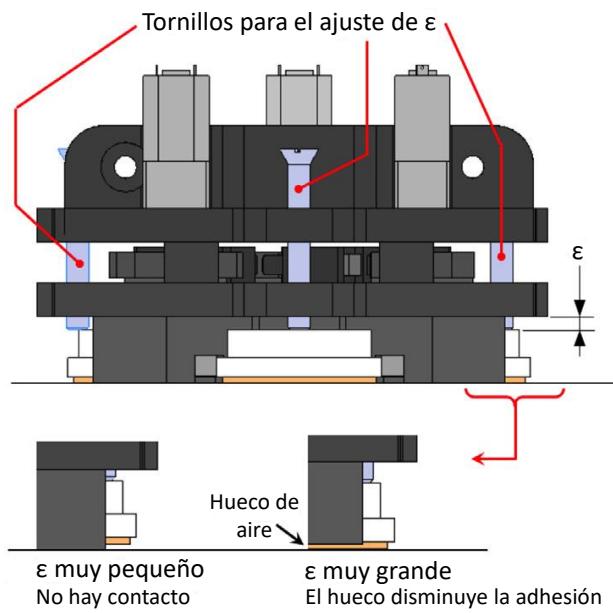


Figura 10: Ajuste de la altura. Figura tomada de (Peidró et al., 2019) con permiso.

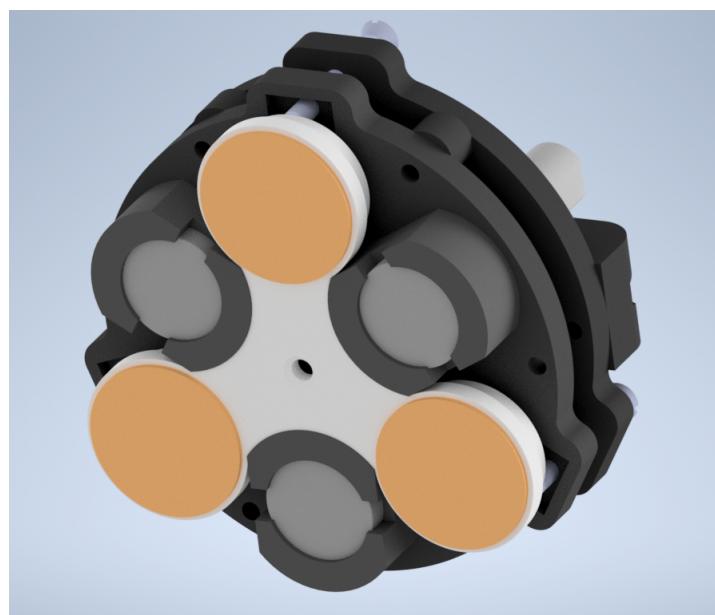


Figura 11: Garra completa original, tal y como se presentó en (Peidró et al., 2019).

3. Control de la aproximación mediante sensores de distancia

El primer objetivo abarcado en el TFG es el de la mejora del sistema de adhesión y fijación de la garra libre a la superficie de pegado. Previamente, el control de la aproximación del efecto final al plano la realizaba un operario manualmente.

El control manual se realiza por medio de incrementos individuales en ambas caderas (θ_A y θ_B) y con control del efecto final, mediante desplazamientos en el plano formado por los ejes **x** e **y** y rotaciones sobre el eje **z**, respecto a la garra fija. Es un proceso complejo de realizar por una persona, ya que para que el pegado sea óptimo, los planos de la garra y de la superficie a la que esta se fija deben ser paralelos y con una distancia mínima entre ellos.

La solución propuesta, con el objetivo de aliviar el trabajo del operario, automatizar la tarea rutinaria y optimizar el proceso, consiste en la adición de 3 sensores de distancia, ubicados de la forma más equidistante posible respecto al centro de la garra y con la mayor separación entre ellos. Los sensores están continuamente tomando medidas, mientras estas son computadas para calcular la matriz de diferencia, la cual expresa la orientación y posición del sistema de referencia deseado en la superficie de pegado respecto al de la garra.

Después, a partir de los datos recogidos por los sensores y mediante un algoritmo que se explicará en profundidad posteriormente, se obtendrán iterativamente los incrementos deseados en cada articulación que consiguen realizar la aproximación deseada de la garra al plano de adhesión.

3.1. Obtención de la matriz de diferencia

Como ya se ha explicado superficialmente, la matriz de diferencia, la cual refleja la pose del punto de pegado respecto al sistema de referencia de la garra, se puede obtener con las medidas tomadas por los 3 sensores en cada instante. Un plano contenido en \mathbb{R}^3 se puede definir naturalmente con un punto perteneciente al plano y un vector ortogonal a él, llamado vector normal, que indica su orientación. Por lo que, a partir de los 3 puntos conocidos a raíz de las medidas tomadas por los sensores, se puede calcular el plano que los contiene.

Consideremos \mathbf{r}_0 , \mathbf{r}_1 y \mathbf{r}_2 , los 3 vectores de posición de los puntos conocidos en el espacio $\mathbf{P}_0 = (x_0, y_0, z_0)$, $\mathbf{P}_1 = (x_1, y_1, z_1)$ y $\mathbf{P}_2 = (x_2, y_2, z_2)$. Un vector ortogonal al plano que contiene los 3 puntos vendrá dado por

$$\mathbf{n} = (\mathbf{r}_0 - \mathbf{r}_1) \times (\mathbf{r}_0 - \mathbf{r}_2). \quad (45)$$

El plano determinado por el punto \mathbf{P}_0 , con vector de posición \mathbf{r}_0 , y el vector $\mathbf{n} =$

(a, b, c) contiene a los puntos \mathbf{P} , con vector de posición \mathbf{r} , para los cuales el vector de \mathbf{P}_0 hasta \mathbf{P} es perpendicular a \mathbf{n} . Cabe recordar la condición de perpendicularidad de 2 vectores: *2 vectores son perpendiculares si, y solo si, su producto escalar es 0.* Por consiguiente, el plano deseado puede ser descrito como el conjunto de todos los puntos \mathbf{r} que cumplen

$$\mathbf{n} \cdot (\mathbf{r} - \mathbf{r}_0) = 0. \quad (46)$$

Expandiendo la ecuación previa, se obtiene la ecuación general del plano

$$a(x - x_0) + b(y - y_0) + c(z - z_0) = 0, \quad (47)$$

que también se puede expresar de forma más compacta como

$$ax + by + cz + d = 0, \quad (48)$$

donde

$$d = -(ax_0 + by_0 + cz_0). \quad (49)$$

También se puede expresar de la forma

$$d = -\mathbf{n} \cdot \mathbf{r}_0. \quad (50)$$

3.1.1. Sensores

Selección de los sensores

Los sensores de distancia, a diferencia de los sensores de proximidad, los cuales indican únicamente la presencia o la ausencia de un objeto, son capaces de dar una medida precisa de la distancia a la que se encuentra el objeto. En el mercado hay un amplio catálogo de diferentes tipos de sensores de distancia. Los más comunes son:

- **Ultrasónicos:** están formados por un emisor, el cual genera una onda de sonido de frecuencia ultrasónica, y un receptor, el cual recibe el eco producido por la onda al rebotar contra el objeto y genera una señal eléctrica en función del tiempo transcurrido.

- **Infrarrojos:** están formados por un fotodiodo y un LED infrarrojo. El LED emite luz infrarroja y el fotodiodo, sensible a luz de la misma longitud de onda que la emitida, la recibe. El voltaje de salida del sensor variará en función de la corriente que pasa por el fotodiodo, la cual vendrá dada por la cantidad de luz infrarroja recibida.
- **LiDAR:** el método que usan para calcular la distancia consiste en emitir un haz láser. La distancia se calcula midiendo el tiempo transcurrido entre la emisión de la señal y la detección del reflejo producido.
- **VCSEL:** son la evolución de los sensores de distancia infrarrojos convencionales. Su funcionamiento es similar a los sensores infrarrojos, pero a diferencia de estos, calculan la distancia en función del tiempo que tarda la señal en ser recibida, no de la intensidad de esta. Su principal ventaja es el considerable menor tamaño respecto a los otros sensores debido a su estructura.

En la siguiente tabla se muestra una comparación de las especificaciones típicas más importantes para los tipos de sensores considerados:

	Rango mínimo	Rango máximo	Resolución	Frecuencia	Coste
Ultrasónicos	Alto	Medio	Variable	Baja	Variable
Infrarrojos	Bajo	Bajo	Media	Media	Bajo
LiDAR	Variable	Muy alto	Media	Muy alta	Alto
VCSEL	Muy bajo	Muy bajo	Muy baja	Media	Medio

Tabla 1: Comparación de las especificaciones típicas para cada tipo de sensor.

Finalmente, se ha decidido optar por los sensores VCSEL. Las características más decisivas para hacer la selección han sido el rango mínimo y la resolución. La aplicación para la que se van a usar requiere la capacidad de tomar medidas de una magnitud muy pequeña y con una buena resolución para hacer los cálculos de forma precisa.

El nombre de este tipo de sensores viene dado por la estructura del emisor láser. El acrónimo VCSEL significa *Vertical-Cavity Surface-Emitting Laser*. Como su nombre indica, son un tipo de LED que, a diferencia de los diodos convencionales, emiten el haz láser perpendicularmente a su superficie a través de una cavidad vertical.

Consisten en 2 espejos, llamados reflectores distribuidos de Bragg, que están colocados de forma paralela a la superficie. Estos semiconductores se dopan para formar una unión de tipo PN, siendo el material de tipo P el que se ubica en la parte superior y el de tipo N en la inferior. Los 2 espejos están separados por una región activa, formada por varios pozos cuánticos, donde se genera la luz. Las capas externas superior e inferior son metales conductores, que sirven como contacto para los semiconductores P y N respectivamente.

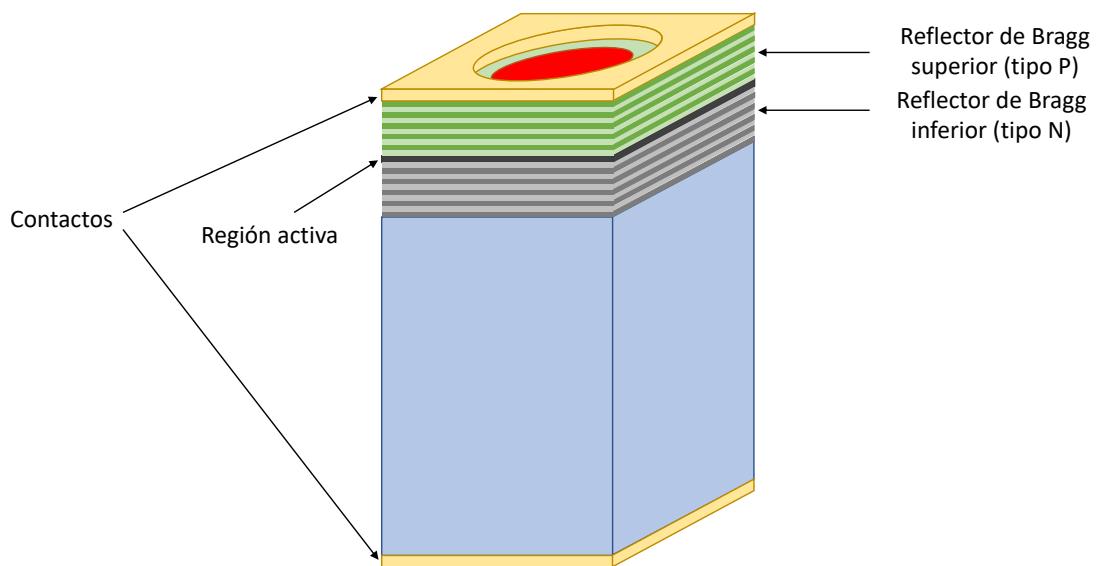


Figura 12: Estructura de un LED láser VCSEL.

VL6180

Concretamente, se ha utilizado el sensor VL6180X de STMicroelectronics, integrado junto con los componentes que precisa en una PCB de SparkFun con el nombre VL6180. Sus especificaciones más importantes que se han considerado son las siguientes:

	Mínimo	Típico	Máximo	Unidades
Tensión de alimentación	2.7	-	-	V
Corriente durante operación	-	300	350	μ A
Frecuencia ¹	-	10	-	Hz
Rango ²	0	-	100	mm
Ruido	-	-	2	mm
Dimensiones	45 x 13 x 2 mm			
Protocolo de comunicación	I ² C			

¹ Aunque 10Hz es el valor por defecto, se puede modificar la frecuencia mediante varios parámetros.

² Se ha comprobado que es capaz de medir hasta unos 180mm (dependiendo de la unidad y de las condiciones del entorno).

Tabla 2: Especificaciones VL6180.

La alimentación del sensor original suponía un problema. Se requería una tensión de 2.8V, por lo que era necesario regular los 3.3V que el microcontrolador de Arduino proporciona. Varias soluciones fueron consideradas:

- **Divisor de tensión convencional:** esta idea fue descartada inmediatamente. La razón fue que la carga formaría parte del divisor, por lo que el valor de la tensión cambiaría con la variación de la carga.

- **PWM y filtro paso bajo RC:** la idea consistía en utilizar un pin PWM de Arduino y cambiar el *duty cycle* para que a la salida del filtro se obtuviesen 2.8V constantes. El problema de esta solución es que hay un tiempo inicial de carga del condensador donde la tensión de salida cambia de 0 a 2.8V, lo cual podría dañar el sensor.
- **Regulador de tensión:** son componentes diseñados para mantener un nivel de tensión de salida constante y regulable. Finalmente, fue la solución utilizada.

La PCB de SparkFun utilizada lleva incorporado un regulador de tensión a costa de un mayor tamaño. Esto permite utilizar los 3.3V que proporciona Arduino como alimentación, ya que serán convertidos a los 2.8V que precisa el sensor.

La comunicación entre el microcontrolador y los sensores se realiza mediante I²C. Este protocolo permite conectar varios dispositivos tan solo usando 2 pines (canales SCL y SDA). El I²C es un sistema que está diseñado como un *bus master-slave*. Esto quiere decir que un dispositivo actúa como “maestro” y los otros como “esclavos”. El maestro se ocupa de generar los pulsos en la línea de reloj (SCL) y comunicarse con los esclavos mediante la línea de datos (SDA). Los esclavos se limitan a responder a las peticiones del maestro por SDA. Su funcionamiento es el siguiente:

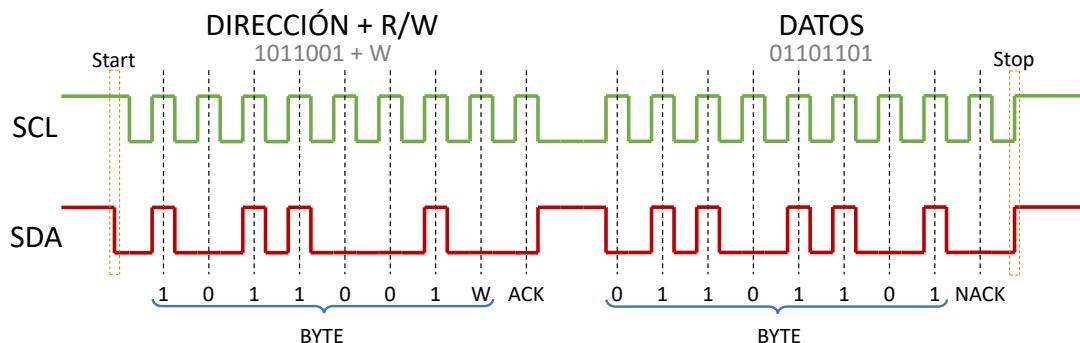


Figura 13: Formato de las líneas SCL y SDA.

El canal SCL es un tren de pulsos generado por el maestro que no varía en ningún momento. Sirve para sincronizar al maestro y a los esclavos para el envío y la recepción de datos.

El canal SDA es el que contiene los datos enviados y recibidos. El orden de los datos a enviar tienen un orden definido:

- El maestro inicia la comunicación enviando un bit de “Start” y generando el tren de pulsos en el canal SCL.
- Inmediatamente, el maestro envía un byte conteniendo los 7 bits de la dirección del esclavo con el que se quiere comunicar y 1 bit indicando si quiere leer o escribir (R/W).

- Los esclavos leen la dirección recibida y si coincide con la suya envían un bit de recepción (ACK).
- A partir de este punto, se envían o reciben los bytes necesarios por parte del maestro y los esclavos, siempre terminados con un bit de recepción.
- El número de bytes enviados depende de la acción que vayan a realizar el maestro y el esclavo. Se suele utilizar un byte para determinar la orden del maestro (tomar una medida, configurar algún parámetro, enviar una acción de control...) y un byte con la respuesta del esclavo.
- En el caso de que el último bit de un byte sea negativo (NACK), la comunicación entre el maestro y el esclavo termina tras un bit de “Stop”.

Ubicación de los sensores

Encontrar la ubicación adecuada de los sensores no fue una tarea sencilla. El diseño original de la garra llevaba que muchos elementos estuviesen contenidos en un espacio muy reducido, por lo que el espacio utilizable donde se podían ubicar los sensores era muy reducido:

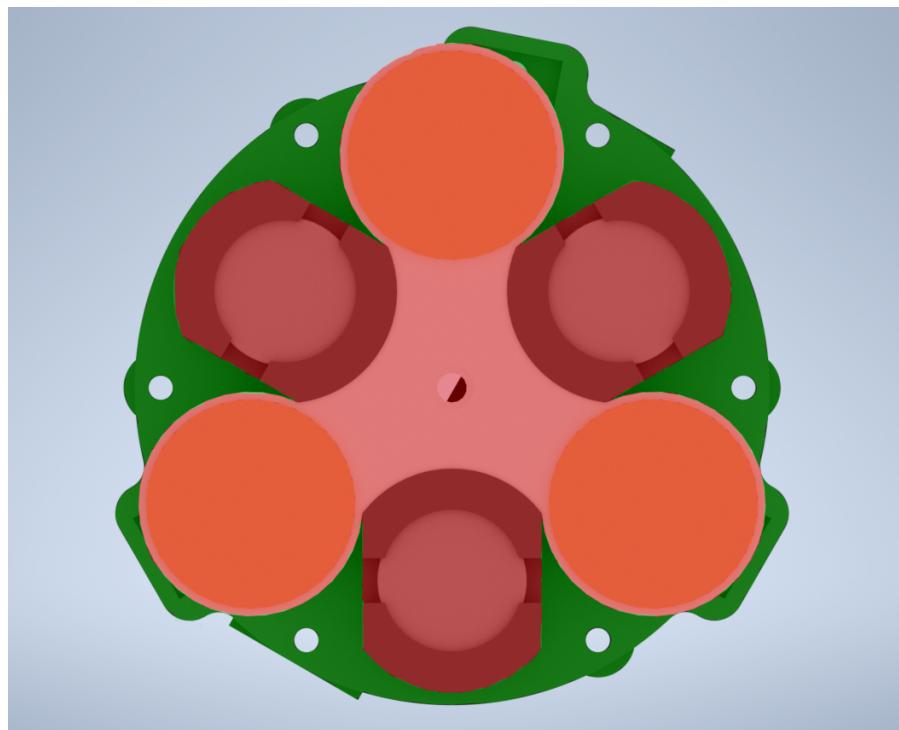


Figura 14: Espacio utilizable (verde) y ocupado (rojo).

De forma ideal, los 3 sensores estarían ubicados de forma que hubiese un desfase de 120° entre cada uno de ellos respecto al centro de la garra. También, sería conveniente que estuviesen lo más alejados posible del centro, para abarcar el mayor espacio posible y hacer un cálculo más preciso del plano medido.

Sin embargo, la disposición de los elementos previamente presentes limitan los posibles emplazamientos de los sensores sin modificar la estructura original. Además, los nuevos elementos no deben sobresalir en exceso el diámetro de la garra original, ya que podría provocar colisiones con la otra garra y, por ende, reducir el espacio de trabajo del robot.

La solución propuesta consiste en aprovechar los ejes por donde pasan los tornillos que ajustan el accesorio de fricción, fijando un extremo del sensor, y modificar la pieza inferior de la garra. La modificación hecha sobre la pieza original se basa en una extrusión para cada sensor, donde se apoyará y fijará el otro extremo del sensor. Además, se han realizado unos orificios en la pieza donde están los sensores en sí, ya que de lo contrario no podrían hacer las medidas. En las siguientes figuras se puede ver el diseño final de la garra:

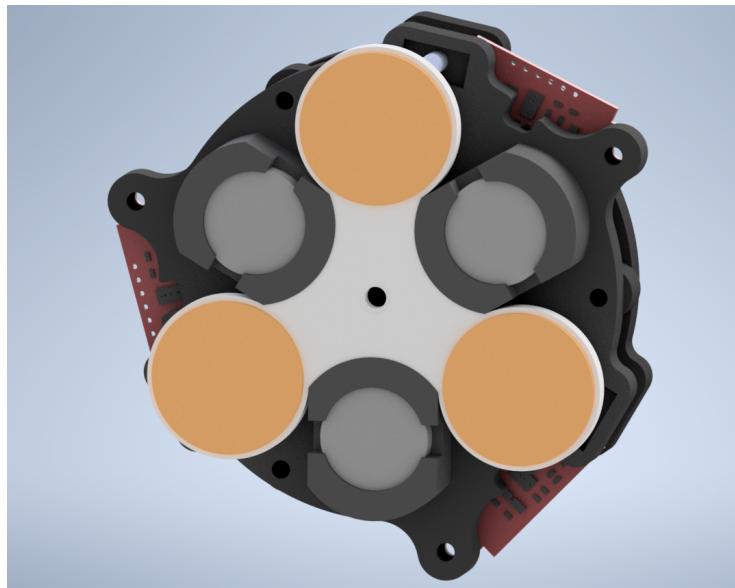


Figura 15: Renderización del diseño final de la garra.

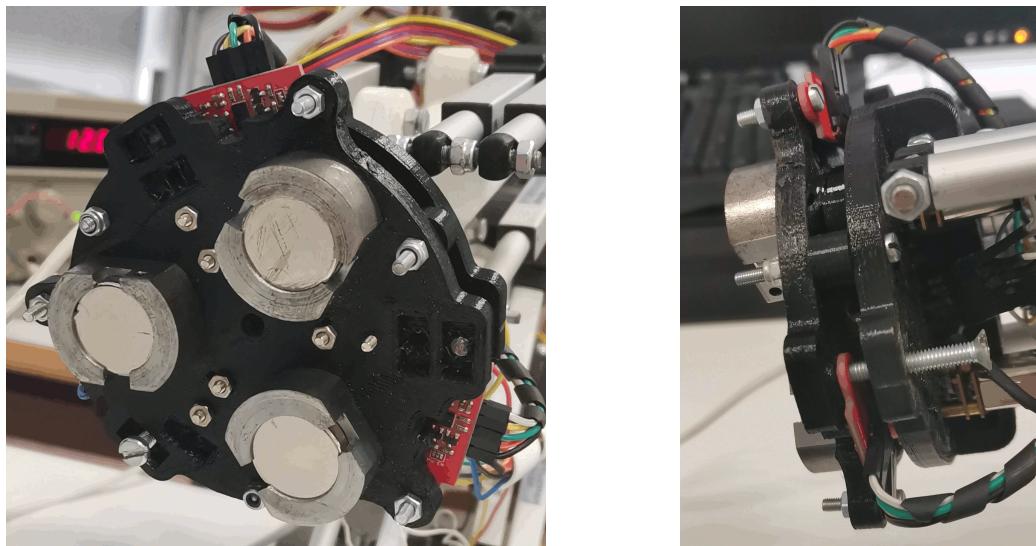


Figura 16: Montaje de los sensores en la garra.

