UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"RESOLUCIÓN CINEMÁTICA DE ROBOTS MODULARES MEDIANTE CURVAS DE RESTRICCIÓN"

TRABAJO FIN DE GRADO

Diciembre -2025

AUTOR: Abdelghani Ouakari Raqoui

DIRECTOR: Adrián Peidro Vidal

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

1.1. ANTECEDENTES

La robótica modular se ha convertido en un campo de investigación de gran relevancia dentro de la ingeniería moderna. Estos sistemas se caracterizan por estar formados por múltiples módulos que pueden combinarse de diferentes maneras, lo que les otorga una gran flexibilidad y adaptabilidad frente a los robots tradicionales.

Esta capacidad de reconfiguración se organiza en tres tipologías fundamentales: la Arquitectura de Cadena, la Arquitectura de Rejilla y la Arquitectura Híbrida

Gracias a esta capacidad, los robots modulares encuentran aplicaciones en ámbitos tan diversos como la automatización industrial, la exploración espacial o la asistencia médica.

No obstante, esta versatilidad plantea un reto fundamental: la resolución de sistemas de ecuaciones no lineales derivados de la cinemática de robots con múltiples grados de libertad. En el caso de los robots estudiados en este Trabajo de Fin de Grado, el número de incógnitas puede llegar a ser muy elevado (hasta 21 en el robot más sencillo), lo que convierte el problema en un desafío tanto teórico como computacional.

1.1.1. CLASIFICACIÓN DE ARQUITECTURAS Y APLICACIÓN

La clasificación principal de la robótica modular se basa en la forma en que los módulos se conectan y estructuran:

- **Arquitectura de Cadena:** Los módulos se conectan secuencialmente, formando estructuras lineales, serpentinas o de brazo. Son altamente articulados y pueden manipular el entorno o moverse a través de espacios estrechos.

- **Arquitectura de Rejilla:** Los módulos se unen a sus vecinos en un patrón tridimensional fijo, formando estructuras cúbicas o celosías. Son sistemas compactos ideales para formar estructuras de soporte o vehículos móviles robustos.
- **Arquitectura Híbrida:** Combina las características de cadena y rejilla. Los módulos pueden ser lineales, pero se conectan en un plano. Esto permite tanto la locomoción como la manipulación.

Dentro de estas tres familias de robots modulares, existe una gran variedad de ejemplos propuestos por diferentes universidades y laboratorios de investigación en todo el mundo. Entre los de tipo cadena, destacan, por ejemplo, los robots CONRO (Shen et al., 2000), mientras que un ejemplo característico de robot tipo rejilla son los M-blocks (Romanishin et al., 2015). En este TFG nos centraremos en la familia de robots modulares de tipo cadena, que ofrecen una mayor versatilidad y libertad de movimiento, al no restringir las posiciones relativas entre módulos a solo un conjunto discreto de posiciones, como ocurre con los robots modulares del tipo rejilla.

Dentro de la familia de los robots modulares de tipo cadena, es destacable el hecho de que, hasta la fecha, la inmensa mayoría de tales robots modulares se han concebido para formar cadenas cinemáticas abiertas, donde los módulos se van sucediendo uno tras otro formando estructuras serpentinas o de tipo brazo, como se ha comentado antes. Esto se debe a que las cadenas cinemáticas abiertas son más fáciles de modelar desde los puntos de vista cinemático y dinámico. Sin embargo, esto también limita las potenciales capacidades de los robots modulares, pues las cadenas cinemáticas abiertas suelen disponer de una rigidez y capacidad de carga limitadas, debido al hecho de que cada módulo debe sostener al resto de módulos que van tras él en la cadena.

Para solucionar esta limitación, en el grupo de investigación de Automatización, Robótica y Visión por Computador (ARVC) de la UMH, recientemente se ha propuesto un nuevo tipo de robot modular, el MASAR (Modular And Single Actuator Robot). Este módulo consta de un único motor encargado de mover una serie de unidades de adhesión (también llamados pads o puertos) mediante engranajes cónicos, de manera que, adhiriendo uno de esos puertos al entorno

(mediante imanes o ventosas), y actuando dicho único motor, se logra que el robot completo pivote alrededor del puerto fijado al entorno. Alternando el puerto que se fija al entorno, el módulo MASAR se convierte en un robot móvil y es posible lograr su avance a lo largo de distintos planos. En (Peidró et al., 2019) se presentó el análisis cinemático de un único módulo MASAR como robot móvil, para resolver la planificación de sus movimientos en un plano.

El interés del robot MASAR no se limita a su comportamiento como robot móvil con un único actuador, sino que también resulta relevante su comportamiento como robot modular. Efectivamente, uniendo una pluralidad de módulos MASAR a través de sus puertos de adhesión, se pueden formar robots modulares multi-módulo con un mayor número de grados de libertad y con la capacidad de ejecutar tareas que un único módulo sería incapaz de acometer. Además, se ha verificado la viabilidad de combinar módulos MASAR para formar robots modulares con cadenas cinemáticas cerradas, contribuyendo a paliar la falta de dichos robots modulares en la literatura científica, y expandiendo las aplicaciones de los robots modulares a tareas en las que se requieren mayores rigideces y capacidades de carga que las cadenas seriales o abiertas no pueden proporcionar.

Como robot modular multi-módulo, el robot MASAR ha sido objeto de dos estudios previos. En primer lugar, en (Peidró et al., 2025), se presentó una formulación cinemática que permite resolver las ecuaciones de lazo del robot, ensamblando una serie de módulos MASAR en un robot modular y estudiar su movilidad y su espacio de trabajo. En segundo lugar, y partiendo de la formulación cinemática del trabajo anterior, en (Prieto, 2025) se presentó una formulación dinámica Lagrangiana que permite construir las ecuaciones de movimiento de robots modulares MASAR con un número arbitrario de módulos, simulando su dinámica directa y allanando el camino para el futuro diseño de controladores de par computado.

Aunque la cinemática de los robots modulares MASAR, tanto de cadenas cinemáticas como abiertas, ha sido resuelta en (Peidró et al., 2025), la solución que en dicho trabajo se presenta tiene un carácter local, en el sentido de que se parte de una configuración inicial del robot modular que satisfaga todas las

restricciones de unión multi-módulo, y se estudia la movilidad y las configuraciones alcanzables por el robot partiendo de dicha configuración inicial. Aún queda por responder, por tanto, diversas cuestiones referentes a la existencia de soluciones del robot (soluciones a su cinemática inversa, es decir, cuántas configuraciones del robot permiten posicionar su efector final en una posición deseada), y al número máximo de tales soluciones.

Para ahondar en esta cuestión, en este TFG se plantea el estudio de la resolución de la cinemática inversa de los robots modulares MASAR mediante métodos que proporcionen más soluciones que el método local mencionado en el párrafo anterior. Para ello, se comenzará con el método de homotopía, que pese a resultar prometedor en un inicio, resultó inviable dado el gran número de soluciones candidatas que genera para los robots MASAR. Tras esto, se abordará un nuevo método, basado en (Peidró et al., 2020), que relaja el problema cinemático inverso suprimiendo una de las ecuaciones a resolver, lo cual define una curva (que llamamos la curva de restricción) en la que pueden encontrarse diversas soluciones del problema cinemático inverso.

Estos métodos se abordarán con dos robots modulares MASAR:

- Primer robot: Una arquitectura sencilla en cadena abierta, en la que uno de los extremos está fijado al suelo y el otro permanece libre, funcionando como un brazo robótico.

- Segundo robot: Una arquitectura en cadena cerrada más extensa y compleja, compuesta por nueve módulos, cuyos dos extremos están fijos al suelo. En este caso, el efector final corresponde al módulo central de robot.

1.2. MOTIVACIÓN

Lo interesante y fundamental de este trabajo es que, al comenzar con el estudio de un robot sencillo y básico, podemos escalar hacia sistemas más complejos sin que los cálculos se vuelvan un problema. El código desarrollado para el primer robot resulta igualmente útil para el segundo; la única diferencia radica en las ecuaciones de restricción y en las variables que se modifican. Sin embargo,

la estructura del código para la cinemática y la búsqueda de soluciones permanece prácticamente igual.

Esta es precisamente la motivación detrás de nuestro enfoque: a partir de un robot simple, es posible construir robots más complejos sin preocuparnos por la dificultad creciente de los cálculos.

1.3. OBJETO DE TRABAJO

El objetivo de este trabajo es desarrollar un código en Matlab que permita resolver la problemática de determinar las incógnitas cinemáticas de un robot para lograr alcanzar un punto definido. La idea es que, en lugar de controlar manualmente el movimiento paso a paso, como así ocurría en el trabajo previo (Peidró et al., 2025), podamos programar un sistema en el que se establezca directamente la posición final deseada obtengamos varias soluciones que alcanzan dicha posición.

1.4. ESTRUCTURA DE LA MEMORIA

La memoria está compuesta de 6 capítulos principales. El primer de ello es la introducción que acabamos de leer, exposición de los antecedentes, motivación y objetivos del trabajo de fin de grado

En el segundo capítulo se trata sobre descripción de robots modulares MASAR: explicando diseño y las configuraciones adoptadas, al igual que los fundamentos teóricos mediante los cuales se ha realizado la programación en Matlab.

El tercer capítulo está dedicado al estudio y aplicación del método de homotopía, revisando si realmente se puede usar en este tipo de robots y comentando los problemas que aparecen al intentar aplicarlo.

El cuarto capítulo trata sobre la resolución de los movimientos de los robots modulares a través de la curva de restricción. Este es el capítulo más importante

y el núcleo central del trabajo, porque aquí es donde se obtuvieron los resultados y se muestran las gráficas que los representan.

El quinto capítulo donde se expone la conclusión, descartando la homotopía por inviable y aplicando con éxito el método de la curva de restricción para hallar múltiples soluciones en los robots analizados.

Finalmente, el sexto capítulo recogerá los anexos de esta memoria, los cuales contendrán la programación realizada en Matlab.

2. DESCRIPCIÓN ROBOTS MODULARES MASAR

2.1. INTRODUCCIÓN

Los robots modulares MASAR se fundamentan en la idea de que, a partir de unidades básicas idénticas, es posible construir sistemas robóticos cada vez más complejos: cada módulo puede operar de manera independiente, pero al conectarse con otros libera todo su potencial, generando estructuras capaces de movimientos más sofisticados; en este capítulo se muestran dos configuraciones que ilustran esa evolución, comenzando con un robot sencillo de tres módulos que controla un punto extremo en un plano horizontal y avanzando hacia un robot de nueve módulos con tres grados de libertad.

2.2. EL MÓDULO MASAR

Cada módulo MASAR es una unidad robótica autónoma y estandarizada, diseñada para poder conectarse con otros módulos y formar robots de distinta morfología.

- Coordenadas de configuración: Cada módulo $i$se describe mediante un vector de 7 variables:

$$\mathbf{X}^i = \left[x^i, y^i, z^i, \alpha^i, \beta^i, \gamma^i, \theta^i\right]^T \quad (1)$$

donde:

- (x_i, y_i, z_i): posición del centro del módulo.
- $(\alpha_i, \beta_i, \gamma_i)$: ángulos de Euler XYZ intrínsecos que definen la orientación.
- θ_i: ángulo girado por el motor interno del módulo.
- Puertos de conexión: Cada módulo dispone de múltiples puertos (A1, A2, …, A10) que permiten uniones axiales y transversales. Estos puertos son los que posibilitan la construcción de robots modulares más grandes.