3.1.2. Filtrado y calibración

En cualquier medida tomada con sensores es inevitable que haya algún tipo de ruido en los datos de salida. Las fuentes del ruido son muchas y prácticamente inevitables, por lo que siempre es necesario un filtrado de los datos obtenidos. Varios tipos de filtros fueron considerados: media, mediana, moda, Hampel, Kalman... Finalmente, se optó por un filtro paso bajo RC.

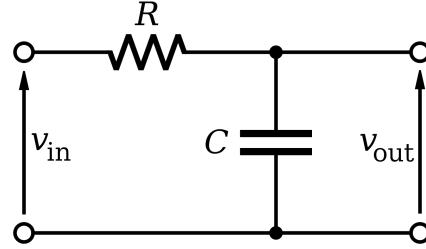


Figura 17: Filtro paso bajo RC.

Partiendo del circuito de un filtro paso bajo RC analógico (figura 17) y aplicando las leyes de Kirchhoff, se obtiene:

$$v_{in}(t) = R i(t) + v_{out}(t) \quad (51)$$

$$i(t) = C \frac{dv_{out}}{dt} \quad (52)$$

Sustituyendo 52 en 51:

$$v_{in}(t) = R C \frac{dv_{out}}{dt} + v_{out}(t) \quad (53)$$

La ecuación se puede discretizar, tomando como periodo dt y representando los muestreos de v_{in} como (x_1, x_2, \dots, x_n) y v_{out} como (y_1, y_2, \dots, y_n) :

$$x_i = R C \frac{y_i - y_{i-1}}{dt} + y_i \quad (54)$$

Reordenándola:

$$y_i = x_i \frac{dt}{RC + dt} + y_{i-1} \frac{RC}{RC + dt} \quad (55)$$

Definiendo

$$\alpha := \frac{dt}{RC + dt} \quad (56)$$

se obtiene:

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1} \quad (57)$$

La frecuencia de corte está relacionada con la constante de tiempo de la forma:

$$RC = \frac{1}{2\pi f_c} \quad (58)$$

Por lo que:

$$\alpha = \frac{dt}{\frac{1}{2\pi f_c} + dt} \quad (59)$$

El filtro paso bajo se puede programar fácilmente en Arduino:

```

1 double dt = 0.033; // valor inicial estimado
2 double salida_anterior = 0;
3 const double freq_corte = 0.5; // variable modificable
4
5 double alpha;
6 unsigned long t = 0;
7
8 void filtro_paso_bajo(double entrada, double &salida){
9     t = millis();
10    alpha = dt / (1 / (2 * PI * freq_corte) + dt);
11
12    salida = alpha * entrada + (1 - alpha) * salida_anterior;
13    salida_anterior = salida;
14
15    dt = (millis() - t)/1000.0;
16}
```

Para la calibración de los sensores, un ajuste lineal es necesario para obtener los valores buscados de las medidas. Las curvas de calibración calculadas para cada sensor son las siguientes:

- Sensor 1: $y = 1.08x - 17.46$
- Sensor 2: $y = 1.04x - 22.46$
- Sensor 3: $y = 1.03x - 17.95$

donde x es la distancia original medida por el sensor e y es la distancia real desde la superficie de la garra, ambas en milímetros.

3.1.3. Representación en tiempo real de la posición y orientación

A modo de herramienta complementaria, se ha diseñado y programado un script en MATLAB que representa en tiempo real el plano de la superficie a la que se acerca la garra. El objetivo del programa es que sea utilizado como apoyo para el operario, en el caso de que esté controlando la aproximación de la garra de forma manual, y como herramienta de supervisión durante proceso de aproximación automático.

Para ejecutar el script es necesaria una versión de MATLAB R2019b o más reciente, ya que las funciones de comunicación por puerto serial fueron actualizadas en esa versión. El programa, que posteriormente se explicará, es el siguiente:

```
1 clear
2 close all
3 clc
4
5 % altura del pie del robot
6 f = 39;
7
8 % Coordenadas de los sensores
9 xs = [23.3 -41.51 17.94];
10 ys = [34.32 -4.55 -38.23];
11 zs = ones(1,3)*-f;
12
13 % Serial
14 s = serialport("COM7", 115200);
15 configureTerminator(s,"CR/LF");
16 flush(s);
17 configureCallback(s,"terminator",@readserial);
18 readline(s);
19
20 % GRAFICA
21 fig = figure;
22 axis equal
23 min = -60; % limites x e y
24 max = 60;
25 axis([min max min max -inf inf])
26 hold on
27 xlabel('x')
28 ylabel('y')
29 zlabel('z')
30 view(45,15)
31
32 % objetos estaticos
33 garra = stlread('garra.stl');
34 trimesh(garra,'FaceColor','k','EdgeColor','k','FaceAlpha',0.6)
35 plot3(xs, ys, zs, 'ob')
36 quiver3(zeros(1,3), zeros(1,3), zeros(1,3), [1, 0, 0], [0, 1, 0],
37 [0, 0, 1], 20, 'b')
38 % REPRESENTACION PRIMERA MEDIDA
39 % primera medida
40 zd = get_d();
41 zm = zs-zd;
42 h1 = plot3(xs, ys, zm, 'or');
```

```

43 h1.ZDataSource = 'zm';
44
45 % lineas sensor-medida
46 z1 = [zs(1) zm(1)];
47 z2 = [zs(2) zm(2)];
48 z3 = [zs(3) zm(3)];
49 h2 = plot3([xs(1) xs(1)], [ys(1) ys(1)], z1, '--r');
50 h3 = plot3([xs(2) xs(2)], [ys(2) ys(2)], z2, '--r');
51 h4 = plot3([xs(3) xs(3)], [ys(3) ys(3)], z3, '--r');
52 h2.ZDataSource = 'z1';
53 h3.ZDataSource = 'z2';
54 h4.ZDataSource = 'z3';
55
56 % plano
57 paso = 1;
58 [X, Y] = meshgrid(min:paso:max);
59 p1 = [xs(1) ys(1) zm(1)]; % puntos medidos
60 p2 = [xs(2) ys(2) zm(2)];
61 p3 = [xs(3) ys(3) zm(3)];
62 normal = cross(p1 - p2, p1 - p3);
63 normal = normal/norm(normal);
64 d = p1(1)*normal(1) + p1(2)*normal(2) + p1(3)*normal(3);
65 Z = (d - (normal(1)*X) - (normal(2)*Y))/normal(3);
66 h5 = mesh(X,Y,Z);
67 h5.ZDataSource = 'Z';
68
69 % sistema de referencia superficie
70 pos_sup = [0, 0, d/normal(3)]';
71 z_aux = (d - (normal(1)*1) - (normal(2)*0))/normal(3);
72 w_sup = normal;
73 u_sup = [1 - 0, 0 - 0, z_aux - d/w_sup(3)];
74 u_sup = u_sup/norm(u_sup);
75 v_sup = cross(w_sup, u_sup);
76
77 h7 = quiver3(pos_sup(1), pos_sup(2), pos_sup(3), w_sup(1), w_sup(2),
    , w_sup(3), 20, 'r');
78 h7 = [h7, quiver3(pos_sup(1), pos_sup(2), pos_sup(3), u_sup(1),
    u_sup(2), u_sup(3), 20, 'r')];
79 h7 = [h7, quiver3(pos_sup(1), pos_sup(2), pos_sup(3), v_sup(1),
    v_sup(2), v_sup(3), 20, 'r')];
80
81 % sistema de referencia deseado
82 pos_des = pos_sup + f*w_sup';
83 h7 = [h7, quiver3(pos_des(1), pos_des(2), pos_des(3), w_sup(1),
    w_sup(2), w_sup(3), 20, 'b')];
84 h7 = [h7, quiver3(pos_des(1), pos_des(2), pos_des(3), u_sup(1),
    u_sup(2), u_sup(3), 20, 'b')];
85 h7 = [h7, quiver3(pos_des(1), pos_des(2), pos_des(3), v_sup(1),
    v_sup(2), v_sup(3), 20, 'b')];
86
87 % BUCLE PARA ACTUALIZAR DATOS
88 while 1
89     % nueva medida
90     zd = get_d();
91     zm = zs-zd;
92
93     % actualizar lineas sensor-medida

```

```

94 z1 = [zs(1) zm(1)];
95 z2 = [zs(2) zm(2)];
96 z3 = [zs(3) zm(3)];
97
98 % actualizar plano
99 p1 = [xs(1) ys(1) zm(1)];
100 p2 = [xs(2) ys(2) zm(2)];
101 p3 = [xs(3) ys(3) zm(3)];
102 normal = cross(p1 - p2, p1 - p3);
103 normal = normal/norm(normal);
104 d = p1(1)*normal(1) + p1(2)*normal(2) + p1(3)*normal(3);
105 Z = (d - (normal(1)*X) - (normal(2)*Y))/normal(3);
106
107 % actualizar sistemas de referencia
108 pos_sup = [0, 0, d/normal(3)]';
109 z_aux = (d - (normal(1)*1) - (normal(2)*0))/normal(3);
110 w_sup = normal;
111 u_sup = [1 - 0, 0 - 0, z_aux - d/w_sup(3)];
112 u_sup = u_sup/norm(u_sup);
113 v_sup = cross(w_sup, u_sup);
114
115 pos_des = pos_sup + f*w_sup';
116 or_des = [u_sup', v_sup', w_sup'];
117 T_des = [or_des, pos_des; [0 0 0 1]]
118
119 delete(h7);
120 h7 = quiver3(pos_sup(1), pos_sup(2), pos_sup(3), w_sup(1),
w_sup(2), w_sup(3), 20, 'r');
121 h7 = [h7, quiver3(pos_sup(1), pos_sup(2), pos_sup(3), u_sup(1),
u_sup(2), u_sup(3), 20, 'r')];
122 h7 = [h7, quiver3(pos_sup(1), pos_sup(2), pos_sup(3), v_sup(1),
v_sup(2), v_sup(3), 20, 'r')];
123
124 h7 = [h7, quiver3(pos_des(1), pos_des(2), pos_des(3), w_sup(1),
w_sup(2), w_sup(3), 20, 'b')];
125 h7 = [h7, quiver3(pos_des(1), pos_des(2), pos_des(3), u_sup(1),
u_sup(2), u_sup(3), 20, 'b')];
126 h7 = [h7, quiver3(pos_des(1), pos_des(2), pos_des(3), v_sup(1),
v_sup(2), v_sup(3), 20, 'b')];
127
128 if ~ishghandle(fig) % si se cierra la ventana
129     break
130 else
131     % dibuja los nuevos datos
132     refreshdata(h1)
133     refreshdata(h2)
134     refreshdata(h3)
135     refreshdata(h4)
136     refreshdata(h5)
137     drawnow
138 end
139 end
140
141 close all
142 clear s

```

Las primeras líneas del código corresponden a la declaración de variables y a la configuración de algunos parámetros. Cabe aclarar el uso de la función `configureCallback`, que se encarga de activar la llamada a una función cada vez que se recibe por serial un terminador. Previamente, se ha configurado con `configureTerminator` el carácter "CR/LF" como terminador. Es decir, cada vez que el programa lea un retorno de carro (CR) con salto de línea (LF) por el serial, llamará a la función `readserial`. La sencilla función `readserial` lee 3 valores enviados por el serial separados por comas y se los asigna a una variable global como vector.

```

1 function readserial(src, ~)
2 global d
3 d = (str2num(readline(src)));
4 end

```

A continuación, en las líneas 33-36 del código original, se dibujan los objetos estáticos que no variarán en ningún instante. La garra se dibuja a partir de un archivo .STL³ y los sensores se representan como circunferencias azules. También se representa en azul el sistema de referencia del origen de la garra.

Seguidamente (línea 40), se toma el primer grupo de 3 medidas mediante la función `get_d()`. La función simplemente recoge la variable global correspondiente a las 3 medidas que crea la función `readserial` y se la asigna a otra variable.

```

1 function [s] = get_d()
2 global d
3 s = d;
4 end

```

Luego, a partir de la línea 46, se proyectan 3 puntos desde el plano original que forman los sensores hasta el plano formado por los puntos del plano leídos por los sensores. Se dibujan en rojo tanto los puntos proyectados como las líneas de las proyecciones. De esta forma, se han obtenido los 3 puntos conocidos en el espacio que conforman el plano de la superficie de pegado, y ya se puede aplicar el procedimiento expuesto en la sección 3.1 (ecuaciones (45)-(50)).

Para cada elemento dibujado, se define la variable que variará a lo largo del tiempo. Actualizar las representaciones de esta forma será más sencillo y menos costoso computacionalmente que volver a dibujar todos los elementos gráficos al completo cuando se produzca un cambio. Este método para ahorrar potencia computacional se utilizará para todos los elementos gráficos en adelante.

Para calcular el plano que contiene a los 3 puntos, se calcula su vector normal mediante el producto vectorial de 2 vectores que contengan a los 3 puntos (ecuación (45)). A partir del vector normal del plano, se puede utilizar la ecuación (50) para calcular d . Mediante la ecuación (48), se puede dibujar el plano en su totalidad.

A partir del plano, es posible construir la matriz de diferencia. Se explicará en detalle su cálculo en la sección 3.2.2.

³Formato de archivo que define geometrías tridimensionales mediante combinación de triángulos.

Para acabar, a lo largo de las líneas 70-85, se dibujan el sistema de referencia de la superficie y el deseado, en rojo y azul respectivamente. El sistema de referencia de la superficie se puede obtener de forma sencilla. El vector que define su eje **z** se corresponde al vector normal previamente calculado. El vector del eje **x** se calculará proyectando el punto (1, 0, 0) desde el plano de la garra hasta el calculado. El vector que define el eje **y** será el producto vectorial de los 2 anteriores. Las coordenadas desde donde se dibujarán se pueden calcular proyectando el punto (0, 0, 0) desde el plano de la garra hasta el calculado.

El sistema de referencia deseado de la garra se obtendrá haciendo una traslación del sistema de referencia de la superficie. Se calcula una traslación del punto \mathbf{P}_0 , una distancia d , a través de un vector \mathbf{v} , mediante:

$$\mathbf{P} = \mathbf{P}_0 + d \mathbf{v} \quad (60)$$

La traslación hecha en el programa para obtener el sistema de referencia deseado se corresponde a una traslación del sistema de referencia de la superficie, una distancia igual a la altura del pie de la garra (distancia entre el origen del sistema de referencia de la garra y el plano formado por los imanes y el accesorio de fricción, variable f en el código, igual a 39mm), a través del vector normal al plano de la superficie.

Finalmente, a partir de la línea 87, el programa entra en un bucle donde se actualizarán los datos y elementos dibujados con cada toma de medidas por parte de los sensores.

En las figuras 18-20 se pueden observar las representaciones que hace el programa en varias situaciones diferentes. Cabe comentar que estas figuras no son simulaciones, sino representaciones en tiempo real a partir de los datos tomados por los sensores montados sobre la garra, acercando y alejando la garra manualmente a una superficie.

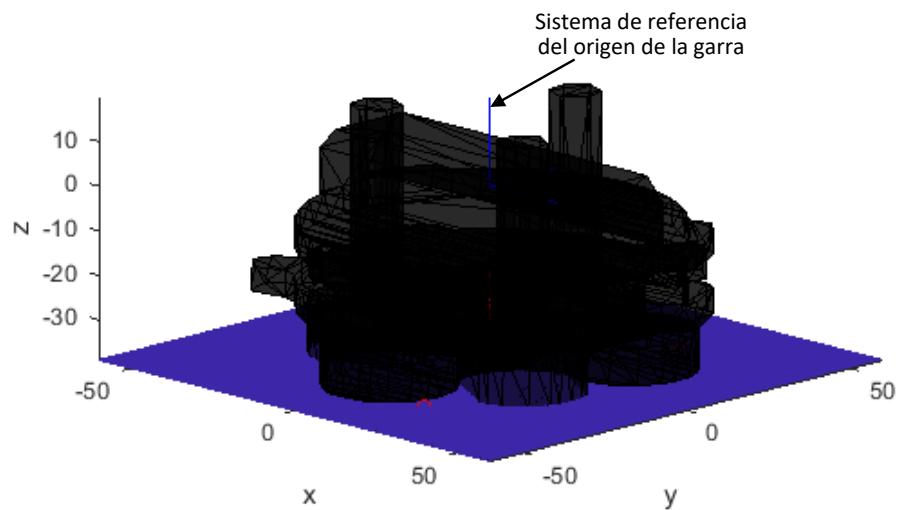


Figura 18: Garra pegada a la superficie.

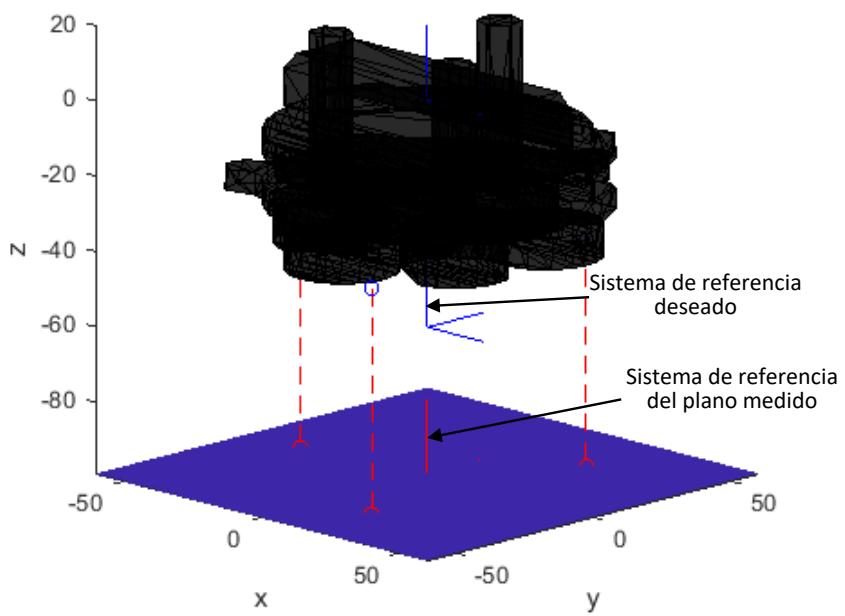


Figura 19: Garra paralela a la superficie.

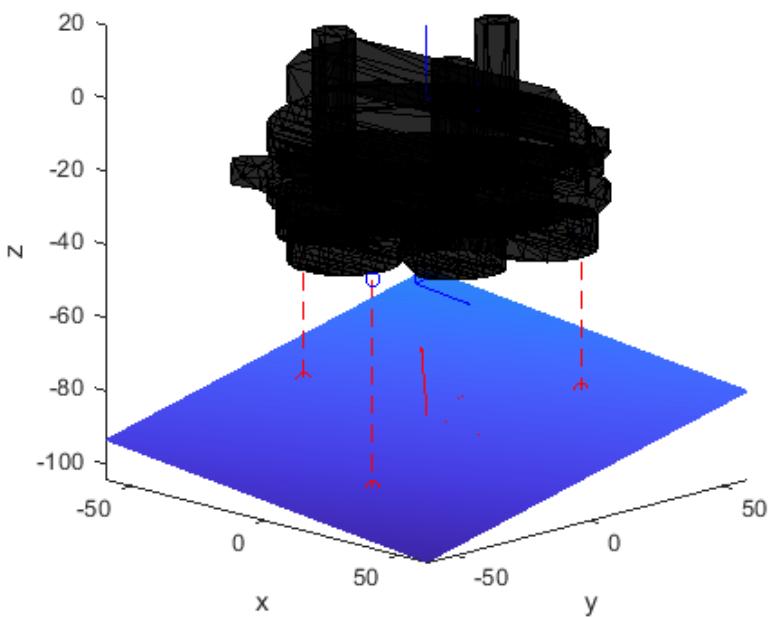


Figura 20: Garra con una pose aleatoria.

3.2. Algoritmo de aproximación mediante Jacobiana

En la sección anterior, se ha estudiado como obtener la matriz de diferencia, la cual da la información de la posición y la orientación de la superficie respecto a la garra, a partir de las medidas de 3 sensores. A continuación, se propondrá un algoritmo que permitirá el acercamiento de la garra de una forma controlada y automática. El diagrama de flujo del algoritmo es el siguiente:

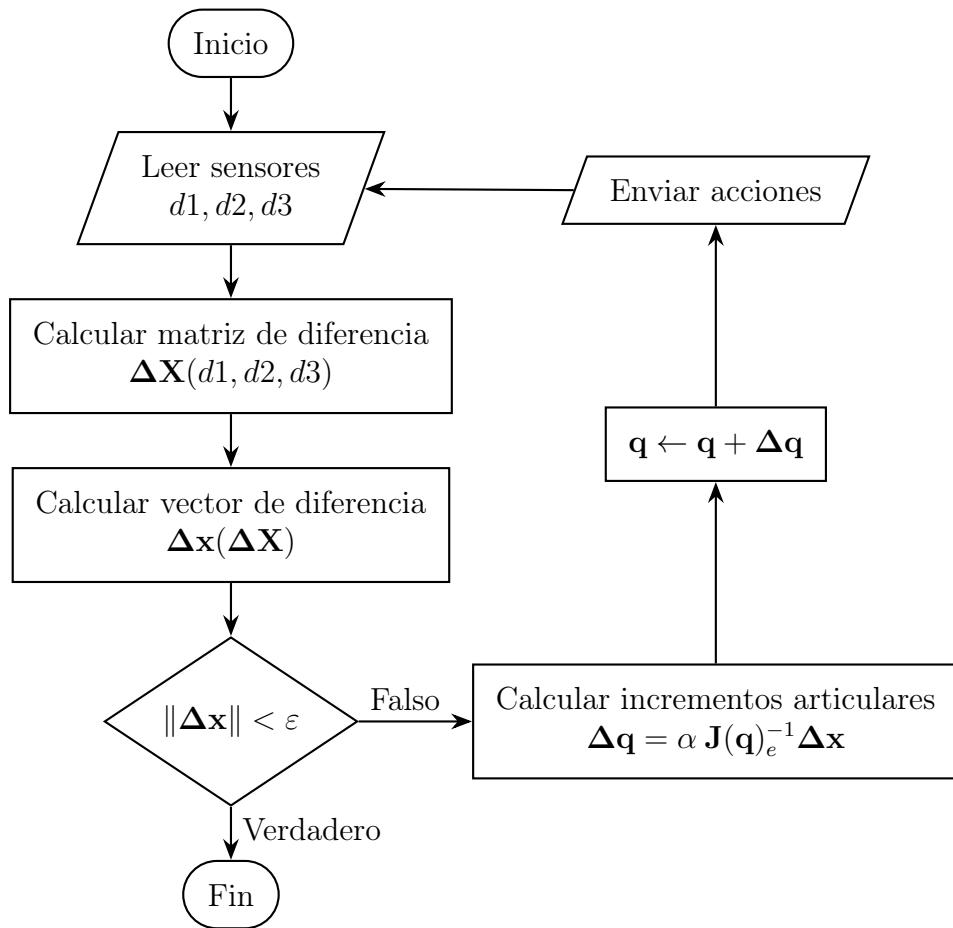


Figura 21: Algoritmo de aproximación.

El algoritmo permite que, desde cualquier posición, la garra del robot siga una trayectoria controlada y realimentada hasta posarse en la superficie donde se pegará.

El código del programa está escrito en el lenguaje de Arduino. Cada operación del diagrama está implementada en el código con una función y la estructura del apartado se dividirá de la misma forma. Primero, se explicará de forma generalizada la función y el proceso que realiza. Luego, se mostrará el código de la función. Y para finalizar, se explicará en detalle el código.

3.2.1. Lectura de los sensores

El primer paso consiste en hacer una lectura de los 3 sensores. Como ya se ha explicado previamente, las lecturas se someten a un filtrado y una calibración para obtener un resultado más conveniente.

```

1 void get_vl6180(double &d1_calibrado, double &d2_calibrado, double
2   &d3_calibrado){
3   const double freq_corte = 0.5;
4   double alpha;
5   int d1, d2, d3;
6
7   const int n_s = 1;
8   static unsigned long t_s = 0;
9   static double d_t;
10
11  d_t = (millis() - t_s)/1000.0;
12  alpha = d_t / (1 / (2* PI * freq_corte) + d_t);
13  t_s = millis();
14
15  d1 = sensor1.getDistance();
16  d1_filtrado[n_s] = alpha * d1 + (1 - alpha) * d1_filtrado[n_s
17 -1];
18  d1_filtrado[n_s-1] = d1_filtrado[n_s];
19  d1_calibrado = 1.08 * d1_filtrado[n_s] - n_1;
20
21  d2 = sensor2.getDistance();
22  d2_filtrado[n_s] = alpha * d2 + (1 - alpha) * d2_filtrado[n_s
23 -1];
24  d2_filtrado[n_s-1] = d2_filtrado[n_s];
25  d2_calibrado = 1.04 * d2_filtrado[n_s] - n_2;
26
27  d3 = sensor3.getDistance();
28  d3_filtrado[n_s] = alpha * d3 + (1 - alpha) * d3_filtrado[n_s
29 -1];
30  d3_filtrado[n_s-1] = d3_filtrado[n_s];
31  d3_calibrado = 1.03 * d3_filtrado[n_s] - n_3;
32 }
```

La función toma los argumentos por referencia. De esta forma, se podrán modificar la variables y que sirvan como salida de la función. A continuación se declaran las variables de la función. Cabe destacar que las variable `t_s` y `d_t` son estáticas, por lo que solo se inicializarán la primera vez que se llame a la función, comportándose a efectos prácticos como variables globales.

Las líneas de código 10-12 corresponden al cálculo de la variable `alpha`, de la forma que se ha explicado en la sección 3.1.2. Mediante la función `millis()`, la cual devuelve el tiempo de ejecución del programa en milisegundos, se calcula la diferencia de tiempo entre cada llamada a la función y se pasa a segundos. Luego, siguiendo la ecuación (59), se calcula el valor de `alpha` para cada iteración.

Seguidamente, a partir de la línea 14, se produce la lectura de los sensores, se hace el filtrado de la señal y se procesa el calibrado. La función `getDistance()` forma parte de la librería que proporciona el fabricante de los sensores. Se ocupa de comunicarse mediante I²C con los sensores para obtener una lectura de la distancia medida en milímetros.

Para concluir, se repite el conjunto de instrucciones de lectura, filtrado y calibrado 2 veces más, una para cada sensor restante.

3.2.2. Cálculo de la matriz de diferencia

A continuación, con la función `calc_T` se calculará la matriz **T** que “ven” los sensores. La matriz calculada contiene implícitamente la pose de la superficie vista desde los sensores, que se usará como matriz de diferencia (matriz ΔX en la figura 21)

El método utilizado es similar al que se emplea para la representación en MATLAB. Está demostrado en la sección 3.1, se utilizan las ecuaciones (45)-(50) para obtener la matriz de diferencia.

```

1 void calc_T(double T[4*4]){
2     double f = 30;
3     double p1[3] = {23.3, 0, 34.32};
4     double p2[3] = {-41.51, 0, -4.55};
5     double p3[3] = {17.94, 0, -38.23};
6     double pos[3] = {0, 0, 0};
7     double a[3], b[3], normal[3], distancia;
8     double u[3] = {1, 0, 0};
9     double v[3];
10
11    get_vl6180(p1[1], p2[1], p3[1]);
12
13    for (int i=0; i<3; i++){
14        a[i] = p1[i] - p3[i];
15        b[i] = p1[i] - p2[i];
16    }
17    cross_product(a, b, normal);
18    normalize(normal);
19
20    distancia = (p1[0]*normal[0] + p1[1]*normal[1] + p1[2]*normal
21 [2])/normal[1];
22    pos[1] = - distancia + f;
23
24    u[1] = - normal[0]/normal[1];
25    normalize(u);
26    cross_product(u, normal, v);
27
28    for (int n=0; n<3; n++){
29        T[n*4 + 0] = u[n];
30        T[n*4 + 1] = normal[n];
31        T[n*4 + 2] = v[n];
32        T[n*4 + 3] = pos[n]/1000.;
33    }
34}
```

```

33     T[3*4 + 0] = 0;
34     T[3*4 + 1] = 0;
35     T[3*4 + 2] = 0;
36     T[3*4 + 3] = 1;
37 }
```

Primero, se declaran e inicializan las variables. Las componentes x e y de los puntos medidos son equivalentes a las de los sensores, por lo que son conocidos. La componente z se obtienen realizando una lectura de los sensores.

A partir de los 3 puntos conocidos contenidos en el plano, se calculan 2 vectores y se computa su producto vectorial para obtener un vector normal al plano. La función para calcular el producto vectorial de 2 vectores tridimensionales es la siguiente:

```

1 void cross_product(double A[3], double B[3], double C[3]){
2     C[0] = A[1]*B[2] - A[2]*B[1];
3     C[1] = -(A[0]*B[2] - A[2]*B[0]);
4     C[2] = A[0]*B[1] - A[1]*B[0];
5 }
```

Se normaliza el vector normal al plano para obtener un vector unitario con la siguiente función:

```

1 void normalize(double v[3]){
2     double magnitud;
3     magnitud = sqrt(pow(v[0],2) + pow(v[1],2) + pow(v[2],2));
4     for (int i=0; i<3; i++){
5         v[i] = v[i]/magnitud;
6     }
7 }
```

A continuación, se proyecta ortogonalmente el punto $(0, 0, 0)$ sobre el plano para obtener la variable **distancia**. De esta forma, es conocida la posición del sistema de referencia del plano, equivalente a $(0, distancia, 0)$.

En el código, se puede justificar el cálculo de la variable **distancia** en la línea 20 a partir de la ecuación (50) y conociendo que los componentes x y z del vector de posición **p** son nulos al coincidir con el eje **y** del origen, se puede obtener la siguiente igualdad:

$$-(a, b, c) \cdot (0, y_0, 0) = d \quad (61)$$

Por lo que:

$$by_0 = d \quad (62)$$

Reordenando la ecuación:

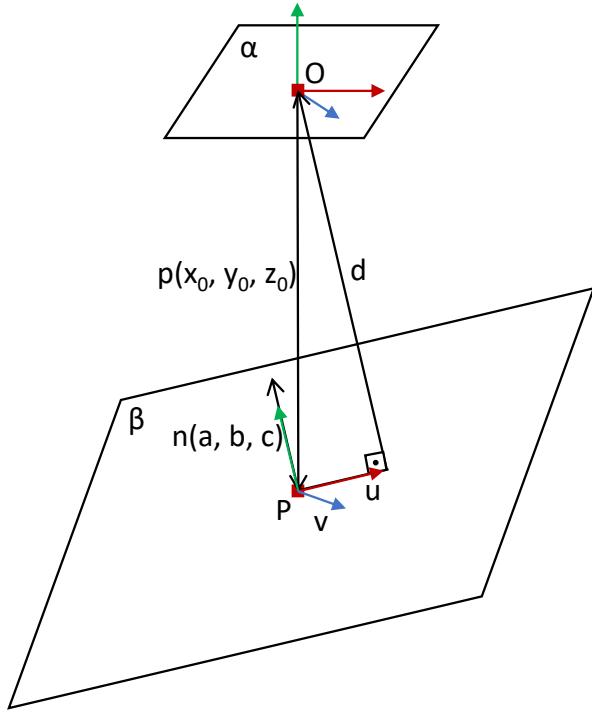


Figura 22: Obtención del sistema de referencia de la superficie medida.

$$y_0 = \frac{d}{b} \quad (63)$$

Este vector de posición \mathbf{p} corresponde a los 3 primeros elementos de la cuarta columna de la matriz de diferencia.

Los elementos de la matriz que definen la rotación se pueden obtener a partir de 3 vectores, cada uno correspondiente a la rotación producida en los ejes \mathbf{x} , \mathbf{y} y \mathbf{z} . El vector que define el eje \mathbf{y} ya es conocido al ser equivalente al vector normal al plano ya calculado.

El vector que define el eje \mathbf{x} , llamado $\mathbf{u} = (u_x, u_y, u_z)$, se obtiene proyectando un punto del eje \mathbf{x} del origen $(1, 0, 0)$ al plano. Al ser una proyección ortogonal respecto al plano del origen, el componente u_x será 1. Se conoce que los ejes de un sistema de referencia siempre guardan una relación de perpendicularidad, por lo que se puede obtener el componente y del vector \mathbf{u} a partir de su vector perpendicular \mathbf{n} , como se puede comprobar en la figura 22:

$$u_y = -\frac{a}{b} \quad (64)$$

Tras normalizar el vector \mathbf{u} , ya se conocen 2 de los 3 ejes que definen la orientación.

Conociendo 2 de los ejes del sistema de referencia de la superficie, se puede

obtener el tercero mediante el producto vectorial de ambos:

$$\mathbf{v} = \mathbf{u} \times \mathbf{n} \quad (65)$$

Para acabar, mediante la función mostrada a continuación, se obtiene la posición del sistema de referencia deseado haciendo una traslación del punto, una distancia equivalente a la altura del pie, siguiendo el vector normal al plano. Al tratarse de una traslación sin rotación, la orientación se mantiene constante.

```

1 void translate(double p[3], double v[3], double d){
2     for (int i=0; i<3; i++){
3         p[i] = p[i] + d*v[i];
4     }
5 }
```

Finalmente, se construye la matriz a partir de la posición y orientación calculadas del sistema de referencia deseado convirtiendo el vector de posición de milímetros a metros.

3.2.3. Obtención del vector de diferencia

El siguiente proceso es obtener un vector que contenga la pose implícita en la matriz de diferencia. El resultado es una traslación y una rotación descritas por un vector de 6 componentes, el cual permitirá calcular los incrementos necesarios para mover el efecto final hasta la pose deseada.

```

1 void tr2delta(double T[4*4], double delta[6]){
2     for (int i=0; i<3; i++){
3         delta[i] = T[i*4 + 3];
4     }
5     delta[3] = 0.5 * (T[2*4 + 1] - T[1*4 + 2]);
6     delta[4] = 0.5 * (T[0*4 + 2] - T[2*4 + 0]);
7     delta[5] = 0.5 * (T[1*4 + 0] - T[0*4 + 1]);
8 }
```

Los 3 primeros elementos del vector Δ determinan la posición o traslación y los 3 restantes la orientación:

$$\Delta = \begin{bmatrix} \Delta_P \\ \Delta_R \end{bmatrix} = \begin{bmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \\ \Delta\omega_x \\ \Delta\omega_y \\ \Delta\omega_z \end{bmatrix} \quad (66)$$

Partiendo de una matriz de transformación \mathbf{T} , se pueden diferenciar los elementos de la siguiente forma:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R}_{3 \times 3} & \Delta_{P \ 3 \times 1} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix} \quad (67)$$

De donde Δ_P se puede extraer directamente de la cuarta columna y Δ_R se puede extraer de la matriz de rotación \mathbf{R} . Si a esta matriz \mathbf{R} se le resta la identidad, se obtiene una matriz antisimétrica de donde se pueden extraer los componentes restantes correspondientes a la orientación (Corke, 2017, p. 67):

$$\mathbf{R} - \mathbf{I} = \begin{bmatrix} 0 & -\Delta\omega_z & \Delta\omega_y \\ \Delta\omega_z & 0 & -\Delta\omega_x \\ -\Delta\omega_y & \Delta\omega_x & 0 \end{bmatrix} \quad (68)$$

3.2.4. Comprobación de la norma del vector

Esta función comprueba si la norma del vector \mathbf{v} introducido es menor que un valor ε .

```

1  bool check_norm(double v[], int n){
2      const double epsilon = 0.01;
3      double a = 0;
4
5      for (int i=0; i<n; i++) {
6          a += pow(v[i], 2);
7      }
8      if (sqrt(a) < epsilon ) {
9          return true;
10     } else {
11         return false;
12     }
13 }
```

Se calcula la norma vectorial de un vector de n dimensiones mediante

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^n v_i^2} = \sqrt{v_1^2 + v_2^2 + \dots + v_{n-1}^2 + v_n^2} \quad (69)$$

Si la norma es menor a un valor ε establecido, la función devuelve el valor booleano o lógico `true`. En caso de que sea mayor o igual devuelve `false`.

3.2.5. Actualización de la configuración articular

La función actualiza el valor de las coordenadas articulares. Se calcula el incremento en las coordenadas articulares intermedias haciendo uso de la Jacobiana

en coordenadas locales del efecto final y se devuelve la configuración articular actualizada.

```

1 void act_q(double q_int[8], double T[4*4], double delta_x[3],
2   boolean fijo_A){
3   double alpha = 0.1;
4   // jacobiana coordenadas globales y locales
5   double J[6*8], Je[6*8], Je_mask[3*8], J_pinv[8*3];
6   jacob(q_int, J, fijo_A);
7   // matriz de rotacion 6x6 --> R = [R' 0; 0 R']
8   double R_actual[3*3], R_actual_t[3*3], R[6*6] = {};
9   R_act(q_int, fijo_A, R_actual);
10  MatrixTranspose((float*)R_actual, 3, 3, (float*)R_actual_t);
11
12  for (int i=0; i<3; i++){
13    for (int j=0; j<3; j++){
14      R[i + 6*j] = R_actual_t[i + 3*j];
15      R[i + 6*j + 7*3] = R_actual_t[i + 3*j];
16    }
17
18  Matrix.Multiply((float*)R, (float*)J, 6, 6, 8, (float*)Je);
19
20  for (int j = 0; j < 8; j++){
21    Je_mask[0*8 + j] = Je[1*8 + j];
22    Je_mask[1*8 + j] = Je[3*8 + j];
23    Je_mask[2*8 + j] = Je[5*8 + j];
24  }
25
26  pinv(Je_mask, J_pinv);
27
28  double delta_q[8], J_times_delta_x[8*1];
29
30  Matrix.Multiply((float*)J_pinv, (float*)delta_x, 8, 3, 1, (float*)J_times_delta_x);
31
32  for (int i=0; i<8; i++){
33    delta_q[i] = alpha * J_times_delta_x[i];
34    q_int[i] += delta_q[i];
35  }
36}

```

El primer paso del algoritmo consiste en calcular la Jacobiana del efecto final en coordenadas locales. Se puede calcular la Jacobiana en coordenadas globales haciendo uso de la función `jacob`. El código de la función se puede consultar en el anexo A.1.

La función genera una matriz 6×8 , llamada Jacobiana, a partir de la configuración articular del robot y dependiendo de qué garra esté fija. La matriz Jacobiana permite saber el incremento de posición que se producirá en el extremo del robot con un incremento determinado de las posiciones articulares.

Sin embargo, esta Jacobiana expresa los movimientos en coordenadas globales. Para obtener la Jacobiana en coordenadas locales del efecto final se multiplica una

matriz de rotación 6×6 por la Jacobiana en coordenadas globales (Corke, 2017, p. 232):

$$\mathbf{J}(\mathbf{q})_e = \mathbf{R} \mathbf{J}(\mathbf{q})_0 \quad (70)$$

donde

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_0^T & \mathbf{0}_{3 \times 3} \\ \mathbf{0}_{3 \times 3} & \mathbf{R}_0^T \end{bmatrix} \quad (71)$$

En el código, la matriz de rotación \mathbf{R}_0^T se obtiene con la función R_act:

```

1 void R_act(double q[8], boolean fijo_A, double R[3*3]){
2     double phi_1A, y_A, phi_2A, theta_A, theta_B, phi_2B, y_B,
3         phi_1B, Theta, Phi_A, Phi_B;
4
5     phi_1A = q[0];
6     y_A = q[1];
7     phi_2A = q[2];
8     theta_A = q[3];
9     theta_B = q[4];
10    phi_2B = q[5];
11    y_B = q[6];
12    phi_1B = q[7];
13
14    Theta = theta_A - theta_B;
15    Phi_A = phi_1A - phi_2A;
16    Phi_B = phi_1B - phi_2B;
17
18    // n
19    R[0] = sin(Phi_A)*sin(Phi_B) +cos(Theta)*cos(Phi_A)*cos(Phi_B);
20    R[3] = cos(Phi_A)*sin(Phi_B) -cos(Theta)*sin(Phi_A)*cos(Phi_B);
21    R[6] = -sin(Theta)*cos(Phi_B);
22
23    // o
24    R[1] = sin(Phi_A)*cos(Phi_B) -cos(Theta)*cos(Phi_A)*sin(Phi_B);
25    R[4] = cos(Phi_A)*cos(Phi_B) +cos(Theta)*sin(Phi_A)*sin(Phi_B);
26    R[7] = sin(Theta)*sin(Phi_B);
27
28    // a
29    R[2] = cos(Phi_A)*sin(Theta);
30    R[5] = -sin(Theta)*sin(Phi_A);
31    R[8] = cos(Theta);
32
33
34    if (!fijo_A){
35        // matriz traspuesta
36        double R_t[3*3];
37        MatrixTranspose((float*)R, 3, 3, (float*)R_t);
38        MatrixCopy((float*)R_t, 3, 3, (float*)R);
39    }
40}

```

La función genera la matriz de rotación actual a partir de la configuración articular actual siguiendo el modelo cinemático (ecuación (24)). En el caso de que sea la garra B la que está fija, la matriz se traspone para obtener el modelo cinemático inverso.

Cabe destacar que para realizar las operaciones matriciales y vectoriales se ha utilizado la librería **MatrixMath**. Para operar con la librería, se utilizan vectores en lugar de matrices, indicando el número de filas y columnas de la matriz.

A continuación, a partir de la línea 20 del código de la función principal **act_q**, se le aplica una máscara a la Jacobiana resultante del producto, eliminando los componentes de la posición en x y z y del giro en y . Esto permitirá que, para el cálculo posterior de los incrementos en las articulaciones, se ignore el error en esos componentes, evitando movimientos innecesarios. Los componentes ignorados por la máscara no son necesarios puesto que para realizar el pegado, no se necesita más que la distancia perpendicular a la superficie (traslación en el eje y) y que los planos sean paralelos (giros en x y z).

El siguiente paso consiste en calcular el incremento que se va a producir en cada articulación. Se calcula con la siguiente ecuación:

$$\Delta \mathbf{q} = \alpha \mathbf{J}(\mathbf{q})_e^{-1} \Delta \mathbf{x} \quad (72)$$

Donde $\Delta \mathbf{q}$ es un vector columna de 8 componentes correspondientes al incremento calculado para cada articulación. α es un escalar que determina la velocidad del algoritmo en converger a la solución final. $\mathbf{J}(\mathbf{q})_e^{-1}$ es la matriz Jacobiana en coordenadas locales del efector final invertida. $\Delta \mathbf{x}$ es el vector columna donde está contenida la pose extraída de la matriz de diferencia. De la misma forma que con $\mathbf{J}(\mathbf{q})_e^{-1}$, en $\Delta \mathbf{x}$ también se han eliminado las filas correspondientes a la posición en x y z y el giro en y .

Para calcular la inversa de la Jacobiana, al ser una matriz no cuadrada, no se puede calcular de la manera convencional. La solución es utilizar una pseudoinversa. Existen varias formas de generalizar la operación de invertir una matriz, pero la más utilizada es la inversa de Moore-Penrose. Consiste en obtener una solución por mínimos cuadrados de un sistema lineal de ecuaciones. El cálculo que se realiza para obtener la pseudoinversa depende del número de filas y columnas que tiene la matriz a invertir. Para esta aplicación, al tener filas linealmente independientes, se utiliza la siguiente ecuación:

$$\mathbf{A}^\dagger = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1} \quad (73)$$

Se ha implementado como la función **pinv**:

```

1 void pinv(double J[3*8], double X[8*3]){
2     double J_t[8*3], A[3*3];
3     MatrixTranspose((float*)J, 3, 8, (float*)J_t);

```

```

4     Matrix.Multiply((float*)J, (float*)J_t, 3, 8, 3, (float*)A);
5     Matrix.Invert((float*)A, 3);
6     Matrix.Multiply((float*)J_t, (float*)A, 8, 3, 3, (float*)X);
7 }
```

Finalmente, se suman los incrementos calculados en cada articulación para obtener la nueva configuración articular que aproximará la garra a la superficie de pegado.

3.2.6. Transformación de las coordenadas articulares intermedias

Los cálculos de los incrementos en cada articulación explicados en la sección anterior se realizan a partir de las coordenadas articulares intermedias. Para ello, primero es necesario transformar los valores de tensión leídos en cada articulación a coordenadas articulares intermedias.

```

1 void volt2int(double tension[10], double q_int[8]){
2     double r_1A, l_1B, r_2A, l_2B, l_1A, r_1B, l_2A, r_2B, theta_A,
3         theta_B, phi_y[2]; // longitudes articulares en m y rad
4     r_1A = (10.183*tension[0] + 101.21) / 1000; // A(0) es r_1A
5     l_1B = (1.3915*tension[1]*tension[1] + 3.2456*tension[1] +
6         100.98) / 1000; // B(1) es l_1B
7     r_2A = (11.304*tension[2] + 95.405) / 1000; // C(2) es r_2A
8     l_2B = (1.5535*tension[3]*tension[3] + 2.5814*tension[3] +
9         102.21) / 1000; // D(3) es l_2B
10    l_1A = (9.877*tension[4] + 102.75) / 1000; // E(4) es l_1A
11    r_1B = (10.756*tension[5] + 97.11) / 1000; // F(5) es r_1B
12    l_2A = (10.315*tension[6] + 100.92) / 1000; // G(6) es l_2A
13    r_2B = (10.399*tension[7] + 101.04) / 1000; // H(7) es r_2B
14    theta_A = -1.09*tension[8] + 2.52;
15    theta_B = -1.0537*tension[9] + 3.0345;
16
17    double h=0.07, b=0.025, p=0.0315;
18    double phi_1A, y_1A, phi_2A, y_2A, phi_1B, y_1B, phi_2B, y_2B;
19    solve_parallel_module(b, p, l_1A, r_1A, phi_y);
20    phi_1A = phi_y[0];
21    y_1A = phi_y[1];
22    solve_parallel_module(b, p, l_2A, r_2A, phi_y);
23    phi_2A = phi_y[0];
24    y_2A = phi_y[1];
25    solve_parallel_module(b, p, l_1B, r_1B, phi_y);
26    phi_1B = phi_y[0];
27    y_1B = phi_y[1];
28    solve_parallel_module(b, p, l_2B, r_2B, phi_y);
29    phi_2B = phi_y[0];
30    y_2B = phi_y[1];
31
32    q_int[0] = phi_1A;
33    q_int[1] = y_1A + y_2A - h;
34    q_int[2] = phi_2A;
35    q_int[3] = theta_A;
36    q_int[4] = theta_B;
37    q_int[5] = phi_2B;
```

```

35     q_int[6] = y_1B + y_2B - h;
36     q_int[7] = phi_1B;
37 }
```

La función toma como entrada los valores leídos correspondientes a las tensiones en cada actuador lineal o potenciómetro en el caso las articulaciones de las caderas.

A continuación, mediante la curva de calibración previamente obtenida, se calculan las longitudes en metros, en el caso de los actuadores lineales, y los giros en radianes, en el caso de las articulaciones de las caderas.