- Representación matemática: La posición y orientación de cada módulo se expresan mediante una matriz de transformación homogénea T^ide tamaño 4×4, que combina rotaciones y traslaciones en el espacio tridimensional.

$$\mathbf{T}^i = \begin{bmatrix} \mathbf{R}_X(\alpha^i)\mathbf{R}_Y(\beta^i)\,\mathbf{R}_Z(\gamma^i) & [x^i, y^i, z^i]^T \\ 0\,0\,0 & 1 \end{bmatrix} \qquad (2)$$

donde $R_v(u)$ representa una rotación de ángulo u alrededor del eje v.

Antes de analizar el robot completo, es necesario recordar cómo está configurado cada módulo MASAR de manera individual.

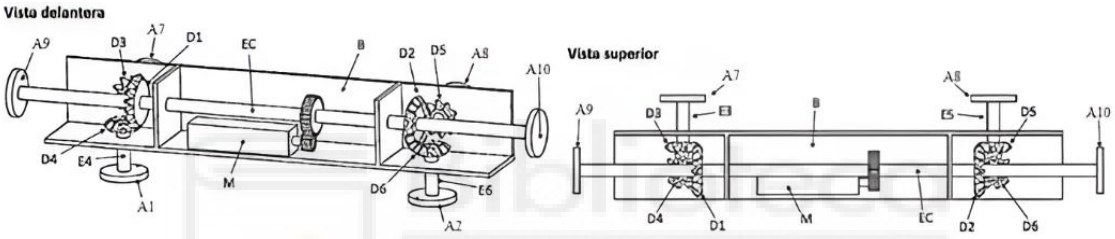

Figura 1: Módulo robótico MASAR

2.3. CONSTRUCCIÓN DEL PRIMER ROBOT MODULAR

El primer robot se forma uniendo tres módulos MASAR en serie, tal como se muestra en la Figura 2.

- El módulo 1 está fijado al suelo.
- El módulo 2 se conecta al módulo 1 mediante un puerto axial.
- El módulo 3 se acopla al módulo 2 y se convierte en el extremo libre del robot.

El resultado es una estructura sencilla que permite controlar la posición de un punto extremo $P = (x, y)$en un plano horizontal.

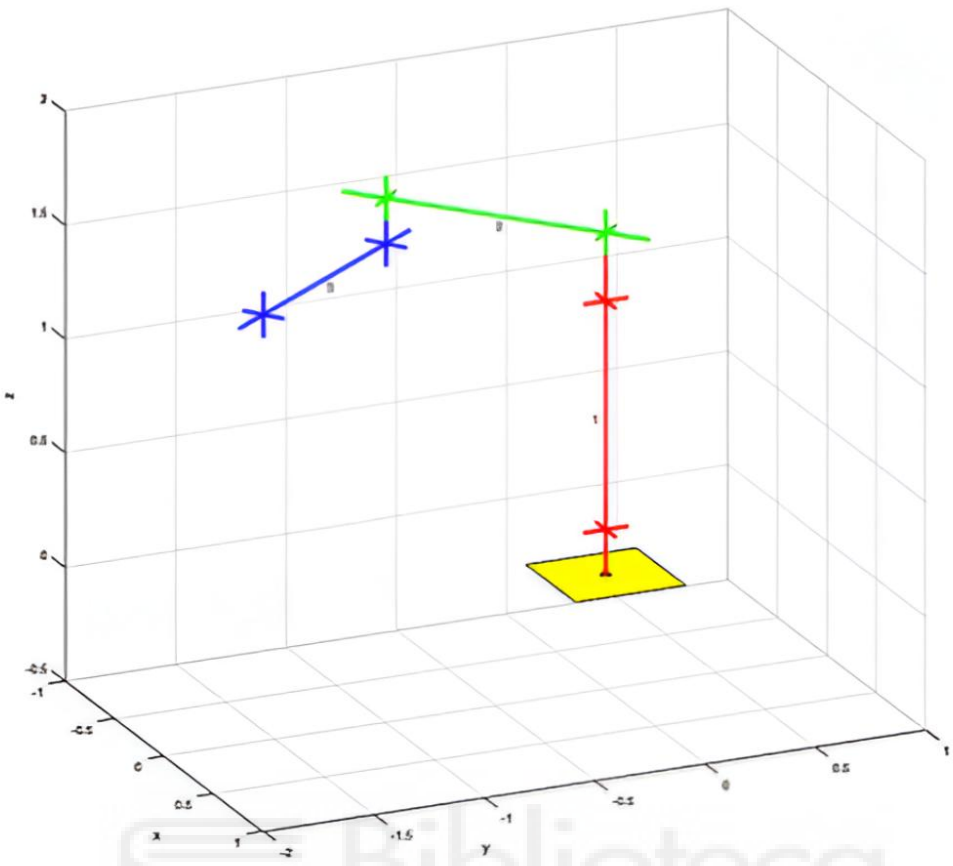

Figura 2: Primer robot modular formado por 3 módulos MASAR.

2.4. PARÁMETROS DE DISEÑO

El comportamiento del robot depende de cuatro parámetros geométricos fundamentales:

- a: distancia entre los centros de los engranajes cónicos.
- b: distancia desde un centro de engranaje hasta el puerto de conexión transversal más cercano.
- c: distancia entre cualquiera de los centros de los engranajes cónicos y el puerto de conexión axial más cercano.
- n: relación de reducción de los engranajes cónicos $(n > 1)$.

Estos parámetros son constantes de diseño, fijados una vez definido el módulo MASAR.

2.5. RESTRICCIONES CINEMÁTICAS

Aunque cada módulo libre en el espacio, sus 7 coordenadas pueden tomar cualquier valor libremente. Al conectarlos aparecen restricciones de unión que reducen las variables independientes.

Las restricciones principales son:

- Unión módulo 1 – módulo 2: asegura la continuidad geométrica entre ambos.
- Unión módulo 2 – módulo 3: garantiza la correcta transmisión de movimiento.
- Unión módulo 1 – suelo: fija la base del robot al entorno.

Estas uniones se expresan mediante ecuaciones matriciales que imponen condiciones de compatibilidad entre las transformaciones homogéneas de los módulos. Que son los siguientes:

Unión entre el módulo 1 y el 2:

$$\mathbf{T}^1 \begin{bmatrix} \mathbf{R}_Y\left(\frac{\pi}{2}\right)\mathbf{R}_Z(\theta^1) & \begin{bmatrix} \frac{a}{2}+c \\ 0 \\ 0 \end{bmatrix} \\ 0\,0\,0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_X(\pi) & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ 0\,0\,0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_Z(-n\,\theta^2) & \begin{bmatrix} -\frac{a}{2} \\ 0 \\ b \end{bmatrix} \\ 0\,0\,0 & 1 \end{bmatrix}^{-1} (\mathbf{T}^2)^{-1} = \mathbf{I}_4 \quad (4)$$

donde $\mathbf{I}_4$ es la matriz identidad de tamaño 4 y el puerto de conexión axial más cercano (A9 o A10, según la Figura 1).

Unión entre el módulo 2 y el 3:

$$\mathbf{T}^2 \begin{bmatrix} \mathbf{R}_Z(n\,\theta^2) & \begin{bmatrix} \frac{a}{2} \\ 0 \\ b \end{bmatrix} \\ 0\,0\,0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_X(\pi) & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ 0\,0\,0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R}_Z(-n\,\theta^3) & \begin{bmatrix} -\frac{a}{2} \\ 0 \\ b \end{bmatrix} \\ 0\,0\,0 & 1 \end{bmatrix}^{-1} (\mathbf{T}^3)^{-1} = \mathbf{I}_4 \quad (5)$$

Unión entre el módulo 1 y el suelo:

$$\mathbf{T}^1 \begin{bmatrix} \mathbf{R}_Y\left(-\frac{\pi}{2}\right)\ \mathbf{R}_Z(-\theta^1) & \begin{bmatrix} -\frac{a}{2}-c \\ 0 \\ 0 \end{bmatrix} \\ 0\ 0\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ 0\ 0\ 0 & 1 \end{bmatrix} \qquad (6)$$

2.6. PROBLEMA A RESOLVER

El objetivo es el siguiente: dada una posición deseada (x, y) del extremo P del módulo 3, calcular las 21 incógnitas correspondientes a los vectores X_1, X_2, X_3.

La configuración en serie de los tres módulos MASAR da lugar a un robot sencillo cuyo extremo libre pertenece al módulo 3 (en azul de la figura 2) y puede controlarse en el plano horizontal mediante la posición $P = (x, y)$. La posición del punto extremo P se obtiene a partir de los parámetros de configuración del módulo 3, empleando su matriz de transformación homogénea T_3definida en la ecuación (2).

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \mathbf{T}^3 \begin{bmatrix} \mathbf{R}_X(\pi)\ \mathbf{R}_Z(n\,\theta^3) & \left[\frac{a}{2}, 0, -b\right]^T \\ 0\ 0\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad (3)$$

Este problema se formula como un sistema de ecuaciones no lineales que combina:

- La ecuación de posición del extremo (ecuación 3).
- Las tres ecuaciones de unión (ecuaciones 4, 5 y 6).

2.7. MÉTODOS DE RESOLUCIÓN

Para este trabajo, hemos planteado la resolución mediante los siguientes enfoques:

- Eliminación algebraica: consiste en manipular las ecuaciones para reducir el número de incógnitas hasta llegar a 1 ecuación con 1 incógnita, que suele ser una ecuación polinómica de grado muy elevado. Sin embargo, puede volverse muy complejo en sistemas con muchos módulos. De hecho, para los robots modulares estudiados en este TFG, este enfoque se vuelve inviable, debido al gran tamaño que adquieren las ecuaciones a medida que se van eliminando incógnitas (hay un total de 21 incógnitas para el primer robot, que es el más sencillo).
- Método de homotopía: técnica numérica que permite encontrar soluciones globales a sistemas no lineales, partiendo de un problema más sencillo y deformándolo progresivamente hasta llegar al problema real. Este método es especialmente prometedor para robots modulares con múltiples grados de libertad. No obstante, como se mostrará en capítulos posteriores, la homotopía no constituye un método infalible: la probabilidad de encontrar soluciones es relativamente baja y el coste computacional requerido es considerable.
- Método de curvas de restricción: Como alternativa, se propone un nuevo enfoque basado en curvas de restricción. Este método consiste en converger mediante el método de Newton a cierta curva, que representa la satisfacción de todas las ecuaciones del problema excepto una ecuación. A continuación, se recorre dicha curva mediante un procedimiento de marcha a lo largo de la misma, para encontrar los puntos en los que se cumpliría la ecuación restante.

2.8. EL SEGUNDO ROBOT MODULAR MASAR

El segundo prototipo analizado en este trabajo corresponde a una configuración considerablemente más compleja que la del primer robot modular, ya que en este caso el sistema está constituido por nueve módulos MASAR conectados de manera sucesiva, lo que le otorga una capacidad de movimiento y control mucho mayor. La estructura se organiza de forma que dos módulos se encuentran fijados al suelo en los extremos, actuando como bases rígidas que proporcionan estabilidad y soporte al conjunto. A partir de estas bases, los demás módulos se enlazan uno tras otro, conformando una cadena estructural que transmite tanto la rigidez como la flexibilidad necesaria para que el robot pueda ejecutar movimientos coordinados. Dentro de esta disposición, el módulo 8 ocupa una posición central y se convierte en el punto de referencia principal para el control del sistema, funcionando como el extremo móvil sobre el cual se concentran las acciones de regulación y seguimiento.