Seguidamente, a partir de la línea de código 16, se resuelve la cinemática directa de cada módulo paralelo para obtener las variables articulares intermedias con la función `solve_parallel_module`.

```

1 void solve_parallel_module(double b, double p, double l, double r,
2     double* phi_y){
3     double phi, y;
4     double epsilon = 0.00001;
5     double r0, r1;
6     phi = atan2(r-l,2*b);
7     y = (l+r)/2;
8     r0 = (p*cos(phi)-b)*(p*cos(phi)-b) + (y+p*sin(phi))*(y+p*sin(phi)) - r*r;
9     r1 = (p*cos(phi)-b)*(p*cos(phi)-b) + (y-p*sin(phi))*(y-p*sin(phi)) - l*l;
10    double Delta_phi, Delta_y;
11    double phi_previo = 0, y_previo = 0;
12    while(sqrt((phi-phi_previo)*(phi-phi_previo)+(y-y_previo)*(y-y_previo)) > epsilon){
13        Delta_phi = (p*(r0 + r1)*sin(phi) + y*(r1 - r0))/(4*p*(y*y*cos(phi) - b*p*sin(phi)*sin(phi)));
14        Delta_y = (y*(r0 + r1)*cos(phi) + b*(r1 - r0)*sin(phi))/(4*(b*p*sin(phi)*sin(phi) - y*y*cos(phi)));
15        phi_previo = phi;
16        y_previo = y;
17        phi = phi + Delta_phi;
18        y = y + Delta_y;
19        r0 = (p*cos(phi)-b)*(p*cos(phi)-b) + (y+p*sin(phi))*(y+p*sin(phi)) - r*r;
20        r1 = (p*cos(phi)-b)*(p*cos(phi)-b) + (y-p*sin(phi))*(y-p*sin(phi)) - l*l;
21    }
22    phi_y[0] = phi;
23    phi_y[1] = y;
}
```

La función `solve_parallel_module` resuelve la cinemática directa de los módulos paralelos. Es decir, calcula las variables articulares intermedias y y φ a partir de las longitudes de los actuadores lineales l y r . La función ha sido proporcionada por los tutores de este TFG.

Se utiliza el método de Newton. A partir de una solución inicial cercana a la

final, se calculan iterativamente incrementos a partir de las ecuaciones que definen la cinemática de los módulos paralelos (ecuaciones (16) y (17)).

Sin embargo, la función que actualiza los valores de la configuración articular explicada en la sección anterior, también devuelve las coordenadas articulares nuevas como intermedias. Por lo cual, será necesario transformarlas una vez más a los valores de tensión que se enviarán a los actuadores como referencia.

```

1 void int2volt(double q_int[8], double tension[10]){
2     double h=0.07, b=0.025, p=0.0315;
3     double phi_1A, y_1A, phi_2A, y_2A, phi_1B, y_1B, phi_2B, y_2B,
4         theta_A, theta_B;
5     phi_1A = q_int[0];
6     y_1A = (q_int[1] + h)/2;
7     y_2A = (q_int[1] + h)/2;
8     phi_2A = q_int[2];
9     theta_A = q_int[3];
10    theta_B = q_int[4];
11    phi_2B = q_int[5];
12    y_1B = (q_int[6] + h)/2;
13    y_2B = (q_int[6] + h)/2;
14    phi_1B = q_int[7];
15
16    double r_1A, l_1B, r_2A, l_2B, l_1A, r_1B, l_2A, r_2B;
17    ik_modulo(b, p, phi_1A, y_1A, l_1A, r_1A);
18    ik_modulo(b, p, phi_2A, y_2A, l_2A, r_2A);
19    ik_modulo(b, p, phi_2B, y_2B, l_2B, r_2B);
20    ik_modulo(b, p, phi_1B, y_1B, l_1B, r_1B);
21
22    double longitudes_mm[8];
23    longitudes_mm[0] = l_1A*1000;
24    longitudes_mm[1] = r_1A*1000;
25    longitudes_mm[2] = l_2A*1000;
26    longitudes_mm[3] = r_2A*1000;
27    longitudes_mm[4] = l_1B*1000;
28    longitudes_mm[5] = r_1B*1000;
29    longitudes_mm[6] = l_2B*1000;
30    longitudes_mm[7] = r_2B*1000;
31    length2voltage(longitudes_mm, tension);
32
33    tension[8] = -0.918*theta_A + 2.32;
34    tension[9] = -0.949*theta_B + 2.88;
35 }
```

La función empieza calculando, mediante la cinemática inversa de cada módulo paralelo, los valores de las longitudes en cada actuador lineal. La función que se encarga de resolver la cinemática inversa de cada módulo paralelo es la siguiente:

```

1 void ik_modulo(double b, double p, double phi, double y, double &l,
2                 double &r){
3     l = sqrt(pow(p*cos(phi) - b, 2) + pow(y - p*sin(phi), 2));
4     r = sqrt(pow(p*cos(phi) - b, 2) + pow(y + p*sin(phi), 2));
5 }
```

La función es muy sencilla, utiliza las ecuaciones de la cinemática inversa ex-

puestas en (16) y (17) para calcular los valores de la longitud de cada actuador lineal que consiguen el ángulo y distancia introducidos como argumento.

Finalmente, a partir de la línea 21 del código de la función `int2volt`, se transforman los valores de las longitudes de cada actuador lineal de metros a milímetros, para posteriormente mediante la función `length2voltage` transformarlos a los valores de tensión que satisfacen las longitudes. De la misma forma, independientemente se calcula la tensión necesaria para obtener los ángulos en radianes de las variables θ_A y θ_B introducidas como entrada.

```

1 void length2voltage(double *lengths, double *v){
2     double l_1A, r_1A, l_2A, r_2A;
3     double l_1B, r_1B, l_2B, r_2B;
4     l_1A = lengths[0];
5     r_1A = lengths[1];
6     l_2A = lengths[2];
7     r_2A = lengths[3];
8     l_1B = lengths[4];
9     r_1B = lengths[5];
10    l_2B = lengths[6];
11    r_2B = lengths[7];
12
13    v[0] = 0.098*r_1A - 9.927;
14    v[1] = -0.00109707*l_1B*l_1B + 0.36962512*l_1B - 25.88490173;
15    v[2] = 0.088*r_2A - 8.3784;
16    v[3] = -0.001315*l_2B*l_2B + 0.426487*l_2B - 29.616165;
17    v[4] = 0.101226*l_1A - 10.400062;
18    v[5] = -0.000422*r_1B*r_1B + 0.202672*r_1B - 16.065812;
19    v[6] = 0.096884212*l_2A - 9.77588537;
20    v[7] = 0.0961003*r_2B - 9.7081615;
21 }
```

3.2.7. Implementación en el código original

El algoritmo completo visto en la figura 21 se debe implementar en el código original de control del robot. Se ha resuelto de la siguiente forma:

```

1 if (aproximacion){
2     calc_T(T);
3
4     tr2delta(T, delta_x);
5     // mascara en delta_x
6     delta_x_mask[0] = delta_x[1];
7     delta_x_mask[1] = delta_x[3];
8     delta_x_mask[2] = delta_x[5];
9
10    volt2int(q_filtrado, q_int);
11    act_q(q_filtrado, T, delta_x_mask, fijo_A, q_desired);
12    int2volt(q_int, q_desired);
13
14    if (check_norm(delta_x_mask)){
15        Serial.println("Fin de la aproximacion");
16        aproximacion = false;
```

```

17     reset_integral();
18 }
19 }
```

El código se encuentra dentro de la función `loop`, la cual se ejecuta en un bucle sin fin mientras el Arduino está encendido. Dentro de la función `loop` se encuentran todas las otras instrucciones que permiten realizar el control del robot, provenientes de otros trabajos previos a este TFG. Entre ellas se encuentran funciones que realizan la lectura de comandos a través de serial, el control PID de los 10 grados de libertad del robot, el envío de las acciones de control y el control manual a través del mando entre otras. Para mejorar la compatibilidad del nuevo código con el previo, se utiliza una variable booleana llamada `aproximacion` que hará que solo se ejecute el código cuando el robot esté en “modo aproximación”.

El código es muy sencillo, ya que la parte de programación extensa se ha hecho en las funciones llamadas. Primero, con `calc_T`, se calcula la matriz de diferencia `T`, la cual contiene la información de la posición y orientación del plano de la superficie vista por los sensores.

Seguidamente, mediante la función `tr2delta` se calcula el vector de 6 componentes correspondientes a la posición y orientación extraídas de la matriz. A este vector, se le aplica una máscara, eliminando los componentes de la posición en `x` y `z` y el giro en `y`, ya que es información no necesaria para la aplicación. De esta forma se simplifica y optimiza los cálculos siguientes.

A continuación, con la función `act_q` se actualizan los valores de las coordenadas articulares del robot. Para ello, sin embargo, será necesario transformar los valores de tensión en cada articulación a coordenadas articulares intermedias mediante la función `volt2int`. De la misma forma, después de actualizar la configuración articular, será necesario transformar las coordenadas articulares intermedias a los valores de tensión en cada articulación mediante `int2volt`.

Finalmente, se comprueba la condición de fin del algoritmo. Con `check_norm`, se evalúa el valor de la norma del vector de diferencia. Si este es menor a un valor ϵ , finaliza el algoritmo cambiando el valor de la variable `aproximacion` a `false`, saliendo así del “modo aproximación” y no volviendo a ejecutar el código hasta que se vuelva a activar externamente. También, cuando se da la condición de fin del algoritmo, se reinicia la acción integral del controlador utilizado y se imprime en el serial un mensaje informativo.

3.3. Simulación de la aproximación

Antes de implementar el código de Arduino en el robot, se ha realizado una simulación para comprobar el correcto funcionamiento del algoritmo.

La simulación es una parte muy importante y necesaria en cualquier ámbito de la ingeniería, ya sea en investigación o al realizar proyectos. Consiste en modelar

matemáticamente y reproducir un sistema con el objetivo de entender y comprobar el correcto funcionamiento del mismo. Realizar una simulación precisa evitará futuros problemas relacionados con comportamientos no esperados del sistema que no se habrían detectado hasta la implementación final.

Se ha elegido MATLAB como entorno de programación por la conveniencia y facilidad de este lenguaje comparado a otros. Al tener funciones para cálculo vectorial y matricial de forma nativa y soporte para librerías, es el lenguaje idóneo para la programación de la simulación.

```

1 close all
2 clear
3 clc
4
5 load('hyrecro.mat')
6 hyrecro.plot(q_home,'workspace',[xmin xmax ymin ymax zmin zmax], ,
7   'noname')
8 xlim([-0.2 0.4]);
9 zlim([-0.2 0.4]);
10 hold on
11 vert = [-0.1 -0.05 0;0.2 -0.05 0;0.2 0.05 0;-0.1 0.05 0; -0.1 -0.05
12   -0.1;0.2 -0.05 -0.1;0.2 0.05 -0.1;-0.1 0.05 -0.1];
13 vert_columna = [0.2 -0.05 -0.2;0.3 -0.05 -0.2;0.3 0.05 -0.2; 0.2
14   0.05 -0.2;0.2 -0.05 0.4;0.3 -0.05 0.4;0.3 0.05 0.4;0.2 0.05
15   0.4];
16 fac = [1 2 6 5;2 3 7 6;3 4 8 7;4 1 5 8;1 2 3 4;5 6 7 8];
17 patch('Vertices',vert,'Faces',fac,'FaceVertexCData', ones(6,3)
18   *255/2,'FaceColor','flat');
19 patch('Vertices',vert_columna,'Faces',fac,'FaceVertexCData', ones
20   (6,3)*255/2,'FaceColor','flat');
21
22 %parametros simulacion
23 epsilon = 0.001;
24 alpha = 0.1;
25
26 %pose deseada
27 posicion = [0.2,0,0.2]';
28 n = [0 0 1]';
29 o = [-1 0 0]';
30 a = cross(n,o);
31 orientacion = [n,o,a];
32 T_des = [orientacion,posicion,[0 0 0 1]];
33
34 %pose actual
35 q = [pi/8 0.15 pi/4 0 -pi/8 pi/8 0.1 -pi/6];
36 %q = [pi/6 0.15 -pi/4 0 0 pi/8 0.12 -pi/12];
37 T_act = se32mat(hyrecro.fkine(q));
38 T = inv(T_act)*T_des;
39 delta_x = tr2delta(T);
40 delta_x([1 3 5]) = [];
41
42 while norm(delta_x) > epsilon
43   R = T_act(1:3,1:3);
44 end

```

```

41 J = [R.', zeros(3,3); zeros(3,3), R.] * hyrecro.jacob0(q);
42 J([1 3 5], :) = [];
43
44 delta_q = alpha*pinv(J)*delta_x;
45 q = q + delta_q';
46
47 hyrecro.plot(q, 'workspace', [xmin xmax ymin ymax zmin zmax], ,
48 noname)
48 T_act = se32mat(hyrecro.fkine(q));
49 T = inv(T_act)*T_des;
50 delta_x = tr2delta(T);
51 delta_x([1 3 5]) = [];
52 end

```

Para la simulación se ha utilizado la librería Robotics Toolbox de Peter Corke. Además de hacer una representación gráfica, la librería proporciona una multitud de herramientas y funciones para hacer operaciones y conversiones a partir de vectores y matrices. Para la simulación se hará uso de la representación gráfica en 3D de la librería y el cálculo de la cinemática directa.

Para ello, previamente se han creado y asignado variables necesarias para la representación. Todas ellas están almacenadas en el archivo `hyrecro.mat`, el cual se utiliza para cargar las variables al principio del código. Las variables son objetos propios de la librería (variable `SerialLink`), límites de los ejes para la representación gráfica, la posición por defecto del robot y los vértices que se utilizarán para dibujar las columnas de apoyo del robot en la animación.

La variable `SerialLink`, como se ha comentado, es un tipo de variable propia de la librería y contiene toda la información geométrica y cinemática del robot. Es un objeto formado por varios `Link`, tantos como articulaciones tenga el robot. La subclase `Link` se forma a partir de los parámetros Denavit-Hartenberg de cada articulación, pudiendo obtener así un modelo cinemático completo del robot (Corke, 2017, p. 197-200).

Seguidamente, se definen los vértices y se dibujan las columnas donde se fijará el robot durante la animación en la representación gráfica.

A continuación, se definen los parámetros `epsilon` y `alpha`, los cuales también son utilizados en la implementación del algoritmo en Arduino. La variable `epsilon` determina el valor máximo que puede tener la norma del vector de diferencia para dar por concluido el algoritmo. El valor de `alpha` determina la velocidad a la que convergerá el algoritmo.

A partir de la línea 22, se define la pose a la que se quiere llevar el efector final y la configuración articular inicial del robot. Se calcula la matriz de transformación a partir de la configuración articular mediante la función `fkine`.

La función calcula la matriz de transformación del efector final visto desde la base del robot haciendo uso de su cinemática directa. La librería genera un modelo cinemático a partir de los parámetros con los que se ha creado el objeto `SerialLink`

equivalente al modelo cinemático estudiado en el apartado 2.1.2.

La función `fkin` devuelve un objeto del tipo `SE3`, propio de la librería. Por lo tanto, es necesario transformarlo a una matriz convencional para operar con ella. Para ello se ha creado la función `se32mat`, que transforma un objeto `SE3` en una matriz convencional de MATLAB.

```

1 function [mat] = se32mat(se3)
2 mat = [se3.n, se3.o, se3.a, se3.t; 0 0 0 1];
3 end

```

De vuelta al código principal, en la línea 34, se emula la matriz de diferencia T obtenida a partir de las medidas de los sensores. La matriz de diferencia equivale a la matriz de transformación del sistema de referencia deseado visto desde el sistema de referencia actual del efecto final. En la siguiente imagen se pueden ver los diferentes sistemas de referencia y como se relacionan mediante matrices de transformación:

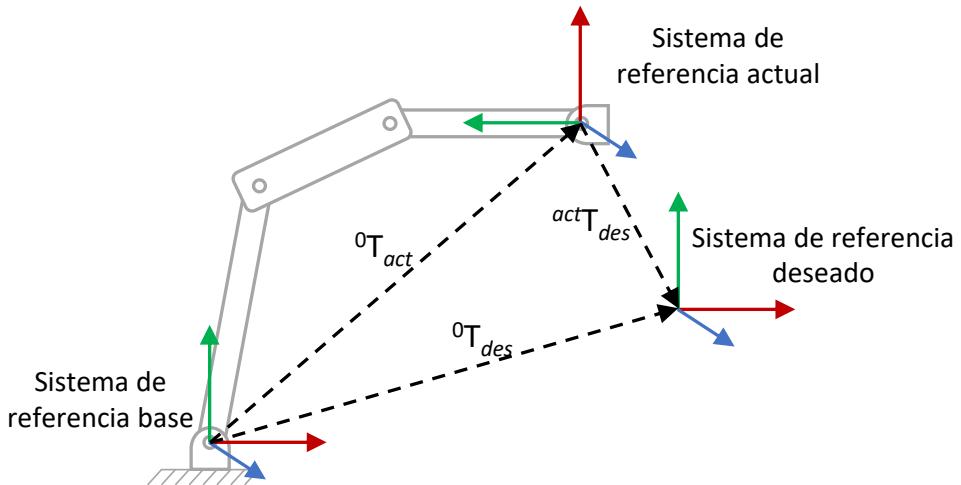


Figura 23: Relación entre las matrices de transformación.

En la imagen se comprueba que existe una ecuación que relaciona las 3 matrices de transformación:

$${}^0T_{act} {}^{act}T_{des} = {}^0T_{des} \quad (74)$$

Por lo que:

$$T = {}^{act}T_{des} = {}^0T_{act}^{-1} {}^0T_{des} \quad (75)$$

Seguidamente, se extrae el vector de diferencia de la matriz mediante la función `tr2delta`. El vector, de 6 componentes, contiene la información de la posición y orientación en x , y y z . De la misma forma que en el código del robot real en

Arduino, se le aplica una máscara al vector, eliminando los componentes relativos a la posición en x y z y a la orientación en y .

```

1 function delta = tr2delta(A)
2 v = 0.5*[A(3,2)-A(2,3); A(1,3)-A(3,1); A(2,1)-A(1,2)];
3 delta = [A(1:3,4); v];
4 end

```

El método utilizado es similar al empleado en la función del programa de Arduino. Se explica en el apartado 3.2.3.

A continuación, el programa entra en un bucle del que saldrá cuando la norma del vector de diferencia sea menor o igual a `epsilon`.

En cada iteración del bucle, se empieza extrayendo la matriz de rotación R , que está contenida en la matriz de transformación del efecto final visto desde la base. Se construye una matriz de rotación 6×6 por la que se multiplicará la matriz Jacobiana en coordenadas globales para obtener la matriz Jacobiana en coordenadas locales, tal como se ha explicado en las ecuaciones (70) y (71). Tal como se hace con el vector de diferencia, se eliminan las filas 1, 3 y 5 de la matriz, correspondientes a la posición en x y z y a la orientación en y .

En la línea 44 del código, se calcula el incremento necesario en cada articulación para acercar la garra a la pose deseada. Su cálculo se hace de la misma forma que en la ecuación (72). Después, se actualiza el valor de las coordenadas articulares, sumándole a cada una el incremento calculado.

Finalmente, a partir de la línea 47, se dibuja el robot con la configuración articular actual haciendo uso de la librería Robotics Toolbox. Para acabar, se calcula la nueva matriz de diferencia y , a partir de ella, el nuevo vector de diferencia, aplicándole de nuevo la máscara para eliminar los componentes no deseados.

Resultados de la simulación

Se han realizado 2 simulaciones para comprobar el comportamiento del algoritmo con 2 configuraciones articulares y situaciones distintas.

Los resultados para la simulación con una configuración articular inicial $q_0 = (\pi/8, 0.15, \pi/4, 0, -\pi/8, \pi/8, 0.1, -\pi/6)$ y unos parámetros $\alpha = 0.1$ y $\varepsilon = 0.001$ son los siguientes:

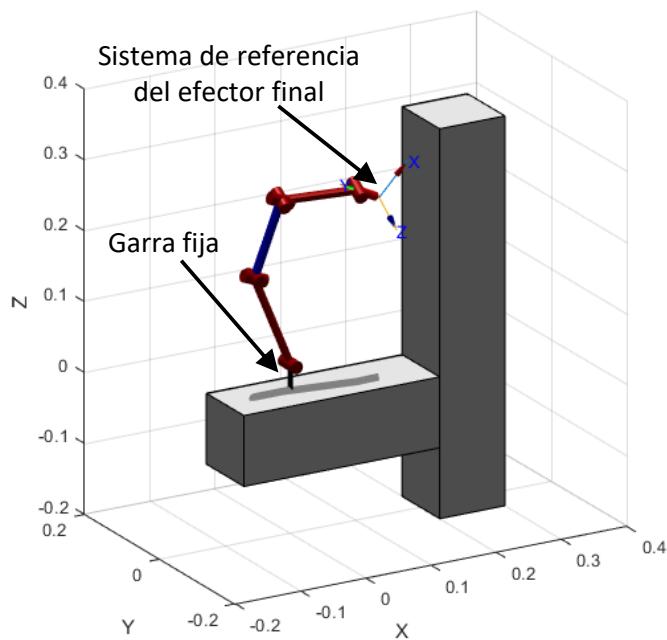


Figura 24: Configuración articular inicial (simulación 1).

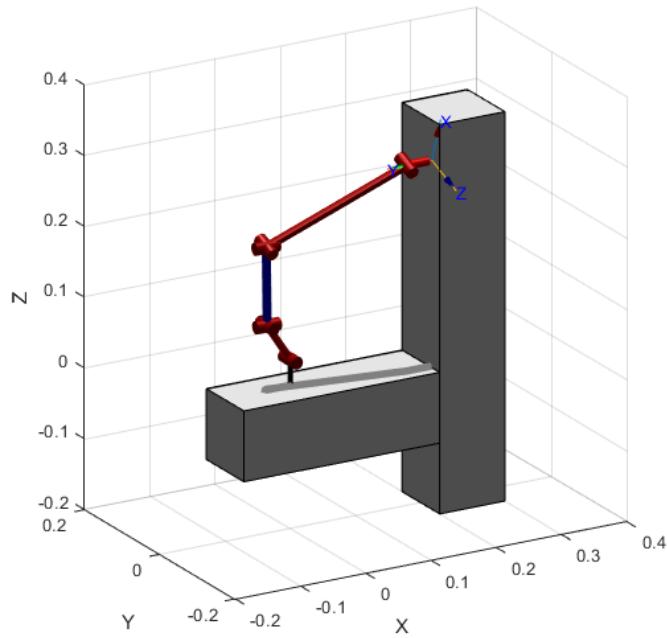


Figura 25: Fin del algoritmo (simulación 1).

Este escenario representa la entrada en el modo de aproximación tras el previo acercamiento manual por parte del operario. El operario acerca la garra libre hasta una posición arbitraria cercana a la superficie y el algoritmo se encarga de completar la aproximación.

En la segunda simulación, se ha utilizado una configuración articular inicial $q_0 = (\pi/6, 0.15, -\pi/4, 0, 0, \pi/8, 0.12, -\pi/12)$ y unos parámetros $\alpha = 0.1$ y $\varepsilon = 0.001$:

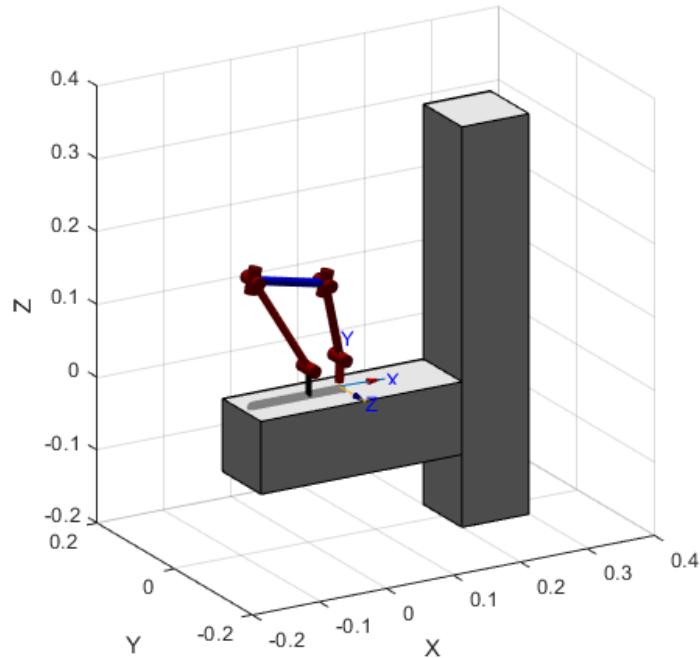


Figura 26: Configuración articular inicial (simulación 2).

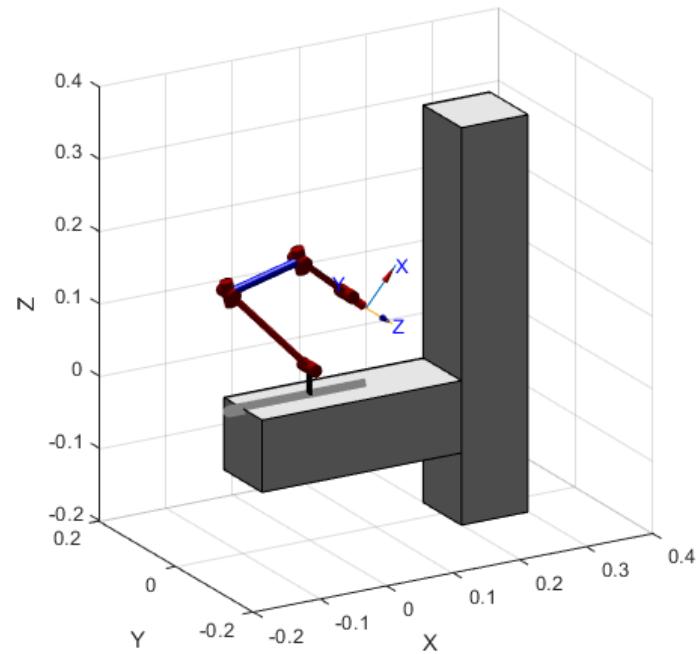


Figura 27: Instante intermedio (simulación 2).

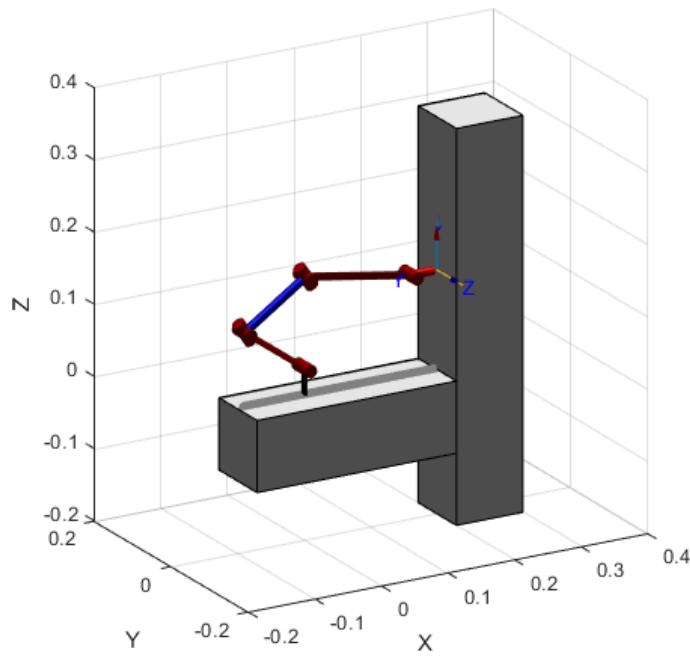


Figura 28: Fin del algoritmo (simulación 2).

En este caso, se simula un cambio de plano completamente automatizado. El robot parte de una posición con ambas garras pegadas en un mismo plano y realiza el cambio de plano.

En el caso real, este proceso no sería posible solamente a partir de los datos de los sensores, ya que estos no siempre están tomando las medidas respecto al plano al que se busca pegar la garra móvil. Mientras que en la simulación se obtiene la matriz de diferencia como diferencia de las matrices de las 2 poses absolutas conocidas (actual y deseado en la figura 23), en la aplicación real se hace a partir de los datos tomados por los sensores.

Por lo tanto, la primera parte del cambio de plano en el caso real debería realizarse obteniendo la matriz de diferencia de forma aproximada al conocerse la ubicación relativa al plano al que se quiere llegar. Cuando la garra estuviese cercana al plano, los sensores empezarían a tomar medidas y el algoritmo se reanudaría obteniendo la matriz de diferencia como se ha explicado.

3.4. Resultados

Finalmente, se ha implementado el código de Arduino explicado a lo largo del capítulo en el prototipo del robot HyReCRo.

Manteniendo fija la garra B, se lleva la garra A a una posición y orientación arbitrarias mediante el control manual del efecto final haciendo uso del mando y

el método descrito en la sección 2.2. Cuando la garra se encuentra en una posición y orientación próximas a la superficie a la que se va a realizar la aproximación, se da la instrucción para el comienzo del algoritmo.

El experimento mostrado corresponde a una aproximación iniciada con una configuración de las variables articulares intermedias inicial de $(0.21, 0.19, 0.54, -0.19, -0.01, -0.15, 0.16, 0.11)$ radianes y unos parámetros $\alpha = 0.1$ y $\varepsilon = 0.002$.

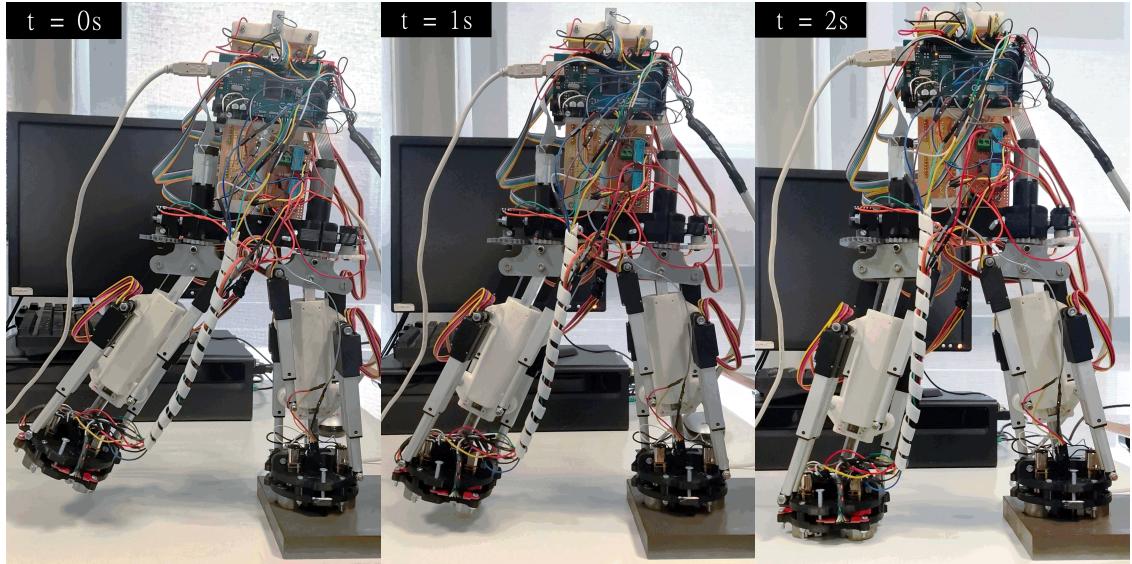


Figura 29: Vista del robot completo durante la aproximación.

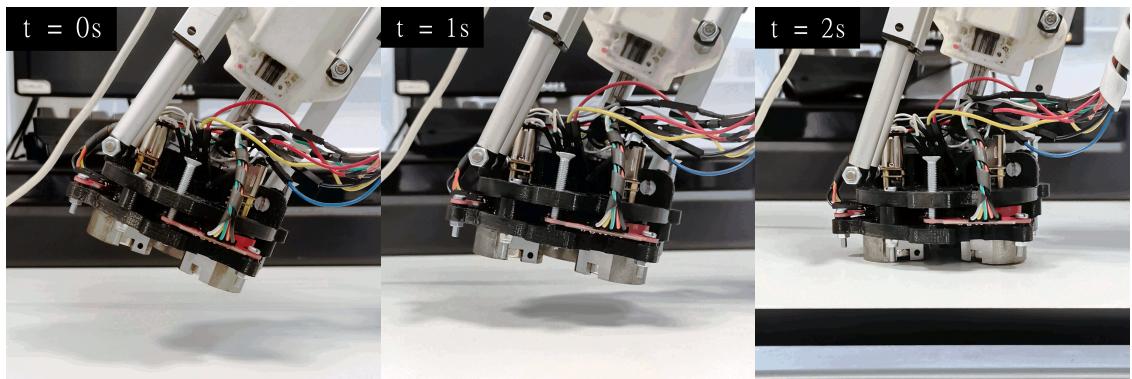


Figura 30: Detalle de la garra durante la aproximación.

En las figuras se muestran 3 instantes durante la ejecución del algoritmo y la aproximación de la garra. Se puede apreciar que se corrige correctamente la posición y orientación del sistema de referencia de la garra móvil relativo a la superficie, vista desde los sensores instalados, hasta pegar completamente la garra contra la superficie. El algoritmo concluye cuando la norma de los errores extraídos a partir de la información proporcionada por los sensores es menor al valor de ε .

Sin embargo, debido a la gran redundancia del robot, el algoritmo no es completamente robusto. Para algunas configuraciones articulares iniciales, la solución a la que converge el algoritmo sitúa la garra móvil en una posición demasiado cercana

a la garra fija, provocando una colisión entre ambas extremidades del robot. Una posible solución para mejorar el algoritmo y evitar la colisión entre extremidades se mencionará en el capítulo 5.

4. Calibración en cadena cerrada

4.1. Introducción

Los métodos de calibración sirven para mejorar la precisión de un robot sin tener que modificar la estructura mecánica de este. A través de software, se puede modificar el sistema de control del robot para que tenga en cuenta los errores debidos a la fabricación o el desgaste. Al calibrar un robot, se asume que existe un error entre la pose del efecto final real y la que debería ser según su modelo cinemático nominal (ideal). El objetivo final de la calibración es identificar los errores en los parámetros geométricos del robot que producen esta diferencia entre el modelo real y nominal (Roth et al., 1987).

Existen varios métodos de calibración, entre ellos los métodos de cadena abierta y cerrada. Los métodos de cadena abierta consisten en medir externamente la posición y orientación del efecto final del robot para obtener los errores. Esto requiere instrumentos muy precisos y que se haga de forma regular, por lo que son métodos muy lentos y costosos (Meggiolaro et al., 2002). Por otro lado, los métodos de cadena cerrada consisten en, únicamente con los valores de las coordenadas articulares del robot, identificar los errores que producen la diferencia (Everett and Lin, 1988).

En este TFG, se ha optado por un método de calibración de cadena cerrada para mejorar la precisión del robot HyReCRo. La estructura mecánica del robot compromete la precisión que tiene al controlar su efecto final, debido a los materiales e imprecisiones durante su fabricación. El robot está completamente formado por piezas impresas en plástico con una impresora 3D. Cuando no se utilizan impresoras extremadamente precisas, es inevitable que aparezca algún error entre el diseño original y el producido, por lo que es necesario realizar una calibración para identificarlos.

Concretamente, para la calibración, se ha seguido el método expuesto en (Bennett and Hollerbach, 1991), llamado *Single Endpoint Contact* (SEC). El método consiste en fijar el extremo libre del robot en una posición fija, pero con rotación libre, comportándose como una articulación esférica a efectos prácticos. Con la restricción en el efecto final, el robot varía su configuración conociendo los valores de todas sus coordenadas articulares. A partir de los datos recogidos por diferentes configuraciones, se resuelve un sistema mediante mínimos cuadrados para obtener los errores buscados.

4.2. Método

Como ya se ha explicado, el objetivo de la calibración es identificar los errores que producen la diferencia entre el modelo nominal y el real. La pose del efecto final de un robot se puede representar con una matriz de transformación homogénea T 4×4 que contiene la traslación y la rotación del efecto final respecto a la base

del robot. La translación y la rotación se pueden parametrizar en un vector, llamado \mathbf{X} , de 6 elementos, 3 correspondientes a la translación y los 3 restantes a la rotación.

Los 3 primeros elementos son el desplazamiento en los ejes \mathbf{x} , \mathbf{y} y \mathbf{z} del efecto final respecto a la base, equivalentes a los 3 primeros elementos de la cuarta columna de la matriz \mathbf{T} . La parte rotacional de la matriz \mathbf{T} (la matriz \mathbf{R} 3×3 contenida en las 3 primeras filas y columnas) es el resultado del producto de 3 giros determinados en un orden específico. El valor de los 3 giros determinan los últimos elementos del vector \mathbf{X} .

Para parametrizar la orientación, se ha decidido seguir los ángulos de Euler XYZ. Es decir, la matriz \mathbf{R} se define como el producto de un giro α en \mathbf{x} , un giro β en \mathbf{y} y un giro γ en \mathbf{z} , en ese orden:

$$\mathbf{R} = X(\alpha)Y(\beta)Z(\gamma) \quad (76)$$

De forma inversa, se pueden identificar los valores de α , β y γ conociendo \mathbf{R} y la siguiente igualad:

$$\mathbf{R} = \begin{bmatrix} \cos \beta \cos \gamma & -\cos \beta \sin \gamma & \sin \beta \\ \cos \alpha \sin \gamma + \cos \gamma \sin \alpha \sin \beta & \cos \alpha \cos \gamma - \sin \alpha \sin \beta \sin \gamma & -\cos \beta \sin \alpha \\ \sin \alpha \sin \gamma - \cos \alpha \cos \gamma \sin \beta & \cos \gamma \sin \alpha + \cos \alpha \sin \beta \sin \gamma & \cos \alpha \cos \beta \end{bmatrix} \quad (77)$$

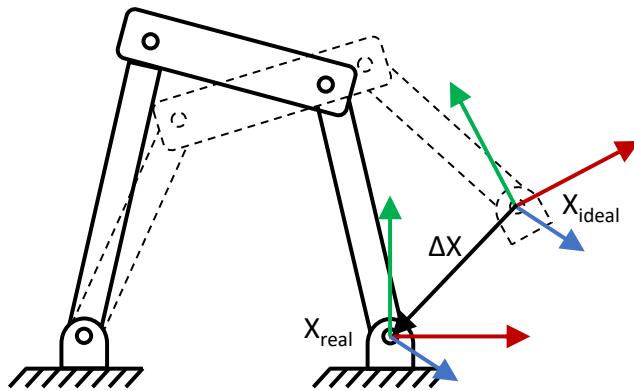


Figura 31: Modelo real y modelo ideal.

De la misma forma, se puede definir el error entre el modelo real y el modelo nominal o ideal mediante el vector $\Delta \mathbf{X}$:

$$\Delta \mathbf{X} = \mathbf{X}^{real} - \mathbf{X}^{ideal} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \alpha \\ \Delta \beta \\ \Delta \gamma \end{bmatrix} \quad (78)$$

Al tratarse de errores generalizados pequeños, el error entre modelos $\Delta \mathbf{X}$ puede calcularse en función de los parámetros de error generalizados, vector ε que recoge los errores que producen la diferencia de pose entre modelos, y la matriz Jacobiana del error en el efecto final (Zhuang et al., 1999):

$$\Delta \mathbf{X} = \mathbf{J}_e \varepsilon \quad (79)$$

También se puede expresar $\Delta \mathbf{X}$ en función de la suma de los errores producidos por los errores geométricos y los errores en las articulaciones:

$$\Delta \mathbf{X} = \mathbf{J}_g \Delta \mathbf{g} + \mathbf{J}_q \Delta \mathbf{q} \quad (80)$$

En este TFG, se supondrá que los errores en las articulaciones son despreciables. Esto significa que la diferencia de pose entre el modelo real y el nominal está producida únicamente por los errores geométricos en la estructura del robot. La ecuación resultante es la siguiente:

$$\Delta \mathbf{X} = \mathbf{J}_g \Delta \mathbf{g} \quad (81)$$

Las variables geométricas del robot Hyrecro, como se ha visto en la sección 2.1, son t , h , b y p . Suponiendo que los errores en las variables h , b y p son iguales en cada pata y módulo paralelo, se puede definir el vector de los errores generalizados $\Delta \mathbf{g}$ como:

$$\Delta \mathbf{g} = \begin{bmatrix} \Delta h \\ \Delta t \\ \Delta b \\ \Delta p \end{bmatrix} \quad (82)$$

Como ya se ha comentado de forma superficial, la calibración en cadena cerrada consiste en fijar el extremo libre del robot e identificar los errores a partir de los valores de las coordenadas articulares. En el método SEC original, se fija el efecto final en una articulación esférica, restringiendo la posición pero no la orientación. Esto permite a robots no redundantes cambiar su configuración articular sin modificar la posición de su efecto final.

En el robot Hyrecro, sin embargo, no es necesario dejar esta libertad en la orientación, ya que al ser un robot redundante puede realizar movimientos en sus articulaciones sin modificar la pose de su extremo. Por lo cual, se puede aprovechar la redundancia del robot fijando completamente ambas garras de forma natural y realizando los movimientos necesarios para la calibración.

Para realizar la calibración, se leen los valores de las coordenadas articulares en m configuraciones articulares distintas. Para compensar los efectos del ruido en la lectura, el número de configuraciones m debe ser bastante mayor al número de grados de libertad del robot.

Conociendo la configuración articular del robot, se pueden calcular las variables \mathbf{X}^{ideal} y \mathbf{J}_g de la ecuación (81) al conocerse el modelo cinemático del robot. Por lo cual, conviene volver a escribir el vector $\Delta\mathbf{X}$ como la diferencia entre poses de ambos modelos (78) y reescribir la ecuación juntando las incógnitas en un mismo lado:

$$-\mathbf{X}^{ideal} = \mathbf{J}_g \Delta\mathbf{g} - \mathbf{X}^{real} \quad (83)$$

Al utilizar el método de calibración en cadena cerrada no es necesario conocer la posición y orientación a la que se fija la garra, sino que se trata como incógnita a determinar. Siendo las variables $\Delta\mathbf{g}$ y \mathbf{X}^{real} ambas constantes e incógnitas, se pueden concatenar, añadiendo a su vez una matriz identidad negativa a la Jacobiana para mantener la igualdad:

$$-\mathbf{X}^{ideal} = [\mathbf{J}_g \quad -\mathbf{I}_{6 \times 6}] \begin{bmatrix} \Delta\mathbf{g} \\ \mathbf{X}^{real} \end{bmatrix} \quad (84)$$

$$-\mathbf{X}^{ideal} = \mathbf{J}_g^* \Delta\mathbf{g}^* \quad (85)$$

Para m configuraciones, representadas como m vectores que almacenan los valores articulares de cada configuración ($\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_m$), la ecuación (85) se puede escribir m veces:

$$\begin{bmatrix} -\mathbf{X}^{ideal}(\mathbf{q}_1) \\ -\mathbf{X}^{ideal}(\mathbf{q}_2) \\ \vdots \\ -\mathbf{X}^{ideal}(\mathbf{q}_m) \end{bmatrix} = \begin{bmatrix} \mathbf{J}_g^*(\mathbf{q}_1) \\ \mathbf{J}_g^*(\mathbf{q}_2) \\ \vdots \\ \mathbf{J}_g^*(\mathbf{q}_m) \end{bmatrix} \Delta\mathbf{g}^* \quad (86)$$

Agrupando en vectores columna:

$$-\mathbf{X}_t^{ideal} = \mathbf{J}_t^* \Delta\mathbf{g}^* \quad (87)$$

Como los errores generalizados y la pose real son constantes, es posible resolver el sistema y estimar el valor de $\Delta\mathbf{g}^*$ mediante una solución de mínimos cuadrados:

$$\Delta\mathbf{g}^* = -(\mathbf{J}_t^{*T} \mathbf{J}_t^*)^{-1} \mathbf{J}_t^{*T} \mathbf{X}_t^{ideal} \quad (88)$$

donde, cabe recordar, el vector de errores generalizados está formado por:

$$\Delta\mathbf{g}^* = \begin{bmatrix} \Delta h \\ \Delta t \\ \Delta b \\ \Delta p \\ x^{real} \\ y^{real} \\ z^{real} \\ \alpha^{real} \\ \beta^{real} \\ \gamma^{real} \end{bmatrix} \quad (89)$$

4.2.1. Jacobiana de Identificación

La matriz Jacobiana \mathbf{J}_e introducida en la ecuación (79) relaciona la diferencia de pose que existe entre el modelo real y el nominal y los errores geométricos y articulares que producen esta diferencia. La matriz, llamada matriz Jacobiana de Identificación en (Zhuang et al., 1999), se obtiene a partir de los valores de las coordenadas articulares y el modelo cinemático y geométrico del robot explicado en la sección 2.1. En esta sección se explicará como se construye la matriz.

Partiendo de la ecuación (81), donde se ha tomado como despreciable el error producido por las articulaciones, se conoce que la matriz tendrá unas dimensiones de 6×4 :

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta \alpha \\ \Delta \beta \\ \Delta \gamma \end{bmatrix} = \mathbf{J}_g \underset{6 \times 4}{=} \begin{bmatrix} \Delta h \\ \Delta t \\ \Delta b \\ \Delta p \end{bmatrix} \quad (90)$$

La matriz Jacobiana, al igual que el vector que define la pose, se puede dividir en dos en función de la posición y la orientación. Se empezará construyendo las 3 primeras filas, correspondientes a la posición.

A partir del modelo cinemático del robot, estudiado en la sección 2.1.2, la posición del efecto final del robot viene dado por la ecuación (23):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = y_A \begin{bmatrix} \sin \varphi_{1A} \\ \cos \varphi_{1A} \\ 0 \end{bmatrix} + y_B \begin{bmatrix} -\cos \Theta \cos \Phi_A \sin \varphi_{2B} - \sin \Phi_A \cos \varphi_{2B} \\ \cos \Theta \sin \Phi_A \sin \varphi_{2B} - \cos \Phi_A \cos \varphi_{2B} \\ \sin \Theta \sin \varphi_{2B} \end{bmatrix} + t \begin{bmatrix} \cos \theta_A \cos \Phi_A \\ -\cos \theta_A \sin \Phi_A \\ -\sin \theta_A \end{bmatrix}$$

donde, cabe recordar, $y_j = y_{1j} + y_{2j} - h$, $\Phi_j = \varphi_{1j} - \varphi_{2j}$ y $\Theta = \theta_A - \theta_B$. Por lo cual, la posición del robot depende de las variables geométricas t y h y de las variables articulares intermedias del robot (las cuales dependen a su vez de las variables geométricas b y p y de las articulaciones lineales del robot).