El parámetro de interés en este prototipo es la posición del centro de coordenadas del módulo 8, que se controla ahora en un espacio tridimensional, a diferencia del primer robot en el que el control se limitaba a un plano horizontal. Esta diferencia marca un salto cualitativo en las capacidades del sistema, pues el robot dispone de tres grados de libertad (3 GDL), lo que significa que el punto de control asociado al módulo 8 puede desplazarse en las tres direcciones del espacio (x, y, z). Gracias a esta característica, el prototipo es capaz de realizar movimientos más versátiles y complejos, ampliando las posibilidades de aplicación en tareas que requieren precisión espacial y adaptabilidad.

La representación gráfica en la figura 3 muestra con claridad cómo los módulos se organizan dentro de la estructura, destacando el papel central del módulo 8 como punto de control y evidenciando la fijación de los módulos extremos al suelo. Esta disposición no solo asegura la estabilidad del sistema, sino que también permite visualizar de manera intuitiva la transición desde un robot sencillo de tres módulos hacia una configuración más avanzada, capaz de desenvolverse en un entorno tridimensional con mayor autonomía y sofisticación.

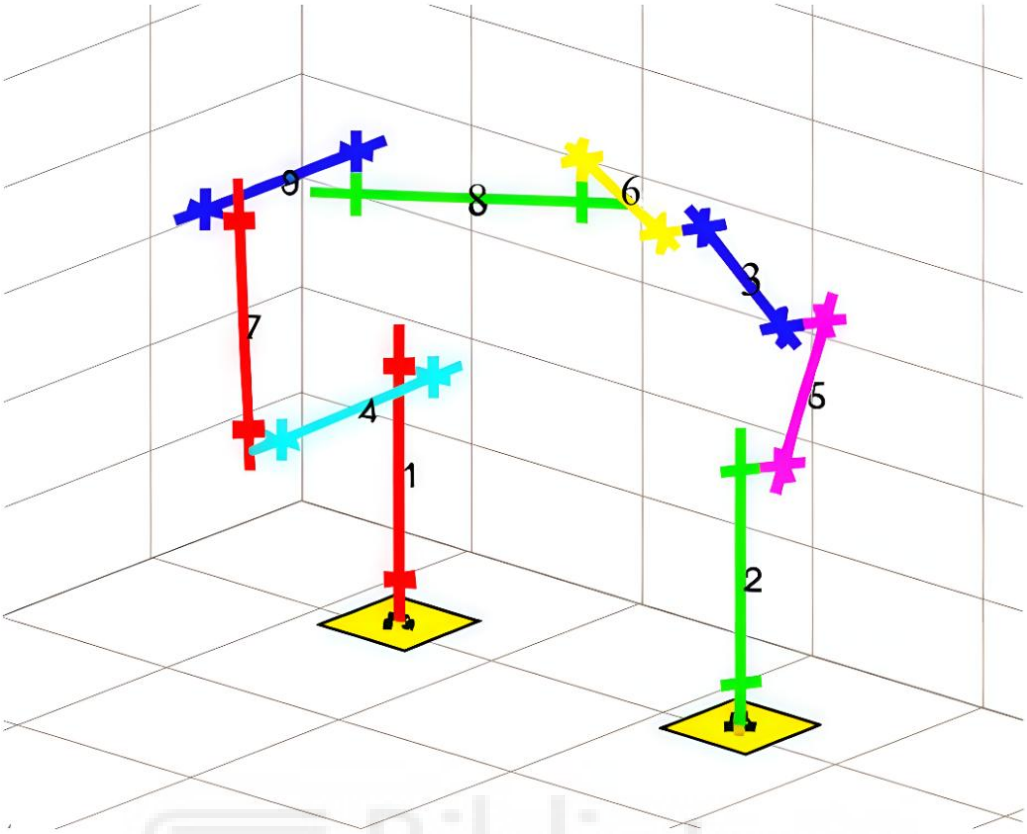

Figura 3: Segundo robot modular formado por 9 módulos MASAR

Conexiones entre los módulos MASAR

En la figura se aprecia con claridad la disposición de los nueve módulos MASAR, representados mediante círculos numerados que facilitan la identificación de cada unidad. Las líneas que enlazan estos círculos señalan las conexiones mecánicas establecidas entre los módulos, mostrando cómo se articula la cadena estructural del robot. Además, junto a cada línea aparece un número que identifica la conexión específica, lo que permite seguir de manera ordenada la secuencia de acoplamientos y comprender mejor la lógica de ensamblaje del sistema. Las restricciones de este robot serían análogas a las de las ecuaciones $(4, 5\ y\ 6)$ del primer robot, pero teniendo en cuenta que hay bastantes más uniones entre módulos. Por tanto, no se mostrarán en detalle las ecuaciones en este punto, sino que se podrán encontrar en los códigos Matlab mostrados posteriormente y en los anexos.

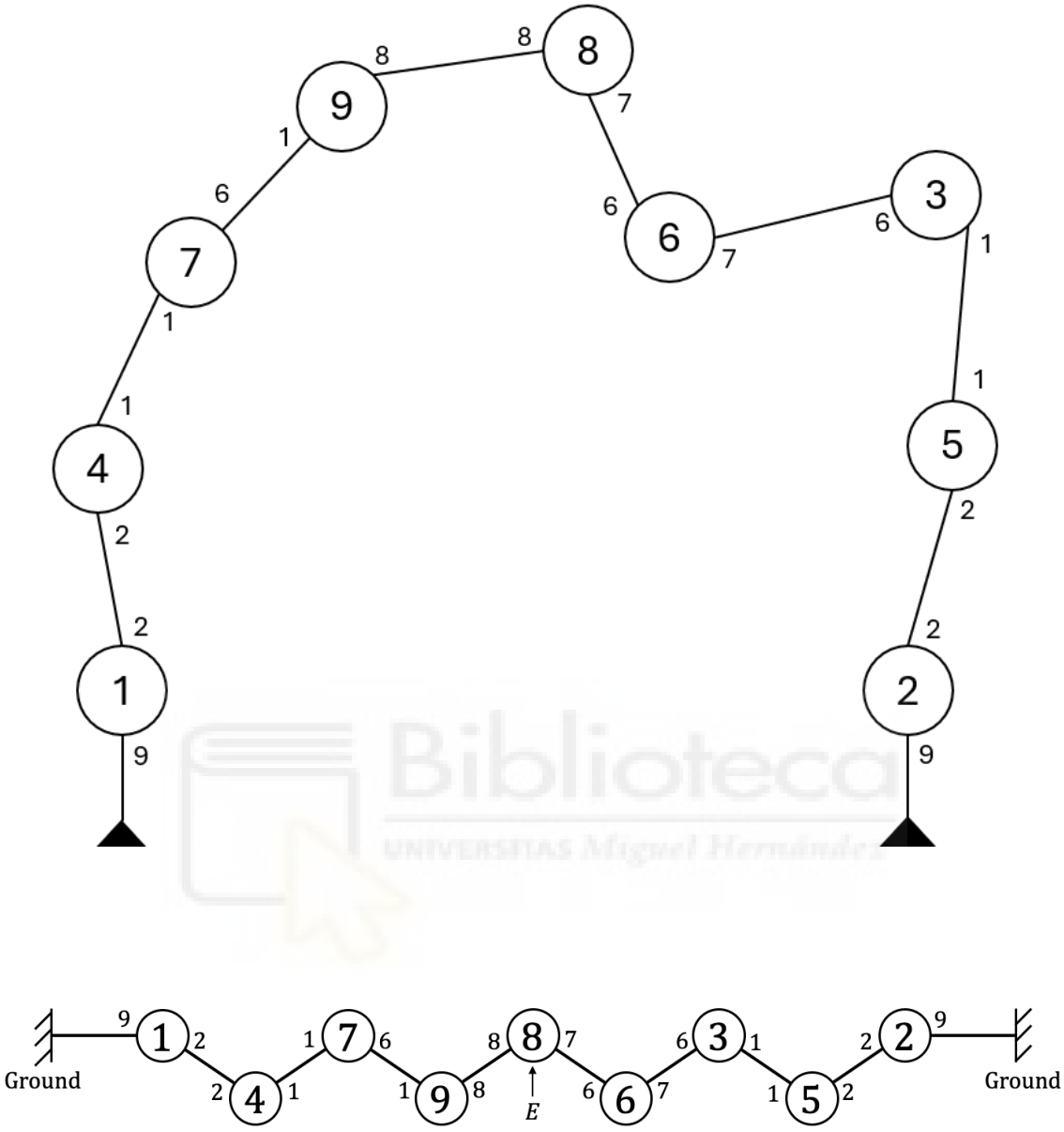

Figura 4: La conexiones entre módulos MASAR

2.9. PROGRAMACIÓN EN MATLAB

Los cálculos fueron realizados utilizando el programa MATLAB y código fuente detallado que se presenta a continuación corresponde al primer robot. Para el segundo robot, se puede encontrar el código análogo en los anexos. Mediante la creación de cinco funciones encargadas de proporcionar todos los resultados.

- El primer paso consiste en definir la función "f", que establece la geometría, el rango de variaciones aplicables al diseño y las ecuaciones de restricción.
- A continuación, se genera la matriz Jacobiana.
- Posteriormente, se crea un script para calcular las matrices de rotación en los ejes "x", "y", "z".
- Finalmente, se utiliza la función atan2, que permite calcular el ángulo conociendo el seno y el coseno.

La función "f" contiene toda la información relacionada con los diseños de los robots de 2 GDL y los cálculos necesarios para obtener las "f", llamando directamente a la función "f".

Al principio, "f" está vacía, y poco a poco se le van añadiendo las ecuaciones de restricción. Cada resultado se transforma en columnas, se acomoda con las demás y se va calculando todo en conjunto.

Al final, se añade una ecuación extra: el coseno al cuadrado más el seno al cuadrado, y luego menos 1. Así se asegura que se cumpla la identidad trigonométrica.

Cuando todo está listo, la función "f" termina con un total de 51 filas.

```
function salida = f(z,C,xy,desired_cos_theta1)

x = xy(1);
y = xy(2);
a = 1;
b = 0.1;
c = 0.2;
n = 2;
x1 = z(1);
```

```
y1 = z(2);
z1 = z(3);
cos_alpha1 = z(4);
sin_alpha1 = z(5);
cos_beta1 = z(6);
sin_beta1 = z(7);
cos_gamma1 = z(8);
sin_gamma1 = z(9);
cos_theta1 = z(10);
sin_theta1 = z(11);
x2 = z(12);
y2 = z(13);
z2 = z(14);
cos_alpha2 = z(15);
sin_alpha2 = z(16);
cos_beta2 = z(17);
sin_beta2 = z(18);
cos_gamma2 = z(19);
sin_gamma2 = z(20);
cos_theta2 = z(21);
sin_theta2 = z(22);
x3 = z(23);
y3 = z(24);
z3 = z(25);
cos_alpha3 = z(26);
sin_alpha3 = z(27);
cos_beta3 = z(28);
sin_beta3 = z(29);
cos_gamma3 = z(30);
sin_gamma3 = z(31);
cos_theta3 = z(32);
sin_theta3 = z(33);
```

Permite calcular el ángulo conociendo el seno y el coseno.

```
alpha1 = atan2cplx(sin_alpha1,cos_alpha1);
beta1 = atan2cplx(sin_beta1,cos_beta1);
gamma1 = atan2cplx(sin_gamma1,cos_gamma1);
theta1 = atan2cplx(sin_theta1,cos_theta1);
alpha2 = atan2cplx(sin_alpha2,cos_alpha2);
beta2 = atan2cplx(sin_beta2,cos_beta2);
gamma2 = atan2cplx(sin_gamma2,cos_gamma2);
theta2 = atan2cplx(sin_theta2,cos_theta2);
alpha3 = atan2cplx(sin_alpha3,cos_alpha3);
beta3 = atan2cplx(sin_beta3,cos_beta3);
gamma3 = atan2cplx(sin_gamma3,cos_gamma3);
theta3 = atan2cplx(sin_theta3,cos_theta3);
```

Ecuaciones de restricciones

```
f0 = [];
f0 = [ f0 ; cos_theta1 - desired_cos_theta1 ];
```

Matriz de transformación homogénea. Representa de forma compacta la posición y la orientación de un módulo.

```
T1 = [ rotx(alpha1)*roty(beta1)*rotz(gamma1) , [x1;y1;z1] ; [0 0 0 1]];
T2 = [ rotx(alpha2)*roty(beta2)*rotz(gamma2) , [x2;y2;z2] ; [0 0 0 1]];
T3 = [ rotx(alpha3)*roty(beta3)*rotz(gamma3) , [x3;y3;z3] ; [0 0 0 1]];
```

Ecuaciones de restricciones

```
ecuacion3 = [x;y]-([1 0 0 0;0 1 0 0]*T3*[[rotx(pi)*rotz((n*theta3))],[(a/2);0;-b];
[0,0,0,1]]*[0;0;0;1]);
f0 = [ f0 ; ecuacion3 ];

ecuacion4 =
T1*[[roty(pi/2)*rotz(theta1)],[((a/2)+c);0;0];[0,0,0,1]]*[[rotx(pi)],[0;0;0];[0,0,0,1
]]*[[rotz((-n*theta2))],[(-a/2);0;b];[0,0,0,1]]^(-1)*T2^(-1)- eye(4);
ecuacion4(end,:) = [];
f0 = [ f0 ; ecuacion4(:) ];

ecuacion5 =
T2*[[rotz((n*theta2))],[(a/2);0;b];[0,0,0,1]]*[[rotx(pi)],[0;0;0];[0,0,0,1]]*[[rotz((-
n*theta3))],[(-a/2);0;b];[0,0,0,1]]^(-1)*T3^(-1)- eye(4);
f0 = [ f0 ; ecuacion5(:) ];

ecuacion6 = (T1*[[roty((-pi/2))*rotz(-theta1)],[((-a/2)-c);0;0];[0,0,0,1]])-([[1 0
0;0 -1 0;0 0 -1],[0;0;0];[0,0,0,1]]);
ecuacion6(end,:) = [];
f0 = [ f0 ; ecuacion6(:) ];

fcs1 = [ cos_alpha1^2 + sin_alpha1^2 - 1 ; cos_beta1^2 + sin_beta1^2 - 1 ;
cos_gamma1^2 + sin_gamma1^2 - 1 ; cos_theta1^2 + sin_theta1^2 - 1 ];
fcs2 = [ cos_alpha2^2 + sin_alpha2^2 - 1 ; cos_beta2^2 + sin_beta2^2 - 1 ;
cos_gamma2^2 + sin_gamma2^2 - 1 ; cos_theta2^2 + sin_theta2^2 - 1 ];
fcs3 = [ cos_alpha3^2 + sin_alpha3^2 - 1 ; cos_beta3^2 + sin_beta3^2 - 1 ;
cos_gamma3^2 + sin_gamma3^2 - 1 ; cos_theta3^2 + sin_theta3^2 - 1 ];
f0 = [ f0 ; fcs1 ; fcs2 ; fcs3 ];

salida = C*f0;

end
```

El sistema se construye inicialmente con 51 ecuaciones: una correspondiente al ángulo θ_1, que al estar conectado al suelo no afecta al movimiento del efector final y únicamente rota el armazón central del primer módulo. Lo fijamos a un valor mediante la ecuación (cos_theta1 - desired_cos_theta1=0). A continuación, se añaden 2 ecuaciones que imponen la igualdad de las coordenadas (x, y) deseadas del efector final. Después, se generan 36 ecuaciones (3×12) correspondientes a las ecuaciones $4, 5\ y\ 6$: cada una de ellas surge de igualar dos matrices de rotación de tamaño 3×3 y un vector de traslación de tamaño 3×1, lo que equivale a 12 ecuaciones escalares por unión. Finalmente, se suman 12 ecuaciones trigonométricas derivadas de los cuatro ángulos $(\alpha, \beta, \gamma, \theta)$ en cada uno de los tres módulos. En total, esto da $1 + 2 + 36 + 12 = 51$ ecuaciones. Sin embargo, al analizar las igualdades de matrices de rotación, se observa que cada una aporta únicamente 3 ecuaciones independientes, ya que las columnas de una matriz de rotación son ortonormales (vectores unitarios y perpendiculares entre sí). Por tanto, en cada unión sobran 6 ecuaciones, lo que implica eliminar $6 \times 3 = 18$ redundancias. De este modo, el sistema queda reducido a $51 - 18 = 33$ ecuaciones independientes, coincidiendo exactamente con el número de incógnitas presentes.

En el segundo robot, el sistema se formula inicialmente con 159 ecuaciones: tres correspondientes a la posición deseada (x, y, z) del efector final. 120 derivadas de las diez uniones entre módulos, ya que cada unión implica comparar matrices de rotación de tamaño 3×3 y vectores de traslación de tamaño 3×1, lo que supone 12 ecuaciones escalares por unión. 36 ecuaciones adicionales provenientes de las igualdades trigonométricas de los cuatro ángulos $(\alpha, \beta, \gamma, \theta)$ en cada uno de los nueve módulos. En total, esto suma $3 + 120 + 36 = 159$ ecuaciones. Sin embargo, al analizar las igualdades de matrices de rotación, se observa que cada unión aporta únicamente seis ecuaciones redundantes, debido a la ortonormales de las columnas de la matriz de rotación (vectores unitarios y perpendiculares entre sí). Por tanto, en las diez uniones se eliminan $6 \times 10 = 60$ ecuaciones sobrantes. De este modo, el sistema queda reducido a $159 - 60 = 99$ ecuaciones independientes, coincidiendo exactamente con el número de incógnitas presentes en el problema.

En cuanto a la función J, esta función se encarga de estimar la Jacobiana por diferencias finitas. La Jacobiana está formada por columnas que son las derivadas de f respecto de cada una de las incógnitas, y esas derivadas las aproximamos por un pequeño incremento de f al variar cada incógnita de forma independiente.

Jacobiana "J":

```
function salida = J(z,C,xy,desired_cos_theta1)

Jf = [];
incremento = 0.00001;
        for i = 1 : size(z)
           dz = zeros(size(z));
           dz(i) = incremento;
           columna = f(z+dz,C,xy,desired_cos_theta1) - f(z,C,xy,desired_cos_theta1);
           Jf = [ Jf , columna ];

        End

Jf = Jf/incremento;
salida = Jf;

end
```

Se desarrolla un script destinado al cálculo de las matrices de rotación sobre los ejes "x", "y" y "z", con el fin de representar adecuadamente las transformaciones espaciales en el sistema de coordenadas.

Función de rotación en "x"	Función de rotación en "y"	Función de rotación en "z"

`function R = rotx(a)`
 	
`R = [ 1  0  0;`
`  0 cos(a) -sin(a);`
`  0 sin(a) cos(a)];`
 	
`end`
	`function R = roty(a)`
	
	`R = [cos(a)  0  sin(a);`
	`  0  1  0;`
	`  -sin(a)  0  cos(a)];`
	
	`end`
		`function R = rotz(a)`
		
		`R = [cos(a) -sin(a)  0;`
		`  sin(a) cos(a)  0;`
		`  0  0  1];`
		
		`end`

Finalmente, se utiliza la función atan2, que permite calcular el ángulo conociendo el seno y el coseno.

```
function val = atan2cplx( s , c )

    if norm(c-(-1)) < 0.00001
       val = pi;
    else
       val = 2*atan( s / (1+c) );

    end

end
```

El código fuente correspondiente a cada una de las funciones desarrolladas se incluirá como anexo al final del documento. Todo esto se ha desarrollado aquí con detalle para el primer robot estudiado (el de 3 módulos). Para el segundo robot, no se desarrolla aquí porque presenta una mayor extensión, y directamente se podrá encontrar en el anexo 6.2.

3. DESARROLLO DEL ESTUDIO Y APLICACIÓN DEL MÉTODO DE HOMOTOPÍA

3.1 INTRODUCCIÓN AL PROBLEMA

El presente trabajo se centra en la resolución del problema de cinemática inversa. Es decir, dada una posición deseada para el módulo final, resolver las incógnitas que permiten alcanzar dicha posición.

3.2 ENFOQUE INICIAL

El estudio arrancó con un enfoque práctico: se optó por primer robot de dos grados de libertad y tres módulos, lo que permitió reducir la complejidad de los cálculos y trabajar con un modelo más manejable; esa simplificación inicial resultó fundamental para validar la metodología y, sobre esa base sólida, escalar después el sistema hacia un robot con tres grados de libertad.

3.3 APLICACIÓN DEL MÉTODO DE HOMOTOPÍA

3.3.1 FUNDAMENTOS TEÓRICOS Y EXPLICACIÓN DEL MÉTODO

Queremos resolver la cinemática inversa, es decir, determinar todas las incógnitas (todas las variables indicadas en la ecuación (1), para todos los módulos), para lograr que el extremo del robot alcance una posición deseada. Si agrupamos las incógnitas en la variable z, el problema a resolver es $f(z) = 0$. Y, como primer intento, se probó a resolver esta ecuación mediante el método de homotopía.

Se aplica el método de homotopía, el cual permite transformar progresivamente un sistema conocido g, cuyas soluciones ya están determinadas, en el sistema deseado f, cuyas soluciones se desean calcular.

El proceso comienza con el sistema original de ecuaciones restricción:

$$\mathrm{f}(z_1, \dots, z_N) = \begin{bmatrix} f_1(z_1, \dots, z_N) \\ \vdots \\ f_N(z_1, \dots, z_N) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

A continuación, se construye un segundo sistema restricción, estructuralmente similar, pero con soluciones conocidas:

Estructuralmente similar quiere decir que el nuevo sistema mantiene la misma forma que el sistema original, en particular el grado de las ecuaciones. La manera más sencilla y la que hemos usado aquí es que las ecuaciones de g tengan el mismo grado que las del sistema original.

En el sistema original $f(z) = 0$, las ecuaciones son de grado 9. Por tanto, se construye $g(z) = 0$ también como un conjunto de ecuaciones de grado 9, de la siguiente manera:

$$z_1^9 - 1 = 0, z_2^9 - 1 = 0, \dots, z_{33}^9 - 1 = 0.$$

Segundo sistema de restricción:

$$\mathrm{g}(z_1, \dots, z_N) = \begin{bmatrix} g_1(z_1, \dots, z_N) \\ \vdots \\ g_N(z_1, \dots, z_N) \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

Con ambos sistemas definidos, se construye la función homotopía entre f y g:

$$H(z_1, \dots, z_N, t) = (1 - t) \cdot f(z_1, \dots, z_N) + t \cdot g(z_1, \dots, z_N) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

- Cuando $t = 1$, el sistema H equivale al sistema g, cuyas soluciones son conocidas.
- Cuando $t = 0$, el sistema H coincide con el sistema f, que es el que se desea resolver.