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = f(t, h, \varphi_{1A}, y_{1A}, \varphi_{2A}, y_{2A}, \theta_A, \theta_B, \varphi_{2B}, y_{2B}, \varphi_{1B}, y_{1B}) \quad (91)$$

La diferencia de posición entre los modelos se puede expresar como la suma de la diferencia producida por las variables geométricas t y h y la producida por las variables articulares intermedias del robot. Las variables θ_A y θ_B no se utilizan ya que no dependen de ninguna variable geométrica y se ha considerado que el error producido por las articulaciones es despreciable. La ecuación resultante es la siguiente:

$$\Delta \mathbf{p} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \frac{\partial f}{\partial(t, h)} \begin{bmatrix} \Delta t \\ \Delta h \end{bmatrix} + \frac{\partial f}{\partial \mathbf{q}_{int}} \begin{bmatrix} \Delta \varphi_{1A} \\ \Delta y_{1A} \\ \Delta \varphi_{2A} \\ \Delta y_{2A} \\ \Delta \varphi_{2B} \\ \Delta y_{2B} \\ \Delta \varphi_{1B} \\ \Delta y_{1B} \end{bmatrix} \quad (92)$$

El siguiente paso será expresar la ecuación únicamente en función de variables geométricas, por lo que habrá que transformar el vector formado por las 8 variables intermedias para que sea compatible con la ecuación (90):

$$\begin{bmatrix} \Delta \varphi_{1A} \\ \Delta y_{1A} \\ \Delta \varphi_{2A} \\ \Delta y_{2A} \\ \Delta \varphi_{2B} \\ \Delta y_{2B} \\ \Delta \varphi_{1B} \\ \Delta y_{1B} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{1A} \\ \mathbf{B}_{2A} \\ \mathbf{B}_{2B} \\ \mathbf{B}_{1B} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} \quad (93)$$

donde \mathbf{B}_{ij} son matrices 2×2 que relacionan los errores en b y p con los errores

producidos en las variables articulares intermedias φ_{ij} y y_{ij} en cada módulo paralelo de la forma:

$$\begin{bmatrix} \Delta\varphi_{ij} \\ \Delta y_{ij} \end{bmatrix} = \mathbf{B}_{ij} \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} \quad (94)$$

El cálculo de \mathbf{B}_{ij} parte de las ecuaciones que definen la cinemática inversa de los módulos paralelos (ecuaciones (16) y (17)). Reordenándolas se obtiene:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} (p \cos \varphi_{ij} - b)^2 + (y_{ij} - p \sin \varphi_{ij})^2 - u_{ij}^2 \\ (p \cos \varphi_{ij} - b)^2 + (y_{ij} + p \sin \varphi_{ij})^2 - v_{ij}^2 \end{bmatrix} = F(y_{ij}, \varphi_{ij}, b, p) \quad (95)$$

Realizando una aproximación lineal:

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = \frac{\partial F}{\partial(b, p)} \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} + \frac{\partial F}{\partial(\varphi_{ij}, y_{ij})} \begin{bmatrix} \Delta\varphi_{ij} \\ \Delta y_{ij} \end{bmatrix} \quad (96)$$

Reordenando la ecuación:

$$\begin{bmatrix} \Delta\varphi_{ij} \\ \Delta y_{ij} \end{bmatrix} = - \left(\frac{\partial F}{\partial(\varphi_{ij}, y_{ij})} \right)^{-1} \frac{\partial F}{\partial(b, p)} \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} \quad (97)$$

Igualando con (94):

$$\mathbf{B}_{ij} = - \left(\frac{\partial F}{\partial(\varphi_{ij}, y_{ij})} \right)^{-1} \frac{\partial F}{\partial(b, p)} \quad (98)$$

Se puede obtener la ecuación definitiva que calcula la diferencia de posición en función de las variables geométricas del robot sustituyendo (98) en (93) y sustituyendo a su vez (93) en (92):

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \frac{\partial f}{\partial(t, h)} \begin{bmatrix} \Delta t \\ \Delta h \end{bmatrix} + \frac{\partial f}{\partial \mathbf{q}_{int}} \begin{bmatrix} - \left(\frac{\partial F}{\partial(\varphi_{1A}, y_{1A})} \right)^{-1} \frac{\partial F}{\partial(b, p)} \\ - \left(\frac{\partial F}{\partial(\varphi_{2A}, y_{2A})} \right)^{-1} \frac{\partial F}{\partial(b, p)} \\ - \left(\frac{\partial F}{\partial(\varphi_{2B}, y_{2B})} \right)^{-1} \frac{\partial F}{\partial(b, p)} \\ - \left(\frac{\partial F}{\partial(\varphi_{1B}, y_{1B})} \right)^{-1} \frac{\partial F}{\partial(b, p)} \end{bmatrix} \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} \quad (99)$$

Para facilitar la explicación y legibilidad, se definirá una nueva matriz \mathbf{B}_t igual a la matriz formada por la concatenación en columna de las 4 matrices \mathbf{B}_{ij} :

$$\mathbf{B}_t = \begin{bmatrix} \mathbf{B}_{1A} \\ \mathbf{B}_{2A} \\ \mathbf{B}_{2B} \\ \mathbf{B}_{1B} \end{bmatrix} = \begin{bmatrix} -\left(\frac{\partial F}{\partial(\varphi_{1A}, y_{1A})}\right)^{-1} \frac{\partial F}{\partial(b, p)} \\ -\left(\frac{\partial F}{\partial(\varphi_{2A}, y_{2A})}\right)^{-1} \frac{\partial F}{\partial(b, p)} \\ -\left(\frac{\partial F}{\partial(\varphi_{2B}, y_{2B})}\right)^{-1} \frac{\partial F}{\partial(b, p)} \\ -\left(\frac{\partial F}{\partial(\varphi_{1B}, y_{1B})}\right)^{-1} \frac{\partial F}{\partial(b, p)} \end{bmatrix} \quad (100)$$

De esta forma quedan definidas las 3 primeras filas de la matriz \mathbf{J}_g :

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial(t, h)} & \frac{\partial f}{\partial \mathbf{q}_{int}} \mathbf{B}_t \end{bmatrix} \begin{bmatrix} \Delta h \\ \Delta t \\ \Delta b \\ \Delta p \end{bmatrix} \quad (101)$$

Al igual que con la posición, las 3 últimas filas de la matriz \mathbf{J}_g relacionan la diferencia en la orientación con los errores en las variables geométricas del robot. Partiendo del modelo cinemático explicado en 2.1.2, la matriz de rotación \mathbf{R} que define la orientación viene dada por la ecuación (24):

$$\mathbf{R} = \begin{bmatrix} \sin \Phi_A \sin \Phi_B + \cos \Theta \cos \Phi_A \cos \Phi_B & \sin \Phi_A \cos \Phi_B - \cos \Theta \cos \Phi_A \sin \Phi_B & \sin \Theta \cos \Phi_A \\ \cos \Phi_A \sin \Phi_B - \cos \Theta \sin \Phi_A \cos \Phi_B & \cos \Phi_A \cos \Phi_B + \cos \Theta \sin \Phi_A \sin \Phi_B & -\sin \Theta \sin \Phi_A \\ -\sin \Theta \cos \Phi_B & \sin \Theta \sin \Phi_B & \cos \Theta \end{bmatrix}$$

donde $\Phi_j = \varphi_{1j} - \varphi_{2j}$ y $\Theta = \theta_A - \theta_B$.

Para relacionar esta matriz con la que define el giro en α , β y γ (77), habrá que extraer las velocidades angulares en cada eje y relacionarlas de esa forma. Las velocidades angulares se pueden extraer mediante la siguiente propiedad:

$$\begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} = \frac{d\mathbf{R}}{dt} \mathbf{R}^T \quad (102)$$

El producto de la derivada de la matriz de rotación respecto al tiempo y la matriz de rotación traspuesta da como resultado una matriz antisimétrica donde se pueden identificar los valores de las velocidades angulares en cada eje.

En la ecuación (24) se puede ver que la orientación del efecto final del robot depende únicamente de las variables θ_j y φ_{ij} :

$$\mathbf{R} = f(\varphi_{1A}, \varphi_{2A}, \theta_A, \theta_B, \varphi_{2B}, \varphi_{1B}) \quad (103)$$

Se puede expresar la derivada de la matriz de rotación respecto al tiempo como la suma de las derivadas parciales respecto a cada variable por su derivada respecto al tiempo. En el caso de la matriz \mathbf{R} :

$$\frac{d\mathbf{R}}{dt} = \frac{\partial \mathbf{R}}{\partial \varphi_{1A}} \dot{\varphi}_{1A} + \frac{\partial \mathbf{R}}{\partial \varphi_{2A}} \dot{\varphi}_{2A} + \frac{\partial \mathbf{R}}{\partial \varphi_{2B}} \dot{\varphi}_{2B} + \frac{\partial \mathbf{R}}{\partial \varphi_{1B}} \dot{\varphi}_{1B} \quad (104)$$

Una vez más, no se han tenido en cuenta las variables θ_j al suponer que su error es despreciable. Las variables intermedias φ_{ij} , en cambio, sí que se tienen en cuenta ya que dependen de las variables geométricas b y p . Seguidamente, se sustituirán las velocidades por errores pequeños, lo cual es equivalente.

Siendo conocida $\frac{d\mathbf{R}}{dt}$, se puede sustituir (104) en (102) e identificar las velocidades angulares ω_x , ω_y y ω_z en función de $\dot{\varphi}_{1A}$, $\dot{\varphi}_{2A}$, $\dot{\varphi}_{2B}$ y $\dot{\varphi}_{1B}$. Las funciones resultantes se pueden escribir de la forma:

$$\begin{bmatrix} \Delta\omega_x \\ \Delta\omega_y \\ \Delta\omega_z \end{bmatrix} = \mathbf{H}_1 \begin{bmatrix} \Delta\varphi_{1A} \\ \Delta\varphi_{2A} \\ \Delta\varphi_{2B} \\ \Delta\varphi_{1B} \end{bmatrix} \quad (105)$$

donde \mathbf{H}_1 es una matriz 3×4 que contiene los coeficientes de las funciones anteriores.

Para la matriz de rotación definida en (77) se sigue el mismo proceso (desde la ecuación (102) hasta la ecuación (105)), resultando en la ecuación:

$$\begin{bmatrix} \Delta\omega_x \\ \Delta\omega_y \\ \Delta\omega_z \end{bmatrix} = \mathbf{H}_2 \begin{bmatrix} \Delta\alpha \\ \Delta\beta \\ \Delta\gamma \end{bmatrix}, \quad \text{donde } \mathbf{H}_2 = \begin{bmatrix} 1 & 0 & \sin\beta & 0 \\ 0 & \cos\alpha & -\sin\alpha\cos\beta & 0 \\ 0 & \sin\alpha & \cos\alpha\cos\beta & 0 \end{bmatrix} \quad (106)$$

Igualando con (105) y reordenando:

$$\begin{bmatrix} \Delta\alpha \\ \Delta\beta \\ \Delta\gamma \end{bmatrix} = \mathbf{H}_2^{-1} \mathbf{H}_1 \begin{bmatrix} \Delta\varphi_{1A} \\ \Delta\varphi_{2A} \\ \Delta\varphi_{2B} \\ \Delta\varphi_{1B} \end{bmatrix} \quad (107)$$

De la misma forma que se ha hecho con la parte de la posición, habrá que transformar el vector compuesto por las variables $\Delta\varphi_{ij}$ para introducir la ecuación resultante en (90). El método seguido es similar al que se utiliza para obtener (98), aunque, en este caso, al necesitar solo relacionar la variable $\Delta\varphi_{ij}$, se descarta la segunda fila de la matriz \mathbf{B}_{ij} , obteniendo \mathbf{B}'_{ij} , matriz 1×2 que relaciona $\Delta\varphi_{ij}$ con Δb y Δp de la forma:

$$\Delta\varphi_{ij} = \mathbf{B}'_{ij} \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} \quad (108)$$

Al igual que en (100), se puede definir un nuevo vector \mathbf{B}'_t que contiene las 4 matrices \mathbf{B}'_{ij} necesarias para realizar la transformación mencionada:

$$\mathbf{B}'_t = \begin{bmatrix} \mathbf{B}'_{1A} \\ \mathbf{B}'_{2A} \\ \mathbf{B}'_{2B} \\ \mathbf{B}'_{1B} \end{bmatrix} \quad (109)$$

Pudiendo, por lo tanto, reescribir la ecuación (107) como:

$$\begin{bmatrix} \Delta\alpha \\ \Delta\beta \\ \Delta\gamma \end{bmatrix} = \mathbf{H}_2^{-1} \mathbf{H}_1 \mathbf{B}'_t \begin{bmatrix} \Delta b \\ \Delta p \end{bmatrix} \quad (110)$$

De esta forma, queda completamente definida la parte de la orientación y, mediante (101) y (110), se puede construir la matriz \mathbf{J}_g en (90) como:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta\alpha \\ \Delta\beta \\ \Delta\gamma \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial(t,h)} & \frac{\partial f}{\partial \mathbf{q}_{int}} \mathbf{B}_t \\ \mathbf{0}_{3 \times 2} & \mathbf{H}_2^{-1} \mathbf{H}_1 \mathbf{B}'_t \end{bmatrix} \begin{bmatrix} \Delta h \\ \Delta t \\ \Delta b \\ \Delta p \end{bmatrix} \quad (111)$$

4.3. Simulación de la calibración

De la misma forma que con el algoritmo de aproximación expuesto en el capítulo 3, se ha realizado una simulación del método de calibración utilizando MATLAB como entorno de programación:

```

1 close all
2 clear
3 clc
4
5 %errores generalizados
6 delta_t = 0.04;
7 delta_h = 0.02;
8 delta_b = 0;
9 delta_p = 0;
10
11 %variables geometricas
12 t = 0.156;
13 h = 0.16;
```

```

14 b = 0.04;
15 p = 0.04;
16
17 %variables geometricas con errores sumados
18 t_error = t + delta_t;
19 h_error = h + delta_h;
20 b_error = b + delta_b;
21 p_error = p + delta_p;
22
23 %posicion y orientacion a la que se fija la garra
24 posicion = [0.25,0,0]';
25 n = [1 0 0]';
26 o = [0 0 -1]';
27 a = cross(n,o);
28 orientacion = [n,o,a]; % (-1.5708, 0, 0) en angulos euler XYZ
29 T_real = [orientacion, posicion; [0 0 0 1]];
30
31 %variables para almacenar
32 J_t = [];
33 X_ideal_t = [];
34 q_int = zeros(10, 10);
35
36 %generacion de 10 configuraciones articulares (modelo con error)
37 for i = 1:10
38     l = 0.195 + rand*0.05;
39     r = 0.195 + rand*0.05;
40     [y2B, phi2B] = fk_par(l, r, b_error, p_error); %cinematica
41     directa modulos paralelos (se introduce el error en b y p)
42     yB = y2B + 0.195 + rand*0.05 - h_error;
43
44     q(i,:) = ik_hyrecro(T_real, phi2B, yB, 0, t_error); %
45     equivalente serie (se introduce el error en t)
46
47     %variables articulares intermedias (se introduce el error en h)
48     y1A = 0.195 + rand*0.05;
49     y2A = q(i,2) - y1A + h_error;
50     y1B = q(i,7) - y2B + h_error;
51     phi1B = q(i,8);
52     phi1A = q(i,1);
53     phi2A = q(i,3);
54     thetaA = q(i,4);
55     thetaB = q(i,5);
56
57     q_int(i,:) = [phi1A, y1A, phi2A, y2A, thetaA, thetaB, phi2B,
58     y2B, phi1B, y1B];
59 end
60
61 %metodo de calibracion
62 for i = 1:10
63     T_ideal = fk_hyrecro(q_int(i,:), t, h); %cinematica directa del
64     modelo nominal
65
66     %pose ideal siguiendo modelo nominal
67     X_ideal_pos = T_ideal(1:3,4);
68     X_ideal_or = xyz_euler(T_ideal);
69     X_ideal = [X_ideal_pos; X_ideal_or];
70     X_ideal_t = [X_ideal_t; X_ideal];

```

```

67
68    %CONSTRUCCION DE LA JACOBIANA
69    %           |   df_dth          df_dqint*B_t      |
70    % J_g = |           zeros(3,2)   inv(H2)*H1*B_t_prima |
71    %           |
72    %
73    %parte posicion
74    J_g_pos = [df_dth(q_int(i,:)), df_dqint(q_int(i,:), t, h)*B_t(
75    q_int(i,:), b, p)];
76    %parte orientacion
77    J_g_or = [zeros(3,2), inv(H2(X_ideal_or))*H1(q_int(i,:))*B_t_prima(q_int(i,:), b, p)];
78
79    J_g = [J_g_pos; J_g_or];
80    J_g_ampliada = [J_g, -eye(6)];
81    J_t = [J_t; J_g_ampliada];
82
83 end
84
85 delta_g = -inv(J_t.' * J_t) * J_t.' * X_ideal_t
86 % delta_g = [delta_t, delta_h, delta_b, delta_p, x_real, y_real,
87 %             z_real, alpha_real, beta_real, gamma_real'];

```

Al comienzo de la simulación, se declaran las 4 variables geométricas (t , h , b , p) y sus errores (δ_t , δ_h , δ_b , δ_p). Cabe aclarar que los errores no se utilizarán en el método de la calibración, puesto que conocer estos es el objetivo de la calibración. Únicamente se utilizarán para inducir un modelo con error con el que se obtendrán las diferentes configuraciones articulares con las que se realizará el método de calibración en cadena cerrada, emulando los errores que tendrá el real, que serán desconocidos *a priori*.

A continuación, se define la posición y orientación a la que se fija la garra durante la simulación.

Seguidamente, en el bucle que empieza en la línea 37, se generan 10 configuraciones articulares siguiendo el modelo con error con las que posteriormente se realizará la calibración. Cada una de las 10 configuraciones articulares se obtiene de la siguiente forma:

Primero, para inducir el error en las variables b y p , se generan aleatoriamente las longitudes de los actuadores lineales de un módulo paralelo. A partir de las longitudes generadas, se utiliza la cinemática directa de los módulos paralelos, para obtener los valores de las coordenadas articulares intermedias correspondientes a ese módulo paralelo. La cinemática directa de los módulos paralelos está programada en la función `fk_par`, la cual utiliza el método expuesto en la sección 2.1.1. El código de la función se puede consultar en A.2.

A continuación, utilizando el modelo cinemático del robot con error, se introduce el error en la variable t . Para ello, se utiliza la función `ik_hyrecro`, la cual utiliza la cinemática inversa del robot expuesta en la sección 2.1.2, pero induciendo un error en la variable geométrica t . El código de la función se puede consultar en A.3.

Finalmente, en las líneas 46 - 53, a las variables articulares intermedias devueltas por la función `ik_hyrecro` se les induce el error en h .

En el siguiente bucle del código, correspondiente a las líneas 59 - 81, se realiza el método de calibración en cadena cerrada expuesto en la sección 4.2. Para cada una de las configuraciones articulares generadas previamente, se sigue el siguiente método:

Primero, se calcula la pose del efecto final siguiendo la cinemática directa del modelo nominal y utilizando la función `fk_hyrecro` (A.4) De la pose calculada, se extrae el vector de posición y se calculan los ángulos de Euler XYZ correspondientes a la orientación calculada. Los 6 elementos extraídos se concatenan en un vector columna, el cual recoge la posición y orientación que producen todas las configuraciones articulares utilizadas.

A continuación, se construye la Jacobiana de Identificación de la forma que se explica en la sección 4.2.1. De la misma forma que se expone en la sección, se construye la matriz en 2 pasos:

Para empezar, las 3 primeras filas de la matriz, correspondientes a la posición, se construyen utilizando las funciones `df_dth`, `df_dqint` y `B_t`, las cuales se pueden consultar en A.5, A.6 y A.7, respectivamente. Su explicación está desarrollada a lo largo de las ecuaciones (91) y (101).

La construcción de la parte de la orientación está explicada a lo largo de las ecuaciones (102) y (110). En el código se construye utilizando las funciones `H1`, `H2` y `B_t_prima`, las cuales se pueden consultar en A.8, A.9 y A.10, respectivamente.

Finalmente, en la línea 83, se resuelve el sistema de la forma explicada en la ecuación (88). El vector resultante corresponde al de la ecuación (89) y contiene la información de los errores geométricos hallados en el modelo del robot y la pose real no conocida *a priori*.

El resultado que se devuelve para el código mostrado es el siguiente:

```
delta_g = [0.04, 0.02, 0, 0, 0.25, 0, 0, -1.57, 0, 0]',
```

El cual es el resultado esperado y correcto; ya que los 4 primeros elementos corresponden a los errores inducidos en t , h , b y p ; mientras que los 6 últimos corresponden a la posición y orientación en ángulos de Euler XYZ a la que se ha fijado la garra que actúa como efecto final.

Para la implementación del método de calibración expuesto en el prototipo se deberá de programar previamente un esquema de control sobreactuado, el cual permita realizar movimientos en las articulaciones del robot mientras ambas garras están fijadas. Cuando se fijan ambas garras para la calibración, el robot queda sobreactuado, ya que la cadena cinemática cerrada resultante tiene únicamente 4 o 5

grados de libertad en función de si los ejes **z** de los pies sean paralelos o antiparalelos o no. Al tener más actuadores que grados de libertad a efectos prácticos, aparecen tensiones internas, las cuales pueden producir daños en la estructura del robot o en sus actuadores. El esquema de control diseñado deberá facilitar los movimientos descritos minimizando las tensiones internas de la cadena cinemática.

5. Conclusiones y trabajos futuros

El principal objetivo abarcado en este TFG ha sido el de la optimización del robot HyReCRo. A raíz del objetivo principal, se han desarrollado y expuesto 2 propuestas: el control cinemático de la adhesión y la calibración en cadena cerrada del robot. En este capítulo se comentarán las conclusiones extraídas de los dos capítulos de desarrollo y se propondrán futuros trabajos a corto y largo plazo.

5.1. Conclusiones

En el capítulo 3 se ha propuesto e implementado un algoritmo que permite el acercamiento de la garra a la superficie de adhesión. El control de la aproximación se realiza cinemáticamente y de forma realimentada a partir de la matriz de diferencia, la cual expresa la posición y orientación de la garra respecto a la superficie, calculada con los datos obtenidos e interpretados de los 3 sensores de distancia instalados en la garra. Además, se ha programado una herramienta que representa en tiempo real la posición y orientación del efecto final del robot. Finalmente, se ha mostrado un experimento satisfactorio de la implementación del algoritmo en el prototipo del robot HyReCRo.

En el capítulo 4 se ha expuesto un método de calibración con el fin de mejorar la precisión en sus movimientos y agarre. Entre los métodos de calibración existentes, se ha optado por una calibración en cadena cerrada, para evitar costes relacionados con equipo adicional para medir con precisión la pose de la garra libre, y para poder realizar la calibración de manera autónoma. Se ha desarrollado el método elegido para el caso de un robot redundante y se ha expuesto como obtener la matriz Jacobiana de Identificación que permite realizar la calibración. Finalmente, se ha realizado una simulación para comprobar el correcto funcionamiento del método.

En ambos casos, la implementación y simulación de las propuestas desarrolladas en los capítulos 3 y 4 han resultado exitosas, permitiendo mejorar la precisión y automatización de la adhesión de las garras del robot.

5.2. Trabajos futuros

En esta sección final, se propondrán líneas de desarrollo futuras derivadas de este proyecto. Los trabajos futuros se dividirán en propuestas a corto y largo plazo.

A corto plazo, es necesaria la implementación del método de calibración en cadena cerrada expuesto en el prototipo del robot HyReCRo. No sin antes, como se ha comentado en el capítulo anterior, diseñar y programar un esquema de control sobreactuado que permita realizar movimientos en las articulaciones del robot con ambas garras fijas minimizando las tensiones internas producidas. De esta forma, la precisión del robot mejorará y, consecuentemente, también lo hará la precisión

del algoritmo de acercamiento de la garra para la adhesión.

También a corto plazo, es conveniente añadir restricciones al algoritmo que rea-liza el control cinemático de la garra durante la aproximación para evitar colisiones. Como se ha comentado en su correspondiente sección, existen soluciones a las que converge el algoritmo donde se sitúa la garra libre junto a la garra fija, provocan-do una colisión entre extremidades. Añadiendo las restricciones y aprovechando la redundancia del robot, se evitaría la colisión y se obtendría un algoritmo más robusto.

A largo plazo, convendría automatizar completamente los movimientos del ro-bot. Actualmente, el control del movimiento se hace de forma teleoperada. Teniendo en cuenta que la aplicación principal del robot HyReCRo es la de inspección y man-tenimiento de estructuras metálicas tridimensionales, se hace necesario automatizar los movimientos del robot, ya que el acceso de operarios a estas estructuras para el control manual es muy complicado y peligroso. Los movimientos a automatizar son el desplazamiento longitudinal y las transiciones, tanto cóncavas como convexas, entre superficies. Una opción para automatización total de los movimientos podría basarse en visión artificial o medidas láser. En cualquier caso, será necesario resolver problemas fundamentales de robótica móvil, como la localización y el *mapping*.

A. Códigos

A.1. jacob

```
1 void jacob(double q[8], double J[6][8], boolean fijo_A){  
2     double t = 11;  
3  
4     if (fijo_A){  
5         //primera fila  
6         J[0][0] = q[1]*cos(q[0]) - q[6]*(cos(q[0] - q[2])*cos(q[5])  
7         - cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[5])) - t*sin(q[0] -  
8         q[2])*cos(q[3]);  
9         J[0][1] = sin(q[0]);  
10        J[0][2] = q[6]*(cos(q[0] - q[2])*cos(q[5]) - cos(q[3] - q  
11        [4])*sin(q[0] - q[2])*sin(q[5])) + t*sin(q[0] - q[2])*cos(q[3])  
12        ;  
13        J[0][3] = q[6]*cos(q[0] - q[2])*sin(q[3] - q[4])*sin(q[5])  
14        - t*cos(q[0] - q[2])*sin(q[3]);  
15        J[0][4] = -q[6]*cos(q[0] - q[2])*sin(q[3] - q[4])*sin(q[5])  
16        ;  
17        J[0][5] = q[6]*(sin(q[0] - q[2])*sin(q[5]) - cos(q[0] - q  
18        [2])*cos(q[3] - q[4])*cos(q[5]));  
19        J[0][6] = -sin(q[0] - q[2])*cos(q[5]) - cos(q[0] - q[2])*  
20        cos(q[3] - q[4])*sin(q[5]);  
21        J[0][7] = 0;  
22  
23        //segunda fila  
24        J[1][0] = q[6]*(sin(q[0] - q[2])*cos(q[5]) + cos(q[0] - q  
25        [2])*cos(q[3] - q[4])*sin(q[5])) - q[1]*sin(q[0]) + t*sin(q[0]  
26        - q[2])*cos(q[3]);  
27        J[1][1] = cos(q[0]);  
28        J[1][2] = -q[6]*(sin(q[0] - q[2])*cos(q[5]) + cos(q[0] - q  
29        [2])*cos(q[3] - q[4])*sin(q[5])) - t*sin(q[0] - q[2])*cos(q[3])  
30        ;  
31        J[1][3] = t*cos(q[0] - q[2])*sin(q[3]) - q[6]*sin(q[0] - q  
32        [2])*sin(q[3] - q[4])*sin(q[5]);  
33        J[1][4] = q[6]*sin(q[0] - q[2])*sin(q[3] - q[4])*sin(q[5]);  
34        J[1][5] = q[6]*(cos(q[0] - q[2])*sin(q[5]) + cos(q[3] - q  
35        [4])*sin(q[0] - q[2])*cos(q[5]));  
36        J[1][6] = cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[5]) - cos  
(q[0] - q[2])*cos(q[5]);  
37        J[1][7] = 0;  
38  
39        //tercera fila  
40        J[2][0] = 0;  
41        J[2][1] = 0;  
42        J[2][2] = 0;  
43        J[2][3] = q[6]*cos(q[3] - q[4])*sin(q[5]) - t*cos(q[3]);  
44        J[2][4] = -q[6]*cos(q[3] - q[4])*sin(q[5]);  
45        J[2][5] = q[6]*sin(q[3] - q[4])*cos(q[5]);  
46        J[2][6] = sin(q[3] - q[4])*sin(q[5]);  
47        J[2][7] = 0;  
48  
49        //cuarta fila  
50        J[3][0] = 0;
```

```

37     J[3][1] = 0;
38     J[3][2] = 0;
39     J[3][3] = sin(q[0] - q[2]);
40     J[3][4] = -sin(q[0] - q[2]);
41     J[3][5] = -cos(q[0] - q[2])*sin(q[3] - q[4]);
42     J[3][6] = 0;
43     J[3][7] = cos(q[0] - q[2])*sin(q[3] - q[4]);
44
45 //quinta fila
46 J[4][0] = 0;
47 J[4][1] = 0;
48 J[4][2] = 0;
49 J[4][3] = cos(q[0] - q[2]);
50 J[4][4] = -cos(q[0] - q[2]);
51 J[4][5] = sin(q[0] - q[2])*sin(q[3] - q[4]);
52 J[4][6] = 0;
53 J[4][7] = -sin(q[0] - q[2])*sin(q[3] - q[4]);
54
55 //sexta fila
56 J[5][0] = -1;
57 J[5][1] = 0;
58 J[5][2] = 1;
59 J[5][3] = 0;
60 J[5][4] = 0;
61 J[5][5] = -cos(q[3] - q[4]);
62 J[5][6] = 0;
63 J[5][7] = cos(q[3] - q[4]);
64 } else {
65 //primera fila
66     J[0][0] = t*cos(q[3])*(sin(q[7] - q[5]) + cos(q[7] - q[5])*cos(q[3] - q[4]) - 2*pow(cos(q[0] - q[2]),2)*sin(q[7] - q[5]) - 2*cos(q[0] - q[2])*sin(q[0] - q[2])*sin(q[7] - q[5]) - 2*pow(cos(q[0] - q[2]),2)*cos(q[7] - q[5])*cos(q[3] - q[4]) + 2*cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q[0] - q[2]));
67     J[0][1] = -sin(q[7] - q[5])*cos(q[2]) - cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q[2]);
68     J[0][2] = t*pow(cos(q[0] - q[2]),2)*sin(q[7] - q[5])*cos(q[3] - t*pow(sin(q[0] - q[2]),2)*sin(q[7] - q[5])*cos(q[3]) + q[1]*cos(q[0] - q[2])*sin(q[7] - q[5])*sin(q[0]) - q[1]*sin(q[0] - q[2])*sin(q[7] - q[5])*cos(q[0]) + t*pow(cos(q[0] - q[2]),2)*cos(q[7] - q[5])*cos(q[3] - q[4])*cos(q[3]) - t*cos(q[7] - q[5])*cos(q[3] - q[4])*pow(sin(q[0] - q[2]),2)*cos(q[3]) - q[1]*cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q[3] - q[4])*cos(q[0]) + 2*t*cos(q[0] - q[2])*sin(q[0] - q[2])*sin(q[7] - q[5])*cos(q[3] - q[4]) - q[1]*cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[0]) - 2*t*cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[0]) - 2*t*cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q[0] - q[2])*cos(q[3]));
69     J[0][3] = (sin(q[0] - q[2])*sin(q[7] - q[5]) + cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q[3] - q[4]))*(t*cos(q[0] - q[2])*sin(q[3]) - q[6]*cos(q[0] - q[2])*sin(q[3] - q[4])*sin(q[5])) - (t*cos(q[0] - q[2])*sin(q[3]) - q[6]*sin(q[0] - q[2])*sin(q[3] - q[4])*sin(q[5]))*(cos(q[0] - q[2])*sin(q[7] - q[5]) - cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q[0] - q[2])) - cos(q[7] - q[5])*sin(q[3] - q[4])*(t*cos(q[3]) - q[6]*cos(q[3] - q[4])*sin(q[5])) - cos(q[7] - q[5])*cos(q[3] - q[4])*(t*sin(q[3]) - q[6]*sin(q[3] - q[4])*sin(q[5])) + cos(q[0] - q[2])*cos(q[7] - q[5])*sin(q[3] - q[4])*sin(q[5])) + cos(q[0] - q[2])*cos(q[7] - q[5])*sin(q[3] - q[4])*cos(q[5]);

```

```

q[3] - q[4])*(q[1]*sin(q[0]) - q[6]*(sin(q[0] - q[2])*cos(q[5])
+ cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q[5])) + t*cos(q[0] -
q[2])*cos(q[3])) + cos(q[7] - q[5])*sin(q[0] - q[2])*sin(q[3] -
q[4])*(q[6]*(cos(q[0] - q[2])*cos(q[5]) - cos(q[3] - q[4])*sin(
q[0] - q[2])*sin(q[5])) - q[1]*cos(q[0]) + t*cos(q[0] - q[2])*cos(
q[3]));
70   J[0][4] = cos(q[7] - q[5))*(t*cos(q[3] - q[4])*sin(q[3]) -
t*pow(cos(q[0] - q[2]), 2)*sin(q[3] - q[4])*cos(q[3]) - q[1]*cos(
q[0] - q[2])*sin(q[3] - q[4])*sin(q[0]) + q[1]*sin(q[0] - q
[2])*sin(q[3] - q[4])*cos(q[0]) - 2*q[6]*cos(q[3] - q[4])*sin(q
[3] - q[4])*sin(q[5]) + 2*q[6]*pow(cos(q[0] - q[2]), 2)*cos(q[3]
- q[4])*sin(q[3] - q[4])*sin(q[5]) + 2*q[6]*cos(q[3] - q[4])*sin(
q[0] - q[2]), 2)*sin(q[3] - q[4])*sin(q[5]) - t*cos(q[0] -
q[2])*sin(q[0] - q[2])*sin(q[3] - q[4])*cos(q[3]));
71   J[0][5] = (cos(q[7] - q[5])*sin(q[0] - q[2]) - cos(q[0] - q
[2])*cos(q[3] - q[4])*sin(q[7] - q[5]))*(q[1]*sin(q[0]) - q
[6]*(sin(q[0] - q[2])*cos(q[5]) + cos(q[0] - q[2])*cos(q[3] - q
[4])*sin(q[5])) + t*cos(q[0] - q[2])*cos(q[3])) - (cos(q[0] - q
[2])*cos(q[7] - q[5]) + cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q
[7] - q[5]))*(q[6]*(cos(q[0] - q[2])*cos(q[5]) - cos(q[3] - q
[4])*sin(q[0] - q[2])*sin(q[5])) - q[1]*cos(q[0]) + t*cos(q[0]
- q[2])*cos(q[3])) - q[6]*(sin(q[0] - q[2])*sin(q[7] - q[5]) +
cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q[3] - q[4]))*(sin(q[0] -
q[2])*sin(q[5]) - cos(q[0] - q[2])*cos(q[3] - q[4])*cos(q[5]))
- q[6]*(cos(q[0] - q[2])*sin(q[7] - q[5]) - cos(q[7] - q[5])*cos(
q[3] - q[4])*sin(q[0] - q[2]))*(cos(q[0] - q[2])*sin(q[5]) +
cos(q[3] - q[4])*sin(q[0] - q[2])*cos(q[5])) - sin(q[7] - q
[5])*sin(q[3] - q[4])*(t*sin(q[3]) - q[6]*sin(q[3] - q[4])*sin(
q[5])) + q[6]*cos(q[7] - q[5])*pow(sin(q[3] - q[4]), 2)*cos(q
[5]);
72   J[0][6] = sin(q[7]);
73   J[0][7] = (cos(q[0] - q[2])*cos(q[7] - q[5]) + cos(q[3] - q
[4])*sin(q[0] - q[2])*sin(q[7] - q[5]))*(q[6]*(cos(q[0] - q[2])
*cos(q[5]) - cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[5])) - q
[1]*cos(q[0]) + t*cos(q[0] - q[2])*cos(q[3])) - (cos(q[7] - q
[5])*sin(q[0] - q[2]) - cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q
[7] - q[5]))*(q[1]*sin(q[0]) - q[6]*(sin(q[0] - q[2])*cos(q[5])
+ cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q[5])) + t*cos(q[0] -
q[2])*cos(q[3])) + sin(q[7] - q[5])*sin(q[3] - q[4])*(t*sin(q
[3]) - q[6]*sin(q[3] - q[4])*sin(q[5]));
74
75   //segunda fila
76   J[1][0] = -t*cos(q[3))*(cos(q[3] - q[4])*sin(q[7] - q[5]) -
cos(q[7] - q[5]) + 2*pow(cos(q[0] - q[2]), 2)*cos(q[7] - q[5])
+ 2*cos(q[0] - q[2])*cos(q[7] - q[5])*sin(q[0] - q[2]) - 2*pow(
cos(q[0] - q[2]), 2)*cos(q[3] - q[4])*sin(q[7] - q[5]) + 2*cos(q
[0] - q[2])*cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[7] - q[5]))
;
77   J[1][1] = cos(q[3] - q[4])*sin(q[7] - q[5])*sin(q[2]) - cos(
q[7] - q[5])*cos(q[2]);
78   J[1][2] = t*pow(cos(q[0] - q[2]), 2)*cos(q[7] - q[5])*cos(q
[3]) - t*cos(q[7] - q[5])*pow(sin(q[0] - q[2]), 2)*cos(q[3]) + q
[1]*cos(q[0] - q[2])*cos(q[7] - q[5])*sin(q[0]) - q[1]*cos(q[7]
- q[5])*sin(q[0] - q[2])*cos(q[0]) - t*pow(cos(q[0] - q[2]), 2)
*cos(q[3] - q[4])*sin(q[7] - q[5])*cos(q[3]) + t*cos(q[3] - q
[4])*pow(sin(q[0] - q[2]), 2)*sin(q[7] - q[5])*cos(q[3]) + 2*t*
cos(q[0] - q[2])*cos(q[7] - q[5])*sin(q[0] - q[2])*cos(q[3]) +

```

```

q[1]*cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q[7] - q[5])*cos(q
[0]) + q[1]*cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[7] - q[5])*
sin(q[0]) + 2*t*cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q[0] - q
[2])*sin(q[7] - q[5])*cos(q[3]);
79
J[1][3] = sin(q[7] - q[5])*sin(q[3] - q[4])*(t*cos(q[3]) -
q[6]*cos(q[3] - q[4])*sin(q[5])) - (t*cos(q[0] - q[2])*sin(q
[3]) - q[6]*sin(q[0] - q[2])*sin(q[3] - q[4])*sin(q[5]))*(cos(q
[0] - q[2])*cos(q[7] - q[5]) + cos(q[3] - q[4])*sin(q[0] - q
[2])*sin(q[7] - q[5])) + cos(q[3] - q[4])*sin(q[7] - q[5])*(t*
sin(q[3]) - q[6]*sin(q[3] - q[4])*sin(q[5])) + cos(q[0] - q[2])
*(cos(q[7] - q[5])*sin(q[0] - q[2]) - cos(q[0] - q[2])*cos(q[3]
- q[4])*sin(q[7] - q[5]))*(t*sin(q[3]) - q[6]*sin(q[3] - q[4])
*sin(q[5])) - cos(q[0] - q[2])*sin(q[7] - q[5])*sin(q[3] - q
[4])*(q[1]*sin(q[0]) + t*cos(q[0] - q[2])*cos(q[3]) - q[6]*sin(
q[0] - q[2])*cos(q[5]) - q[6]*cos(q[0] - q[2])*cos(q[3] - q[4])
*sin(q[5])) + sin(q[0] - q[2])*sin(q[7] - q[5])*sin(q[3] - q
[4])*(q[1]*cos(q[0]) - t*cos(q[0] - q[2])*cos(q[3]) - q[6]*cos(
q[0] - q[2])*cos(q[5]) + q[6]*cos(q[3] - q[4])*sin(q[0] - q[2])
*sin(q[5]));
80
J[1][4] = -sin(q[7] - q[5])*(t*cos(q[3] - q[4])*sin(q[3]) -
t*pow(cos(q[0] - q[2]), 2)*sin(q[3] - q[4])*cos(q[3]) - q[1]*
cos(q[0] - q[2])*sin(q[3] - q[4])*sin(q[0]) + q[1]*sin(q[0] - q
[2])*sin(q[3] - q[4])*cos(q[0]) - 2*q[6]*cos(q[3] - q[4])*sin(q
[3] - q[4])*sin(q[5]) + 2*q[6]*pow(cos(q[0] - q[2]), 2)*cos(q[3]
- q[4])*sin(q[3] - q[4])*sin(q[5]) + 2*q[6]*cos(q[3] - q[4])*
pow(sin(q[0] - q[2]), 2)*sin(q[3] - q[4])*sin(q[5]) - t*cos(q[0]
- q[2])*sin(q[0] - q[2])*sin(q[3] - q[4])*cos(q[3]));
81
J[1][5] = (cos(q[0] - q[2])*sin(q[7] - q[5]) - cos(q[7] - q
[5])*cos(q[3] - q[4])*sin(q[0] - q[2]))*(q[6]*(cos(q[0] - q[2])
*cos(q[5]) - cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[5])) - q
[1]*cos(q[0]) + t*cos(q[0] - q[2])*cos(q[3])) - (sin(q[0] - q
[2])*sin(q[7] - q[5]) + cos(q[0] - q[2])*cos(q[7] - q[5])*cos(q
[3] - q[4]))*(q[1]*sin(q[0]) - q[6]*(sin(q[0] - q[2])*cos(q[5])
+ cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q[5])) + t*cos(q[0] -
q[2])*cos(q[3])) - q[6]*(cos(q[0] - q[2])*cos(q[7] - q[5]) +
cos(q[3] - q[4])*sin(q[0] - q[2])*sin(q[7] - q[5]))*(cos(q[0] -
q[2])*sin(q[5]) + cos(q[3] - q[4])*sin(q[0] - q[2])*cos(q[5]))
- q[6]*(cos(q[7] - q[5])*sin(q[0] - q[2]) - cos(q[0] - q[2])*cos(q
[3] - q[4])*sin(q[7] - q[5]))*(sin(q[0] - q[2])*sin(q[5]) -
cos(q[0] - q[2])*cos(q[3] - q[4])*cos(q[5])) - cos(q[7] - q
[5])*sin(q[3] - q[4])*(t*sin(q[3]) - q[6]*sin(q[3] - q[4])*sin(q
[5])) - q[6]*sin(q[7] - q[5])*pow(sin(q[3] - q[4]), 2)*cos(q
[5]));
82
J[1][6] = cos(q[7]);
83
J[1][7] = (sin(q[0] - q[2])*sin(q[7] - q[5]) + cos(q[0] - q
[2])*cos(q[7] - q[5])*cos(q[3] - q[4]))*(q[1]*sin(q[0]) - q
[6]*(sin(q[0] - q[2])*cos(q[5]) + cos(q[0] - q[2])*cos(q[3] - q
[4])*sin(q[5])) + t*cos(q[0] - q[2])*cos(q[3])) - (cos(q[0] - q
[2])*sin(q[7] - q[5]) - cos(q[7] - q[5])*cos(q[3] - q[4])*sin(q
[0] - q[2]))*(q[6]*(cos(q[0] - q[2])*cos(q[5]) - cos(q[3] - q
[4])*sin(q[0] - q[2])*sin(q[5])) - q[1]*cos(q[0]) + t*cos(q[0]
- q[2])*cos(q[3])) + cos(q[7] - q[5])*sin(q[3] - q[4])*(t*sin(q
[3]) - q[6]*sin(q[3] - q[4])*sin(q[5]));
84
85 //tercera fila
86 J[2][0] = -(pow(2, 1/2)*t*(cos(2*q[2] - 2*q[0] - 2*q[3] + q
[4] + PI/4) + cos(2*q[2] - 2*q[0] + q[4] + PI/4) - sin(2*q[0] -

```

```

    2*q[2] + q[4] + PI/4) - cos(2*q[2] - 2*q[0] + 2*q[3] - q[4] +
PI/4)))/4;
87     J[2][1] = -sin(q[3] - q[4])*sin(q[2]);
88     J[2][2] = -sin(q[3] - q[4])*(t*pow(sin(q[0] - q[2]), 2)*cos(
q[3]) - t*pow(cos(q[0] - q[2]), 2)*cos(q[3]) + q[1]*cos(q[0] - q
[2])*cos(q[0]) + q[1]*sin(q[0] - q[2])*sin(q[0]) + 2*t*cos(q[0]
- q[2])*sin(q[0] - q[2])*cos(q[3]));
89     J[2][3] = (t*cos(2*q[3] - q[4]))/2 - (q[1]*sin(q[2] + q[3]
- q[4]))/2 - (q[1]*sin(q[2] - q[3] + q[4]))/2 - (t*cos(2*q[0] -
2*q[2] - 2*q[3] + q[4]))/4 - (t*sin(2*q[0] - 2*q[2] - 2*q[3] +
q[4]))/4 - (t*cos(2*q[0] - 2*q[2] + 2*q[3] - q[4]))/4 - (t*sin(
2*q[0] - 2*q[2] + 2*q[3] - q[4]))/4;
90     J[2][4] = -q[6]*sin(q[5])*pow(cos(q[0] - q[2]), 2)*pow(cos(
q[3] - q[4]), 2) + t*cos(q[3])*pow(cos(q[0] - q[2]), 2)*cos(q[3]
- q[4]) + q[6]*sin(q[5])*pow(cos(q[0] - q[2]), 2)*pow(sin(q[3] -
q[4]), 2) + t*cos(q[3])*cos(q[0] - q[2])*cos(q[3] - q[4])*sin(q
[0] - q[2]) + q[1]*sin(q[0])*cos(q[0] - q[2])*cos(q[3] - q[4])
- q[6]*sin(q[5])*pow(cos(q[3] - q[4]), 2)*pow(sin(q[0] - q[2]),
2) + q[6]*sin(q[5])*pow(cos(q[3] - q[4]), 2) - q[1]*cos(q[0])*cos(
q[3] - q[4])*sin(q[0] - q[2]) + q[6]*sin(q[5])*pow(sin(q[0] -
q[2]), 2)*pow(sin(q[3] - q[4]), 2) - q[6]*sin(q[5])*pow(sin(q
[3] - q[4]), 2) + t*sin(q[3])*sin(q[3] - q[4]);
91     J[2][5] = 0;
92     J[2][6] = 0;
93     J[2][7] = 0;
94
95     //cuarta fila
96     J[3][0] = -cos(q[7] - q[5])*sin(q[3] - q[4]);
97     J[3][1] = 0;
98     J[3][2] = cos(q[7] - q[5])*sin(q[3] - q[4]);
99     J[3][3] = -sin(q[7] - q[5]);
100    J[3][4] = sin(q[7] - q[5]);
101    J[3][5] = 0;
102    J[3][6] = 0;
103    J[3][7] = 0;
104
105    //quinta fila
106    J[4][0] = sin(q[7] - q[5])*sin(q[3] - q[4]);
107    J[4][1] = 0;
108    J[4][2] = -sin(q[7] - q[5])*sin(q[3] - q[4]);
109    J[4][3] = -cos(q[7] - q[5]);
110    J[4][4] = cos(q[7] - q[5]);
111    J[4][5] = 0;
112    J[4][6] = 0;
113    J[4][7] = 0;
114
115    //sexta fila
116    J[5][0] = cos(q[3] - q[4]);
117    J[5][1] = 0;
118    J[5][2] = -cos(q[3] - q[4]);
119    J[5][3] = 0;
120    J[5][4] = 0;
121    J[5][5] = 1;
122    J[5][6] = 0;
123    J[5][7] = -1;
124 }
125 }
```