Se varía t de forma continua desde 1 hasta 0. Esto permite seguir las trayectorias de las soluciones conocidas de g hasta transformarlas progresivamente en las soluciones de f.

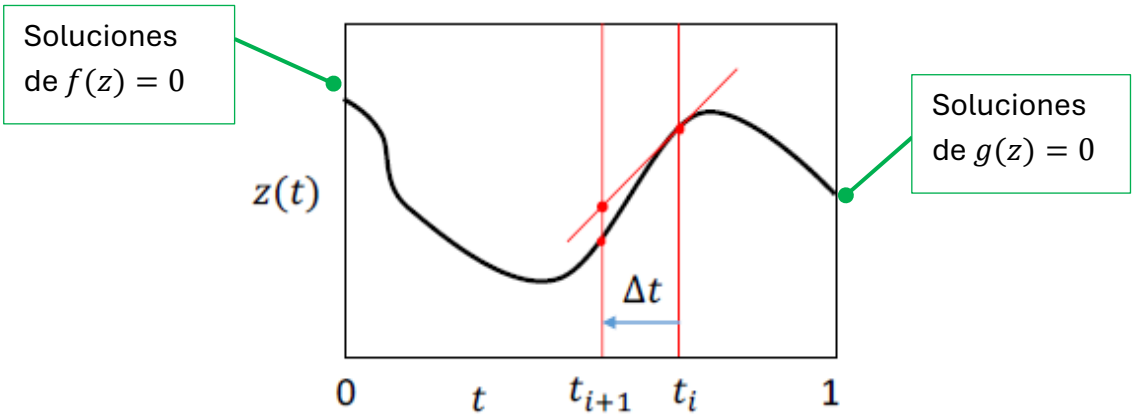

Figura 5: Trayectoria de soluciones de homotopía

Como se muestra en la figura anterior, la clave del método consiste en transformar el sistema inicial en uno más manejable y con soluciones conocidas, para luego aplicar homotopía y, mediante un seguimiento de trayectorias, obtener las soluciones deseadas del sistema original.

3.3.2 IMPLEMENTACIÓN NUMÉRICA

Inicialmente, se definió una función f, que contiene las ecuaciones de restricción del primer robot. Esta función depende de un vector z compuesto por 33 incógnitas y su objetivo es que todas las ecuaciones se igualen a cero.

Una vez definida f, se construyó una función g, estructuralmente similar, pero utilizando ecuaciones más sencillas, como polinomios de grado 9 y diseñadas específicamente para que sus soluciones sean conocidas.

El sistema se programó en MATLAB, implementando tanto la función homotopía H como su jacobiana. Durante el proceso, se actualizan iterativamente las variables del vector z en función de la derivada parcial H, aplicando un método de Euler y usa la corrección Newton mediante la inversa de la jacobiana multiplicada homotopía.

Este enfoque requiere gran precisión numérica, ya que pequeñas desviaciones pueden desviar significativamente la trayectoria de las soluciones esperadas.

3.4 LIMITACIONES DEL MÉTODO DE HOMOTOPÍA

El problema es lo siguiente, es bien sabido que, cuando se aplica el método de homotopía, solo algunas de las soluciones del sistema $g(z) = 0$ logran convertirse en verdaderas soluciones del sistema $f(z) = 0$. El resto de las soluciones de $g(z) = 0$ terminarán divergiendo a infinito, por lo que se tendrán que descartar. Ahora bien, en el caso del primer robot, el sistema $g(z) = 0$ presenta un número descomunal de soluciones, del orden de 10^{31}. En efecto, $g(z)$ se construye como: $z_1^9 - 1 = 0, z_2^9 - 1 = 0, \dots, z_{33}^9 - 1 = 0$, de modo que cada ecuación tiene 9 soluciones posibles. Al combinar todas las opciones para las 33 incógnitas, se obtiene: $9^{33} = 3 \cdot 10^{31}$.

Sin embargo, sistema deseado $f(z) = 0$ tendrá muy pocas soluciones (previsiblemente 2). Por tanto, resulta muy ineficiente hacer una búsqueda para ver cuáles de esas 10^{31} soluciones terminan convergiendo en las 2 soluciones verdaderas de la ecuación $f(z) = 0$. Por tanto, para sistemas con tantas ecuaciones e incógnitas como en nuestro robot, este enfoque no resulta nada práctico, y lo descartamos.

3.5 CONCLUSIÓN Y CAMBIO DE ENFOQUE

Dado que el método de homotopía no ofrecía una solución práctica, debido a su limitación para este tipo de robot en este caso concreto, se decidió abandonar esta vía. Como alternativa, se explorará un nuevo enfoque, basado en el uso de curvas de restricción, que se desarrollará en el siguiente capítulo.

4. CURVAS DE RESTRICCIÓN

4.1 INTRODUCCIÓN AL PROBLEMA

Para el primer robot queremos que $f(z) = 0$. Teníamos un sistema de 33 ecuaciones independientes, una vez descontadas las 6 ecuaciones dependientes que surgen de igualar dos matrices de rotación (en total se eliminan $6x3 = 18$ ecuaciones dependientes, porque hay 3 igualdades de matrices de rotación en las ecuaciones $4, 5\ y\ 6$) y 33 incógnitas; generamos un vector z aleatorio de ese tamaño que no cumple las ecuaciones y lo vamos refinando con el método de Newton, calculando la Jacobiana para refinarlo.

El problema del método de Newton es que depende del punto inicial: arrancando desde un punto podemos converger a una solución concreta; desde otro punto podemos converger a una solución distinta o incluso no converger. Eso dificulta sistematizar la búsqueda de todas las soluciones.

4.2 APLICACIÓN DEL MÉTODO DE CURVAS DE RESTRICCIÓN

Para facilitar la convergencia a alguna solución, en lugar de resolver las 33 ecuaciones a la vez, quitamos una y trabajamos con 32 ecuaciones y 33 incógnitas:

$$f_1(z_1, \dots, z_{33}) = 0$$

$$f_2(z_1, \dots, z_{33}) = 0$$

$$\vdots$$

$$f_{33}(z_1, \dots, z_{33}) = 0$$

Al eliminar una ecuación obtenemos un sistema subdeterminado con una incógnita libre: existe un conjunto infinito de soluciones que, gráficamente, forman una curva en el espacio de los tres ejes. Si fijamos un valor cualquiera para la incógnita sobrante y resolvemos las demás, obtenemos un punto sobre esa curva.

La idea práctica es en dos pasos. Primero, partiendo de un punto aleatorio, aplicamos Newton solo a las 32 ecuaciones hasta que la iteración nos lleve a la curva. Segundo, recorremos la curva con un método de continuación, avanzando por la dirección tangente y buscando los puntos donde la ecuación que quitamos se anula. Cuando esa función cambia de signo a lo largo de la curva sabemos que existe al menos otro punto donde es cero, la curva es cerrada y las soluciones suelen aparecer en pares. Si la función mantiene siempre el mismo signo en la curva, no hay soluciones y será necesario explorar otra curva.

Al desviarnos de la curva volvemos a usar Newton para reincorporarnos y así sucesivamente, hasta recorrer la curva y localizar las soluciones del sistema completo.

4.3 MÉTODO DE NEWTON Y CONTINUACIÓN: DESARROLLO EN MATLAB

Podemos quitar cualquier ecuación, salvo las últimas (las que obligan a que $cos\text{^}2 + sin\text{^}2 = 1$). Quitar estas provocarían resultados impredecibles e incorrectos, porque entraría en contradicción con las líneas de código donde calculamos los ángulos usando la función $atan2cplx$. Seguiremos trabajando con la misma función f definida anteriormente y con la Jacobiana. El método de Newton se aplicará partiendo de un vector "z" aleatorio, realizando iteraciones hasta alcanzar un punto sobre la curva de soluciones. A partir de ese punto, se procede a recorrer la curva mediante el método de continuación.

En lo que sigue, explicaremos el bloque de código en MATLAB que implementa este procedimiento.

4.3.1 ANÁLISIS DEL CÓDIGO DEL PRIMER ROBOT: CONVERGENCIA Y TRAZADO

Bloque inicial: tangencia de convergencia y trazos:

En la primera parte del código se establecen las bases necesarias para la ejecución del programa. Se definen las variables fundamentales que permitirán llevar a cabo los cálculos posteriores y se prepara el entorno de trabajo. En este bloque se implementa el ejemplo denominado $ejemplo1$, cuyas variables han sido obtenidas previamente mediante el método local utilizado en desarrollos previos del robot MASAR, como se ha comentado en la introducción de este TFG. Además, se invoca la función $adapt\_Y\_to\_z$. En este mismo bloque se fija el incremento que se utilizará en los cálculos iterativos, generación de un vector columna de 33 elementos aleatorios, que llamamos z_0 y se especifica la ecuación que se desea eliminar. Todo ello se organiza en un conjunto de variables que servirán como punto de partida para los bloques posteriores.

Bloque de convergencia hacia la curva:

La segunda sección del código está dedicada a la aproximación de la solución mediante el método de Newton. Este bloque constituye el núcleo del proceso de convergencia hacia la curva buscada.

- En primer lugar, se calculan las funciones iniciales F_0mediante la función "f".
- A continuación, se obtiene la matriz jacobiana J_0utilizando la función "J".
- Posteriormente, se elimina la ecuación seleccionada (eq_a_quitar) de $f0$ y de la jacobiana, con el fin de simplificar el sistema y centrarse en las variables relevantes.

En este bloque se incluye un procedimiento específico para eliminar ecuaciones redundantes. Este paso resulta fundamental, ya que asegura que el sistema de ecuaciones con el que se trabaja se mantenga consistente y únicamente compuesto por las ecuaciones independientes. De esta manera, se evita la duplicidad de información y se incrementa la eficiencia del cálculo. Finalmente,

la aproximación obtenida se almacena en la variable denominada curva, que servirá como referencia en los bloques posteriores.

Bloque de avance a lo largo de la curva:

La tercera parte del código se centra en el desplazamiento progresivo a lo largo de la curva previamente hallada. Para ello se implementa un bucle en el que se recalcula la jacobiana y se elimina nuevamente la ecuación seleccionada, manteniendo la coherencia con el procedimiento anterior. Este proceso permite identificar las soluciones en los puntos donde la función se anula, es decir, en aquellos valores en los que el resultado es igual a cero, garantizando así la consistencia del recorrido sobre la curva.

- En este bloque se eliminan las ecuaciones redundantes nuevamente.

- Las soluciones obtenidas deben presentarse en pares, lo que garantiza la simetría del sistema y permite el cierre adecuado de las curvas.

- En este bloque se define un bucle que invierte el sentido del vector tangente cuando este apunta hacia atrás, garantizando que el avance se realice siempre en la dirección correcta.

- A continuación, si el desplazamiento se desvía de la curva, se aplica nuevamente el método de Newton para reconducir la trayectoria y mantener la precisión del cálculo.

El resultado de este proceso se almacena en la variable curva, que se va actualizando conforme el sistema avanza a lo largo de la trayectoria.

Bloque de visualización:

Una vez obtenida la curva, se procede a su representación gráfica mediante la función plot. Este bloque permite visualizar de manera clara y diferenciada las distintas fases del proceso:

- La aproximación inicial se muestra en color rojo, lo que facilita identificar el recorrido previo hasta alcanzar la curva.

- La curva final se representa en color azul, destacando el resultado definitivo del cálculo.

Aunque el espacio de representación es de dimensión 33, se proyecta en un espacio tridimensional para hacerlo comprensible y visualizable. Esta proyección permite interpretar la curva como un anillo, lo que constituye una simplificación necesaria para poder analizarla en nuestro entorno tridimensional.

Bloque de animación:

Finalmente, se implementa un bloque destinado a la animación del movimiento a lo largo de la curva. Este componente resulta especialmente relevante, ya que permite visualizar de manera dinámica la evolución de la trayectoria y los puntos en los que la función se anula. La representación animada ofrece una perspectiva más completa del proceso, pues no solo muestra el resultado final, sino también el recorrido realizado sobre las curvas cerradas. De este modo, se obtiene una visión más intuitiva y detallada del comportamiento del sistema, facilitando tanto la interpretación matemática como la comprensión geométrica.

El código fuente correspondiente a cada una de las funciones desarrolladas se adjuntará como anexo al final del documento de la memoria, con el fin de facilitar su consulta y respaldar la implementación descrita en el cuerpo principal del trabajo.

4.3.2 ANÁLISIS DEL CÓDIGO DEL SEGUNDO ROBOT:

El segundo robot está constituido por nueve módulos interconectados. Los módulos 1 y 2 permanecen fijados al suelo, actuando como elementos de referencia y soporte estructural.

El análisis del código se organiza en dos scripts principales:

Función f

- En este apartado se define el diseño del robot, especificando la finalidad del sistema y su estructura modular.

- Se establecen las ecuaciones de restricción que regulan el comportamiento de los módulos, garantizando que el movimiento se ajuste a las condiciones físicas y geométricas impuestas.
- Dichas restricciones permiten delimitar el espacio de soluciones posibles y asegurar la coherencia del modelo matemático.

Función de convergencia_trazado

- En este bloque se implementa el método de Newton, utilizado para aproximarse de manera iterativa a la curva objetivo.
- Una vez alcanzada la convergencia, el sistema se desplaza siguiendo el trazado de la curva, lo que permite localizar los puntos nulos.

4.3.3 DEFINICIÓN DE LA FUNCIÓN f PARA EL SEGUNDO ROBOT

La nueva función Fcorrespondiente al segundo robot se construye siguiendo una serie de pasos metodológicos que permiten formalizar su comportamiento:

Bloque de definición de variables y dimensiones:

- Se establecen un total de 99 variables, que representan las magnitudes necesarias para describir el sistema.
- Estas variables incluyen parámetros geométricos y dimensiones relevantes para la caracterización de los módulos.

Bloque atan2cplx:

- El bloque atan2cplx se utiliza para calcular el ángulo a partir de las componentes seno y coseno.

Bloque de construcción de las matrices de rotación:

- Se generan las matrices de rotación correspondientes a los nueve módulos del robot.

- Dichas matrices permiten describir la orientación espacial de cada módulo y su relación con los demás.

Bloque de ecuaciones de restricción:

- Se definen las ecuaciones de restricción que regulan la unión entre módulos, dependiendo de la pata o punto de conexión utilizado.
- Se obtienen 11 ecuaciones de restricción que aseguran la estabilidad del sistema.
- El punto de control se establece en el centro de coordenadas del módulo 8, que actúa como efector final de referencia para el análisis del movimiento.

Bloque de función matemática de consistencia angular

- Finalmente, se incorpora la condición trigonométrica fundamental:

$$\sin^2(\theta) + \cos^2(\theta) = 1$$

- Aplicada a los tres ángulos principales del sistema: α, β, γ y θ.
- Esta relación asegura la coherencia matemática en la definición de las rotaciones y garantiza la validez del modelo.

4.3.4 DEFINICIÓN CONVERGENCIA_Y_TRAZADO PARA EL SEGUNDO ROBOT

El código correspondiente al segundo robot mantiene una estructura similar al desarrollado para el primer robot, aunque incorpora ciertas modificaciones específicas que permiten adaptar el modelo a la nueva configuración.

Se redefine la variable aleatoria inicial z_0, que pasa a ser una columna de 99 elementos. ajustándose al número de variables previamente establecidas en la función f. Además, la variable aleatoria z_0, se multiplica por 10, lo que permite acelerar el proceso de búsqueda de la solución. El resto del código se conserva sin cambios sustanciales respecto al modelo anterior.

Posteriormente, se introduce una nueva variable denominada xy, definida como una matriz de tres componentes (x, y, z). En esta matriz, los valores de (x, y) se modifican de manera controlada, mientras que la coordenada z se mantiene fija en z = 2. Esta configuración representa el movimiento del centro del efector final del robot, permitiendo analizar su desplazamiento en el plano horizontal.

Finalmente**,** al variar los valores de (x, y) se observa cómo el sistema converge hacia la curva objetivo. La modificación progresiva de estas componentes genera diferentes curvas al que se convergencia, que describen las trayectorias posibles del robot. El análisis permite identificar los puntos en los que la función se anula sobre la curva, lo que constituye un criterio fundamental para validar la estabilidad y coherencia del modelo.

4.3.5 ANÁLISIS GRÁFICO TRAS LA VARIACIÓN DE X E Y

La gráfica muestra el comportamiento del sistema al evaluar la función el punto donde se desea llevar el robot en la posición $(0{,}0{,}2)$, correspondiente al centro del módulo 8. En este caso, se obtienen dos soluciones, lo cual es coherente con la naturaleza par del sistema, que permite múltiples configuraciones válidas bajo las mismas condiciones.

El proceso representado en la gráfica se desarrolla en dos etapas consecutivas:

Convergencia mediante el método de Newton

- En primer lugar, se aplica el método de Newton para aproximarse a la curva solución.
- Este método parte de una condición inicial aleatoria y, mediante iteraciones sucesivas, converge hacia un punto que satisface las ecuaciones de restricción del sistema.
- En la gráfica, esta trayectoria se representa en color rojo, indicando el camino seguido hasta alcanzar la solución.

Marcha a lo largo de la curva

- Una vez alcanzado el punto de convergencia, se aplica el método de marcha a lo largo de la curva, que permite recorrer la solución de forma continua.
- Este método genera una trayectoria que sigue la geometría de la curva solución, facilitando el análisis del comportamiento global del sistema.
- En la gráfica, esta trayectoria se representa en color azul.

Además, los puntos en negro indican las posiciones en las que la función se anula, es decir, donde se cumple la condición de solución. Estos puntos validan la efectividad de ambos métodos y confirman la existencia de múltiples soluciones en el sistema.

Esta representación gráfica permite visualizar claramente el flujo del algoritmo: primero se alcanza la curva mediante convergencia, y posteriormente se recorre dicha curva para analizar su comportamiento global.

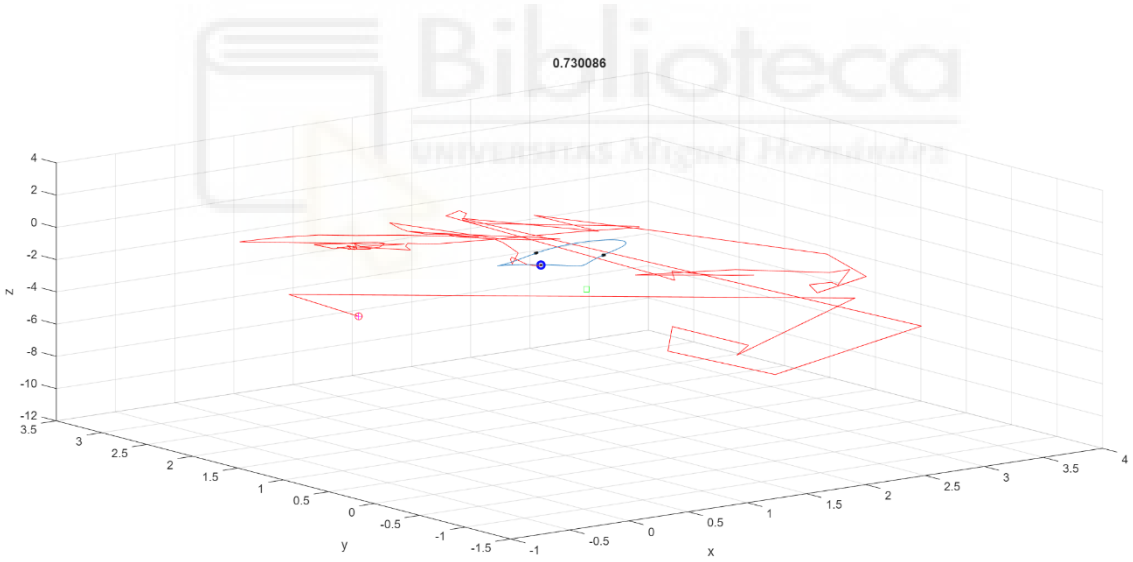

Figura 6: representa al segundo robot ubicado en la coordenada (0, 0, 2)

A continuación, se presentan las gráficas correspondientes a las diferentes posiciones deseadas. Estas se han obtenido a partir de la Figura 6, la cual representa el movimiento que puede realizar el centro del módulo 8 en el efecto final. Las gráficas resultan de variar los valores de x e y, manteniendo constante la coordenada z

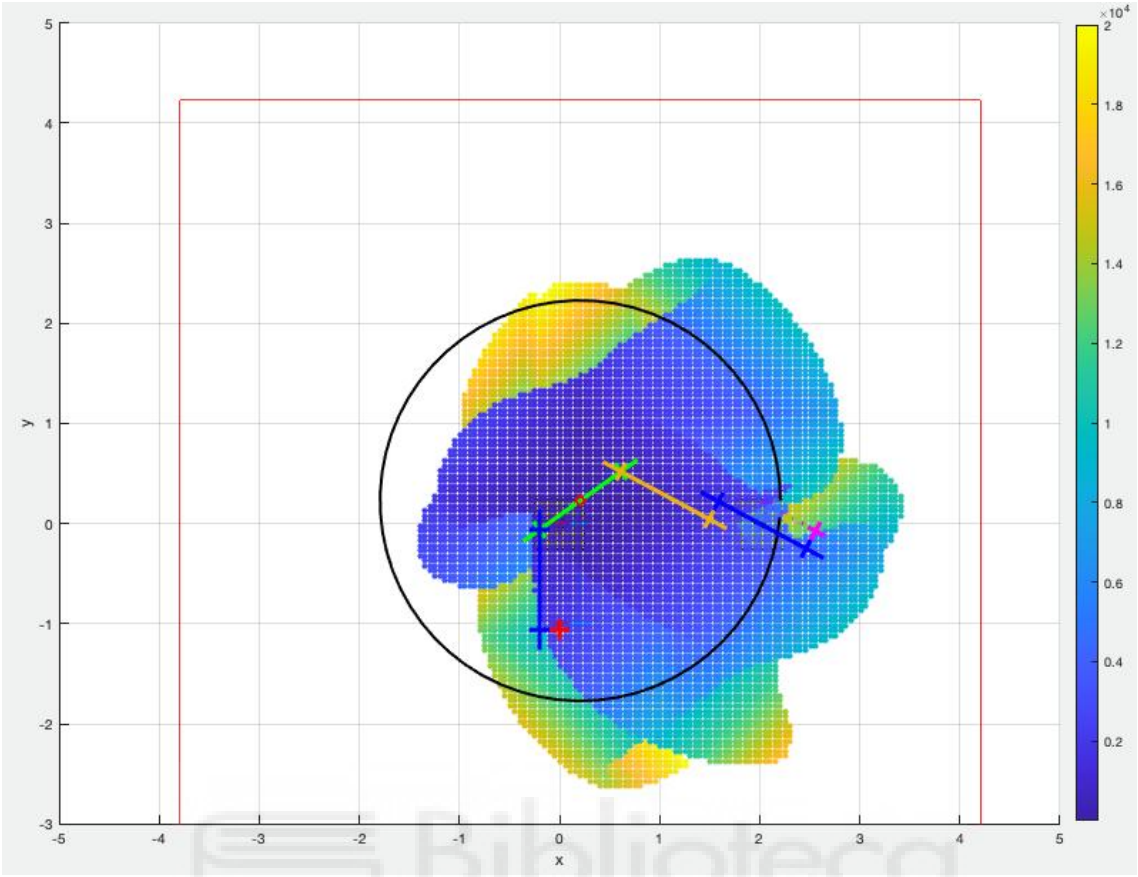

Figura 7: Trayectorias posibles del centro de coordenadas en el módulo 8

A continuación, se presentan las gráficas correspondientes a las diferentes posiciones deseadas, Que se tomaron de la figura 6 que representa el movimiento que puede El centro del módulo 8 hacer del efecto final. Obtenidas al variar los valores de x e y mientras se mantiene constante la coordenada z.