A.2. fk_par

```
1 function [y, phi] = fk_par(l, r, b, p)
2 k2 = 2*b^2 + 2*p^2 - l^2 - r^2;
3 k1 = ((b + p)^2 - (l^2 + r^2)/2)*((b - p)^2 - (l^2 + r^2)/2);
4 k0 = b^2*(l + r)^2*(l - r)^2/4;
5 raices = roots([1 k2 k1 k0]);
6
7 epsilon = max(raices(real(raices)>0&imag(raices)==0));
8 y = sqrt(epsilon);
9
10 cos_phi = (2*y^2 + 2*b^2 + 2*p^2 - l^2 - r^2)/(4*b*p);
11 sin_phi = (r^2 - l^2)/(4*y*p);
12 phi = atan2(sin_phi, cos_phi);
13 end
```

A.3. ik_hyrecro

```

1 function [q] = ik_hyrecro(T, phi_2B, y_B, phi_1B, t)
2 % Esta funcion solo da una solucion de las 2 o 4 ramas
3 % phi_1B solo es necesario como argumento de entrada cuando los
4 % ejes z de los pies son paralelos o antiparalelos
5 if T(3,3) == 1 || T(3,3) == -1 % ejes z de los pies son paralelos
6 % o antiparalelos
7 % Matriz de rotacion
8 % disp('ejes z de los pies son paralelos o antiparalelos')
9 Theta = acos(T(3,3));
10 Phi_A = atan2(T(1,2), T(2,2)) + T(3,3)*(phi_1B - phi_2B);
11
12 % Vector de posicion
13 theta_A = asin((y_B*sin(Theta)*sin(phi_2B) - T(3,4))/ t);
14 theta_B = theta_A - Theta;
15 K1 = T(1,4) + y_B*cos(Theta)*cos(Phi_A)*sin(phi_2B) + y_B*sin(
16 Phi_A)*cos(phi_2B) - t*cos(theta_A)*cos(Phi_A);
17 K2 = T(2,4) - y_B*cos(Theta)*sin(Phi_A)*sin(phi_2B) + y_B*cos(
18 Phi_A)*cos(phi_2B) + t*cos(theta_A)*sin(Phi_A);
19 y_A = sqrt(K1^2 + K2^2);
20 phi_1A = atan2(K1, K2);
21
22 phi_2A = phi_1A - Phi_A;
23
24 else
25 % Matriz de rotacion
26 % disp('ejes z de los pies NO son paralelos o antiparalelos')
27 Theta = acos(T(3,3));
28
29 Phi_A = atan2(-T(2,3)/sin(Theta), T(1,3)/sin(Theta));
30 Phi_B = atan2(T(3,2)/sin(Theta), -T(3,1)/sin(Theta));
31
32 % Vector de posicion
33 theta_A = asin((y_B*sin(Theta)*sin(phi_2B) - T(3,4))/ t);
34 theta_B = theta_A - Theta;
35 K1 = T(1,4) + y_B*(cos(Theta)*cos(Phi_A)*sin(phi_2B) + sin(
36 Phi_A)*cos(phi_2B)) - t*cos(theta_A)*cos(Phi_A);
37 K2 = T(2,4) + y_B*(cos(Phi_A)*cos(phi_2B) - cos(Theta)*sin(
38 Phi_A)*sin(phi_2B)) + t*cos(theta_A)*sin(Phi_A);
39 y_A = sqrt(K1^2 + K2^2);
40 phi_1A = atan2(K1, K2);
41
42 phi_2A = phi_1A - Phi_A;
43 phi_1B = Phi_B + phi_2B;
44 end
45
46 q = [phi_1A, y_A, phi_2A, theta_A, theta_B, phi_2B, y_B, phi_1B];
47 end

```

A.4. fk_hyrecro

```

1 function [T] = fk_hyrecro(q, t, h)
2
3 phi_1A = q(1);
4 y_1A = q(2);
5 phi_2A = q(3);
6 y_2A = q(4);
7 theta_A = q(5);
8 theta_B = q(6);
9 phi_2B = q(7);
10 y_2B = q(8);
11 phi_1B = q(9);
12 y_1B = q(10);
13
14 y_A = y_1A + y_2A - h;
15 y_B = y_1B + y_2B - h;
16 Theta = theta_A - theta_B;
17 Phi_A = phi_1A - phi_2A;
18 Phi_B = phi_1B - phi_2B;
19
20
21 m1 = [sin(phi_1A); cos(phi_1A); 0];
22 m2 = [-cos(Theta)*cos(Phi_A)*sin(phi_2B) - sin(Phi_A)*cos(phi_2B);
23   cos(Theta)*sin(Phi_A)*sin(phi_2B) - cos(Phi_A)*cos(phi_2B);
24   sin(Theta)*sin(phi_2B)];
25 m3 = [cos(theta_A)*cos(Phi_A); -cos(theta_A)*sin(Phi_A); -sin(
26   theta_A)];
27 p = y_A*m1 + y_B*m2 + t*m3;
28
29 n = [sin(Phi_A)*sin(Phi_B) + cos(Theta)*cos(Phi_A)*cos(Phi_B);
30   cos(Phi_A)*sin(Phi_B) - cos(Theta)*sin(Phi_A)*cos(Phi_B);
31   -sin(Theta)*cos(Phi_B)];
32
33 o = [sin(Phi_A)*cos(Phi_B) - cos(Theta)*cos(Phi_A)*sin(Phi_B);
34   cos(Phi_A)*cos(Phi_B) + cos(Theta)*sin(Phi_A)*sin(Phi_B);
35   sin(Theta)*sin(Phi_B)];
36
37 a = cross(n,o);
38 orientacion = [n,o,a];
39 T = [orientacion,p;[0 0 0 1]];
40 end

```

A.5. df_dth

```
1 function J = df_dth(q)
2 phi_1A = q(1);
3 %y_1A = q(2);
4 phi_2A = q(3);
5 %y_2A = q(4);
6 theta_A = q(5);
7 theta_B = q(6);
8 phi_2B = q(7);
9 %y_2B = q(8);
10 %phi_1B = q(9);
11 %y_1B = q(10);
12
13 c1 = [cos(phi_1A - phi_2A)*cos(theta_A); -sin(phi_1A - phi_2A)*cos(
14     theta_A); -sin(theta_A)];
15 c2 = [sin(phi_1A - phi_2A)*cos(phi_2B) - sin(phi_1A) + cos(phi_1A -
16     phi_2A)*cos(theta_A - theta_B)*sin(phi_2B);
17     cos(phi_1A - phi_2A)*cos(phi_2B) - cos(phi_1A) - cos(theta_A -
18     theta_B)*sin(phi_1A - phi_2A)*sin(phi_2B);
19     -sin(theta_A - theta_B)*sin(phi_2B)];
20 J = [c1, c2];
21 end
```

A.6. df_dqint

```

1 function J = df_dqint(q, t, h)
2 phi_1A = q(1);
3 y_1A = q(2);
4 phi_2A = q(3);
5 y_2A = q(4);
6 theta_A = q(5);
7 theta_B = q(6);
8 phi_2B = q(7);
9 y_2B = q(8);
10 %phi_1B = q(9);
11 y_1B = q(10);
12
13 f1 = [cos(phi_1A)*(y_1A - h + y_2A) - (cos(phi_1A - phi_2A)*cos(
    phi_2B) - cos(theta_A - theta_B)*sin(phi_1A - phi_2A)*sin(
    phi_2B))*(y_1B - h + y_2B) - t*sin(phi_1A - phi_2A)*cos(theta_A),
    sin(phi_1A), (cos(phi_1A - phi_2A)*cos(phi_2B) - cos(theta_A -
    theta_B)*sin(phi_1A - phi_2A)*sin(phi_2B))*(y_1B - h + y_2B)
    + t*sin(phi_1A - phi_2A)*cos(theta_A), sin(phi_1A), (sin(
    phi_1A - phi_2A)*sin(phi_2B) - cos(phi_1A - phi_2A)*cos(theta_A -
    theta_B)*cos(phi_2B))*(y_1B - h + y_2B), - sin(phi_1A -
    phi_2A)*cos(phi_2B) - cos(phi_1A - phi_2A)*cos(theta_A -
    theta_B)*sin(phi_2B), 0, - sin(phi_1A - phi_2A)*cos(phi_2B) -
    cos(phi_1A - phi_2A)*cos(theta_A - theta_B)*sin(phi_2B)];
14 f2 = [cos(phi_1A)*(y_1A - h + y_2A) - (cos(phi_1A - phi_2A)*cos(
    phi_2B) - cos(theta_A - theta_B)*sin(phi_1A - phi_2A)*sin(
    phi_2B))*(y_1B - h + y_2B) - t*sin(phi_1A - phi_2A)*cos(theta_A),
    sin(phi_1A), (cos(phi_1A - phi_2A)*cos(phi_2B) - cos(theta_A -
    theta_B)*sin(phi_1A - phi_2A)*sin(phi_2B))*(y_1B - h + y_2B)
    + t*sin(phi_1A - phi_2A)*cos(theta_A), sin(phi_1A), (sin(
    phi_1A - phi_2A)*sin(phi_2B) - cos(phi_1A - phi_2A)*cos(theta_A -
    theta_B)*cos(phi_2B))*(y_1B - h + y_2B), - sin(phi_1A -
    phi_2A)*cos(phi_2B) - cos(phi_1A - phi_2A)*cos(theta_A -
    theta_B)*sin(phi_2B), 0, - sin(phi_1A - phi_2A)*cos(phi_2B) -
    cos(phi_1A - phi_2A)*cos(theta_A - theta_B)*sin(phi_2B)];
15 f3 = [
                           0,
                           0,
                           0,
                           sin(theta_A - theta_B)*
                           cos(phi_2B)*(y_1B - h + y_2B),
                           sin(theta_A - theta_B)*sin(phi_2B),
                           0,
                           sin(theta_A - theta_B)*sin(phi_2B)];
16 J = [f1; f2; f3];
17 end

```

A.7. B_t

```

1 function B_t = B_t(q, b, p)
2 phi_1A = q(1);
3 y_1A = q(2);
4 phi_2A = q(3);
5 y_2A = q(4);
6 %theta_A = q(5);
7 %theta_B = q(6);
8 phi_2B = q(7);
9 y_2B = q(8);
10 phi_1B = q(9);
11 y_1B = q(10);
12
13 f1 = [ ((y_1A + p*sin(phi_1A))*(2*b - 2*p*cos(phi_1A)))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)) - ((y_1A - p*sin(phi_1A))*(2*b - 2*p*cos(phi_1A)))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)), ((y_1A - p*sin(phi_1A))*(2*cos(phi_1A)*(b - p*cos(phi_1A)) - 2*sin(phi_1A)*(y_1A + p*sin(phi_1A))))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)) - ((y_1A + p*sin(phi_1A))*(2*cos(phi_1A)*(b - p*cos(phi_1A)) + 2*sin(phi_1A)*(y_1A - p*sin(phi_1A))))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2))];
14 f2 = [((b*sin(phi_1A) - y_1A*cos(phi_1A))*(2*b - 2*p*cos(phi_1A)))/(4*(y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)) - ((b*sin(phi_1A) + y_1A*cos(phi_1A))*(2*b - 2*p*cos(phi_1A)))/(4*(y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)), ((b*sin(phi_1A) + y_1A*cos(phi_1A))*(2*cos(phi_1A)*(b - p*cos(phi_1A)) + 2*sin(phi_1A)*(y_1A - p*sin(phi_1A))))/(4*(y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)) - ((b*sin(phi_1A) - y_1A*cos(phi_1A))*(2*cos(phi_1A)*(b - p*cos(phi_1A)) - 2*sin(phi_1A)*(y_1A + p*sin(phi_1A))))/(4*(y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2))];
15
16 f3 = [ ((y_2A + p*sin(phi_2A))*(2*b - 2*p*cos(phi_2A)))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)) - ((y_2A - p*sin(phi_2A))*(2*b - 2*p*cos(phi_2A)))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)), ((y_2A - p*sin(phi_2A))*(2*cos(phi_2A)*(b - p*cos(phi_2A)) - 2*sin(phi_2A)*(y_2A + p*sin(phi_2A))))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)) - ((y_2A + p*sin(phi_2A))*(2*cos(phi_2A)*(b - p*cos(phi_2A)) + 2*sin(phi_2A)*(y_2A - p*sin(phi_2A))))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2))];
17 f4 = [((b*sin(phi_2A) - y_2A*cos(phi_2A))*(2*b - 2*p*cos(phi_2A)))/(4*(y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)) - ((b*sin(phi_2A) + y_2A*cos(phi_2A))*(2*b - 2*p*cos(phi_2A)))/(4*(y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)), ((b*sin(phi_2A) + y_2A*cos(phi_2A))*(2*cos(phi_2A)*(b - p*cos(phi_2A)) + 2*sin(phi_2A)*(y_2A - p*sin(phi_2A))))/(4*(y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)) - ((b*sin(phi_2A) - y_2A*cos(phi_2A))*(2*cos(phi_2A)*(b - p*cos(phi_2A)) - 2*sin(phi_2A)*(y_2A + p*sin(phi_2A))))/(4*(y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2))];
18
19 f5 = [ ((y_2B + p*sin(phi_2B))*(2*b - 2*p*cos(phi_2B)))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2)) - ((y_2B - p*sin(phi_2B))*(2*b - 2*p*cos(phi_2B)))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2)), ((y_2B - p*sin(phi_2B))*(2*cos(phi_2B)*(b - p*cos(phi_2B)) - 2*sin(phi_2B)*(y_2B + p*sin(phi_2B))))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2))]
```

```

y_2B + p*sin(phi_2B)))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2)) - ((y_2B + p*sin(phi_2B))*(2*cos(phi_2B)*(b - p*cos(phi_2B)) + 2*sin(phi_2B)*(y_2B - p*sin(phi_2B))))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2));
20 f6 = [((b*sin(phi_2B) - y_2B*cos(phi_2B))*(2*b - 2*p*cos(phi_2B)))/(4*(y_2B^2*cos(phi_2B) - b*p*sin(phi_2B)^2)) - ((b*sin(phi_2B) + y_2B*cos(phi_2B))*(2*b - 2*p*cos(phi_2B)))/(4*(y_2B^2*cos(phi_2B) - b*p*sin(phi_2B)^2)), ((b*sin(phi_2B) + y_2B*cos(phi_2B))*(2*cos(phi_2B)*(b - p*cos(phi_2B)) + 2*sin(phi_2B)*(y_2B - p*sin(phi_2B))))/(4*(y_2B^2*cos(phi_2B) - b*p*sin(phi_2B)^2)) - ((b*sin(phi_2B) - y_2B*cos(phi_2B))*(2*cos(phi_2B)*(b - p*cos(phi_2B)) - 2*sin(phi_2B)*(y_2B + p*sin(phi_2B))))/(4*(y_2B^2*cos(phi_2B) - b*p*sin(phi_2B)^2));
21
22 f7 = [((y_1B + p*sin(phi_1B))*(2*b - 2*p*cos(phi_1B)))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2)) - ((y_1B - p*sin(phi_1B))*(2*b - 2*p*cos(phi_1B)))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2)), ((y_1B - p*sin(phi_1B))*(2*cos(phi_1B)*(b - p*cos(phi_1B)) - 2*sin(phi_1B)*(y_1B + p*sin(phi_1B))))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2)) - ((y_1B + p*sin(phi_1B))*(2*cos(phi_1B)*(b - p*cos(phi_1B)) + 2*sin(phi_1B)*(y_1B - p*sin(phi_1B))))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2));
23 f8 = [((b*sin(phi_1B) - y_1B*cos(phi_1B))*(2*b - 2*p*cos(phi_1B)))/(4*(y_1B^2*cos(phi_1B) - b*p*sin(phi_1B)^2)) - ((b*sin(phi_1B) + y_1B*cos(phi_1B))*(2*b - 2*p*cos(phi_1B)))/(4*(y_1B^2*cos(phi_1B) - b*p*sin(phi_1B)^2)), ((b*sin(phi_1B) + y_1B*cos(phi_1B))*(2*cos(phi_1B)*(b - p*cos(phi_1B)) + 2*sin(phi_1B)*(y_1B - p*sin(phi_1B))))/(4*(y_1B^2*cos(phi_1B) - b*p*sin(phi_1B)^2)) - ((b*sin(phi_1B) - y_1B*cos(phi_1B))*(2*cos(phi_1B)*(b - p*cos(phi_1B)) - 2*sin(phi_1B)*(y_1B + p*sin(phi_1B))))/(4*(y_1B^2*cos(phi_1B) - b*p*sin(phi_1B)^2));
24
25 B_t = [f1; f2; f3; f4; f5; f6; f7; f8];
26 end

```

A.8. H1

```
1 function H = H1(q)
2 phi_1A = q(1);
3 %y_1A = q(2);
4 phi_2A = q(3);
5 %y_2A = q(4);
6 theta_A = q(5);
7 theta_B = q(6);
8 %phi_2B = q(7);
9 %y_2B = q(8);
10 %phi_1B = q(9);
11 %y_1B = q(10);
12
13 f1 = [ 0, 0, -cos(phi_1A - phi_2A)*sin(theta_A - theta_B), cos(
14     phi_1A - phi_2A)*sin(theta_A - theta_B)];
14 f2 = [ 0, 0, sin(phi_1A - phi_2A)*sin(theta_A - theta_B), -sin(
15     phi_1A - phi_2A)*sin(theta_A - theta_B)];
15 f3 = [-1, 1,
16                 -cos(theta_A - theta_B),
17                         cos(theta_A - theta_B)];
18
17 H = [f1; f2; f3];
18 end
```

A.9. H2

```
1 function H = H2(xyz)
2 alpha = xyz(1);
3 beta = xyz(2);
4
5 f1 = [1, 0, sin(beta)];
6 f2 = [0, cos(alpha), -cos(beta)*sin(alpha)];
7 f3 = [0, sin(alpha), cos(alpha)*cos(beta)];
8
9 H = [f1; f2; f3];
10 end
```

A.10. B_t_prima

```

1 function B_t = B_t_prima(q, b, p)
2 phi_1A = q(1);
3 y_1A = q(2);
4 phi_2A = q(3);
5 y_2A = q(4);
6 %theta_A = q(5);
7 %theta_B = q(6);
8 phi_2B = q(7);
9 y_2B = q(8);
10 phi_1B = q(9);
11 y_1B = q(10);
12
13 f1 = [ ((y_1A + p*sin(phi_1A))*(2*b - 2*p*cos(phi_1A)))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)) - ((y_1A - p*sin(phi_1A))*(2*b - 2*p*cos(phi_1A)))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)), ((y_1A - p*sin(phi_1A))*(2*cos(phi_1A)*(b - p*cos(phi_1A)) - 2*sin(phi_1A)*(y_1A + p*sin(phi_1A))))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2)) - ((y_1A + p*sin(phi_1A))*(2*cos(phi_1A)*(b - p*cos(phi_1A)) + 2*sin(phi_1A)*(y_1A - p*sin(phi_1A))))/(4*(p*y_1A^2*cos(phi_1A) - b*p^2*sin(phi_1A)^2))];
14
15 f3 = [ ((y_2A + p*sin(phi_2A))*(2*b - 2*p*cos(phi_2A)))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)) - ((y_2A - p*sin(phi_2A))*(2*b - 2*p*cos(phi_2A)))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)), ((y_2A - p*sin(phi_2A))*(2*cos(phi_2A)*(b - p*cos(phi_2A)) - 2*sin(phi_2A)*(y_2A + p*sin(phi_2A))))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2)) - ((y_2A + p*sin(phi_2A))*(2*cos(phi_2A)*(b - p*cos(phi_2A)) + 2*sin(phi_2A)*(y_2A - p*sin(phi_2A))))/(4*(p*y_2A^2*cos(phi_2A) - b*p^2*sin(phi_2A)^2))];
16
17 f5 = [ ((y_2B + p*sin(phi_2B))*(2*b - 2*p*cos(phi_2B)))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2)) - ((y_2B - p*sin(phi_2B))*(2*b - 2*p*cos(phi_2B)))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2)), ((y_2B - p*sin(phi_2B))*(2*cos(phi_2B)*(b - p*cos(phi_2B)) - 2*sin(phi_2B)*(y_2B + p*sin(phi_2B))))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2)) - ((y_2B + p*sin(phi_2B))*(2*cos(phi_2B)*(b - p*cos(phi_2B)) + 2*sin(phi_2B)*(y_2B - p*sin(phi_2B))))/(4*(p*y_2B^2*cos(phi_2B) - b*p^2*sin(phi_2B)^2))];
18
19 f7 = [ ((y_1B + p*sin(phi_1B))*(2*b - 2*p*cos(phi_1B)))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2)) - ((y_1B - p*sin(phi_1B))*(2*b - 2*p*cos(phi_1B)))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2)), ((y_1B - p*sin(phi_1B))*(2*cos(phi_1B)*(b - p*cos(phi_1B)) - 2*sin(phi_1B)*(y_1B + p*sin(phi_1B))))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2)) - ((y_1B + p*sin(phi_1B))*(2*cos(phi_1B)*(b - p*cos(phi_1B)) + 2*sin(phi_1B)*(y_1B - p*sin(phi_1B))))/(4*(p*y_1B^2*cos(phi_1B) - b*p^2*sin(phi_1B)^2))];
20
21 B_t = [f1; f3; f5; f7];
22 end

```

Referencias

- Bennett, D. and Hollerbach, J. (1991). Autonomous calibration of single-loop closed kinematic chains formed by manipulators with passive endpoint constraints. *IEEE Transactions on Robotics and Automation*, 7(5):597–606.
- Corke, P. (2017). *Robotics, Vision and Control*. Springer International Publishing.
- Everett, L. J. and Lin, C. (1988). Kinematic calibration of manipulators with closed loop actuated joints. In *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pages 792–797. IEEE.
- Meggiolaro, M., Scrifignano, G., and Dubowsky, S. (2002). Analysis and optimization of closed-loop manipulator calibration with fixed endpoint. In *Proceedings 2002 CONEM II National Congress of Mechanical Engineering*.
- Peidró, A., Gil, A., Marín, J. M., Berenguer, Y., Payá, L., and Reinoso, O. (2016). Monte-carlo workspace calculation of a serial-parallel biped robot. In *Robot 2015: Second Iberian Robotics Conference*, pages 157–169. Springer.
- Peidró, A., Gil, A., Marín, J. M., Berenguer, Y., and Reinoso, O. (2015). Kinematic analysis and simulation of a hybrid biped climbing robot. In *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, volume 2, pages 24–34. IEEE.
- Peidró, A., Tavakoli, M., Marín, J. M., and Reinoso, Ó. (2019). Design of compact switchable magnetic grippers for the HyReCRo structure-climbing robot. *Mechatronics*, 59:199–212.
- Roth, Z., Mooring, B., and Ravani, B. (1987). An overview of robot calibration. *IEEE Journal on Robotics and Automation*, 3(5):377–385.
- Schmidt, D. and Berns, K. (2013). Climbing robots for maintenance and inspections of vertical structures—a survey of design aspects and technologies. *Robotics and Autonomous Systems*, 61(12):1288–1305.
- Tavakoli, M., Marques, L., and de Almeida, A. T. (2011). 3DCLIMBER: Climbing and manipulation over 3D structures. *Mechatronics*, 21(1):48–62.
- Zhuang, H., Motaghedi, S. H., and Roth, Z. S. (1999). Robot calibration with planar constraints. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, volume 1, pages 805–810. IEEE.