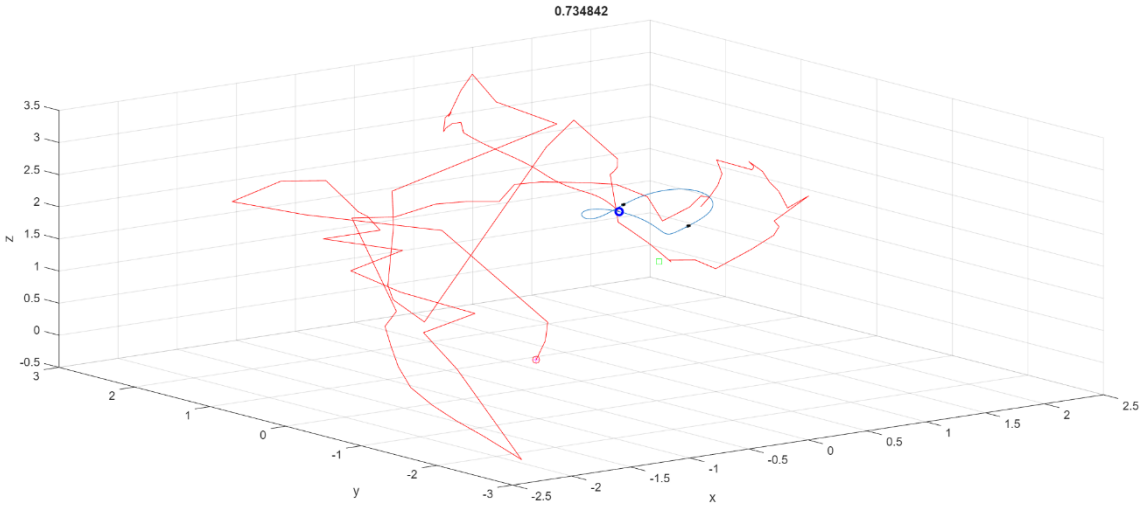

Figura 8: representa al segundo robot ubicado en la coordenada (1, 0, 2)

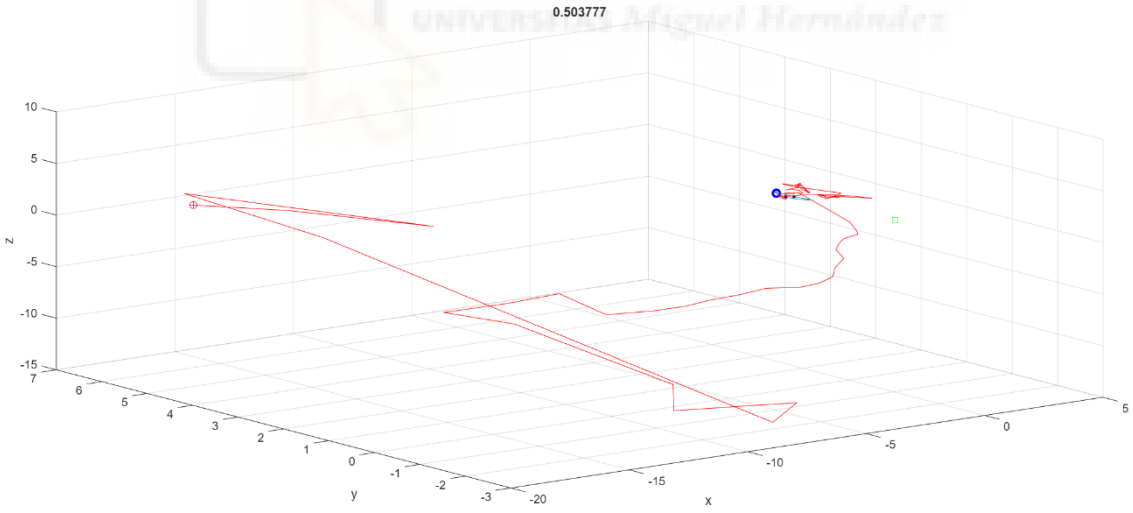

Figura 9: representa al segundo robot ubicado en la coordenada (0, 1, 2)

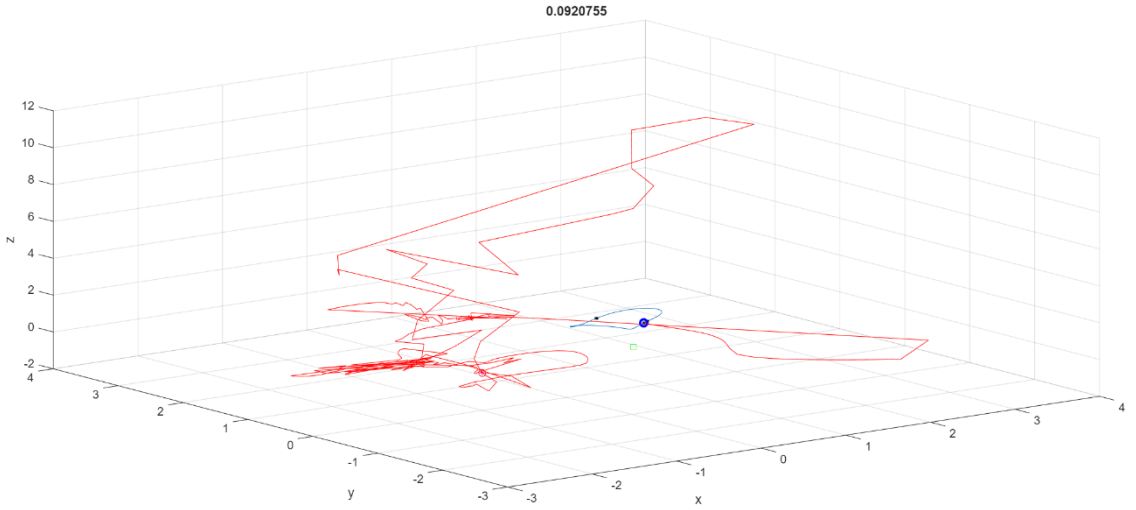

Figura 10: representa al segundo robot ubicado en la coordenada (1, 1, 2)

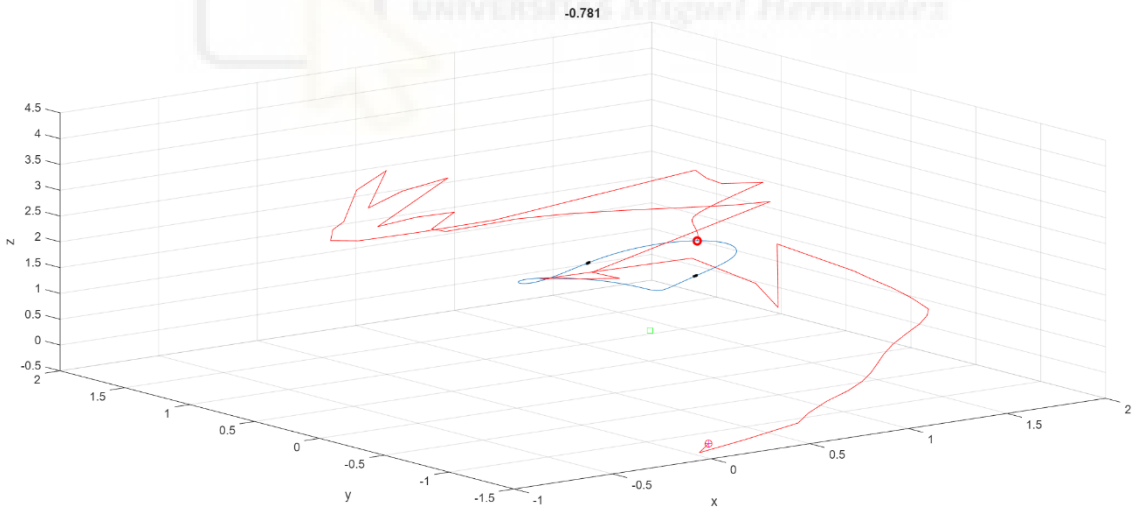

Figura 11: representa al segundo robot ubicado en la coordenada (1, 2, 2)

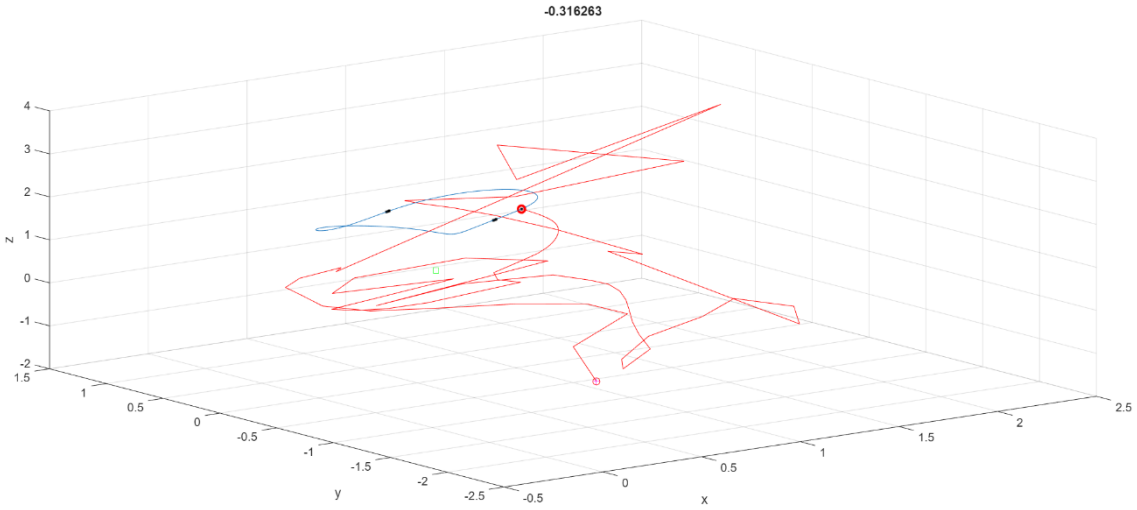

Figura 12: representa al segundo robot ubicado en la coordenada (2, 0, 2)

En resumen, después de revisar las soluciones anteriores, todavía queda por ver la última variación del método. Se trata de cambiar la ecuación que se elimina, partiendo de una curva ya conocida, para comprobar si puede trazarse otra curva que incluya nuevas soluciones. Este punto se explicará en el próximo apartado.

4.4 EXPLORACIÓN DE SOLUCIONES ALTERNATIVAS DENTRO DE LA CURVA

4.4.1. INTRODUCCIÓN

El propósito de este apartado es explorar soluciones alternativas dentro de una curva previamente obtenida. La estrategia consiste en modificar la ecuación que se elimina, partiendo de una curva ya conocida, con el fin de comprobar si es posible generar otra curva que incorpore nuevas soluciones. De esta manera, se amplía el análisis y se investiga la existencia de trayectorias adicionales que complementen las soluciones identificadas en el estudio inicial.

El código desarrollado y explicado en este apartado se encuentra en el anexo 6.2.4.

4.4.2. DESARROLLO DEL PROCESO

Selección de la curva inicial: Se parte de la curva mostrada en la Figura 9, en la cual se identifican dos soluciones.

Asignación de ejes (Sección 3): Se ejecuta la sección correspondiente para representar la curva de la Figura 9, destacada en color azul.

Localización de la primera solución (Sección 4): Se determina el primer punto donde la función se anula, lo cual ocurre cuando $f_0(eq\_a\_quitar)$ cambia de signo.

Refinamiento de la solución (Sección 5): Como el valor inicial no es exactamente cero, se añaden este bloque que ajusta el cálculo hasta lograr la anulación precisa de la función.

Modificación de variables (Sección 2): Se cambia la ecuación que elimina, se ejecuta nuevamente el proceso y se observa cuándo se produce el cambio de signo. El procedimiento se mantiene en ejecución hasta que se decida interrumpirlo, momento en el que se guarda la variable obtenida (por ejemplo, $prueba12\_eq\_38$).

Consideraciones finales

No todas las ecuaciones que se eliminan generan soluciones útiles:

- En algunos casos no aparecen nuevas soluciones.
- En otros, se repiten los mismos puntos ya encontrados en la curva original.

Por lo tanto, no todas las ecuaciones resultan productivas para ampliar el conjunto de soluciones.

4.4.3. REPRESENTACIÓN GRÁFICA DE LOS RESULTADOS

Finalmente, se incluyen las gráficas correspondientes, donde se observan claramente los resultados obtenidos.

En primer lugar, se presentan las gráficas correspondientes, en las que se muestran las soluciones obtenidas en función de la ecuación que se elimina. Tal como puede observarse en las Figuras 13 y 14, aparecen curvas adicionales que permiten identificar nuevas soluciones. Estas curvas amplían el conjunto de resultados y proporcionan un número mayor de alternativas respecto a las soluciones iniciales.

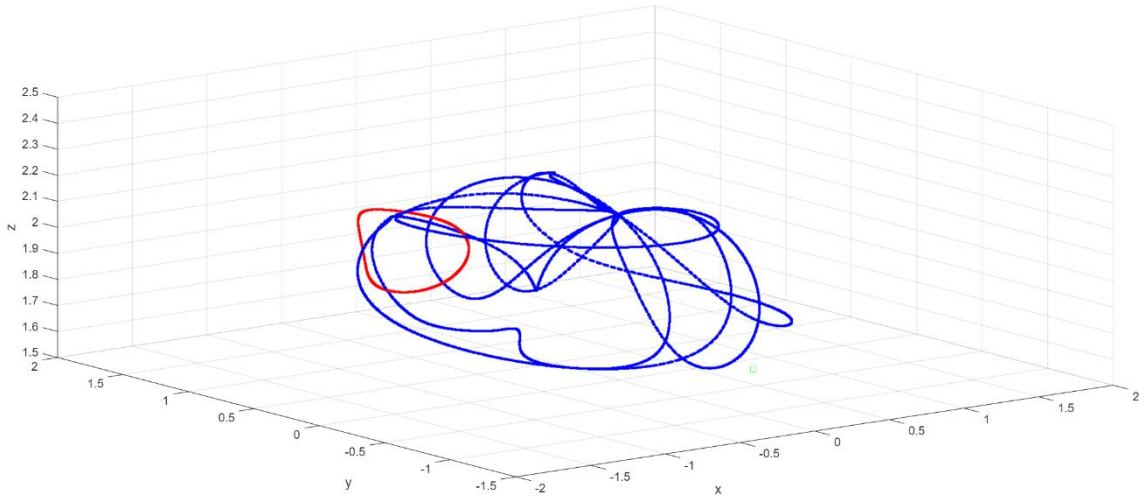

Figura 13: Nueva curva (azul) que se genera a partir de la original (roja) cuando se suprime la componente 38 del vector de funciones f.

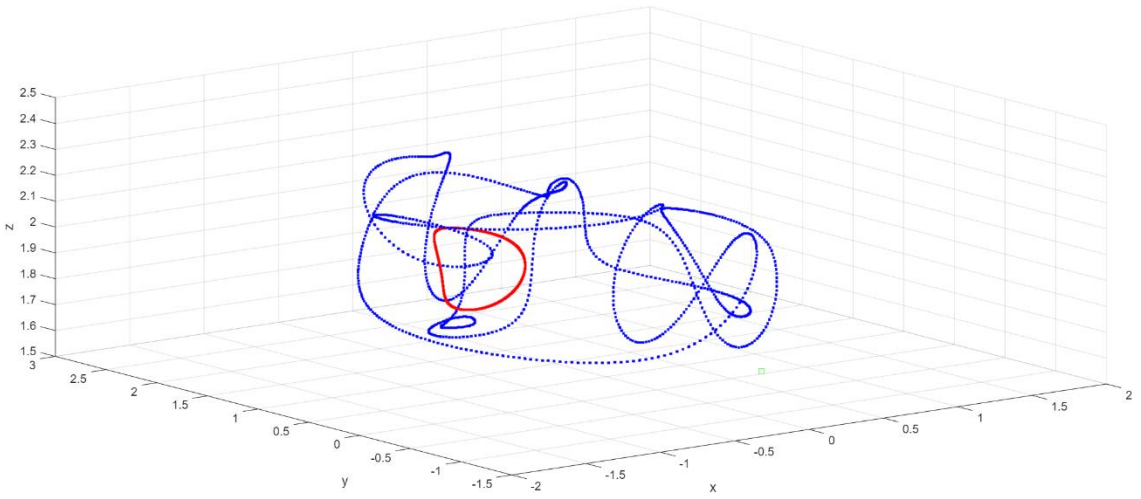

Figura 14: Nueva curva (azul) que se genera a partir de la original (roja) cuando se suprime la componente 110 del vector de funciones f.

En las Figuras 15 y 16 se observa el caso previamente comentado, en el cual aparecen nuevas curvas al modificar la ecuación que se elimina. Sin embargo, dichas curvas atraviesan exactamente los mismos puntos que ya habían sido identificados en la curva original. Por lo tanto, aunque se generan representaciones adicionales, no se obtienen soluciones nuevas y el conjunto de resultados permanece sin cambios.

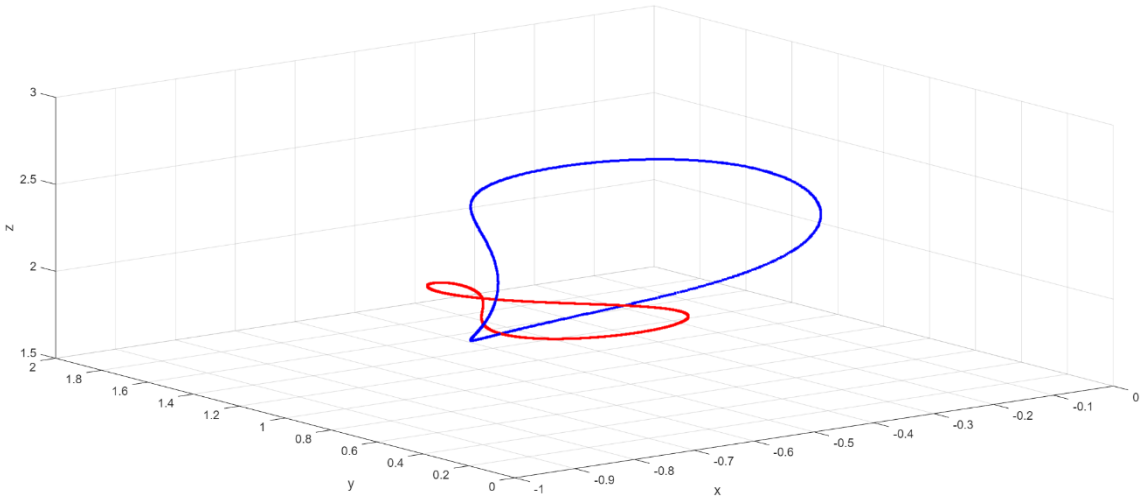

Figura 15: Nueva curva (azul) que se genera a partir de la original (roja) cuando se suprime la componente 27 del vector de funciones f.

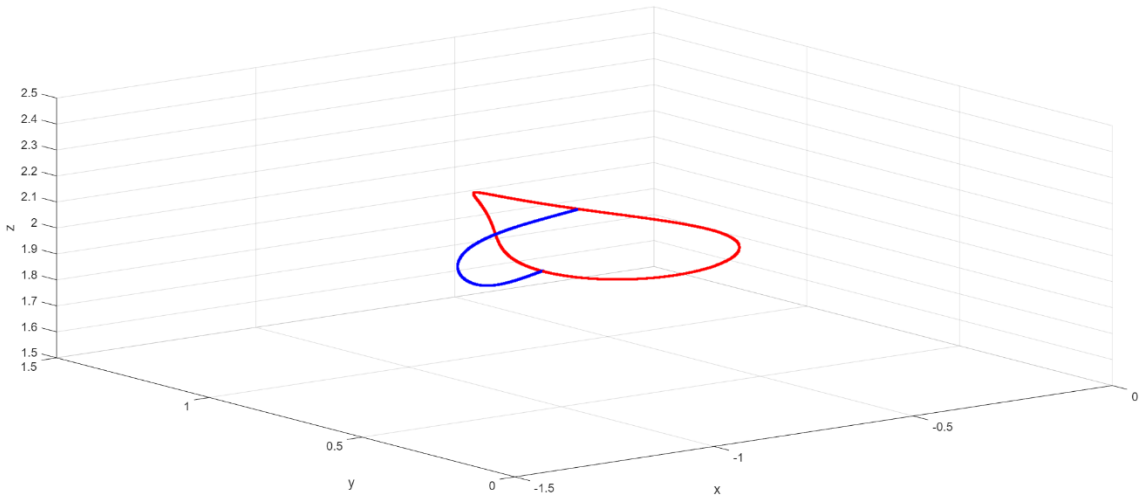

Figura 16: Nueva curva (azul) que se genera a partir de la original (roja) cuando se suprime la componente 63 del vector de funciones f.

En las Figuras 17 y 18 se observa el primer caso mencionado. Al modificar la ecuación que se elimina, no aparecen nuevas soluciones: el proceso únicamente se ejecuta en el mismo punto inicial y permanece allí sin desplazarse hacia otros valores. En consecuencia, estas curvas no aportan información adicional, el procedimiento falla en este escenario y no resulta útil para ampliar el conjunto de soluciones.

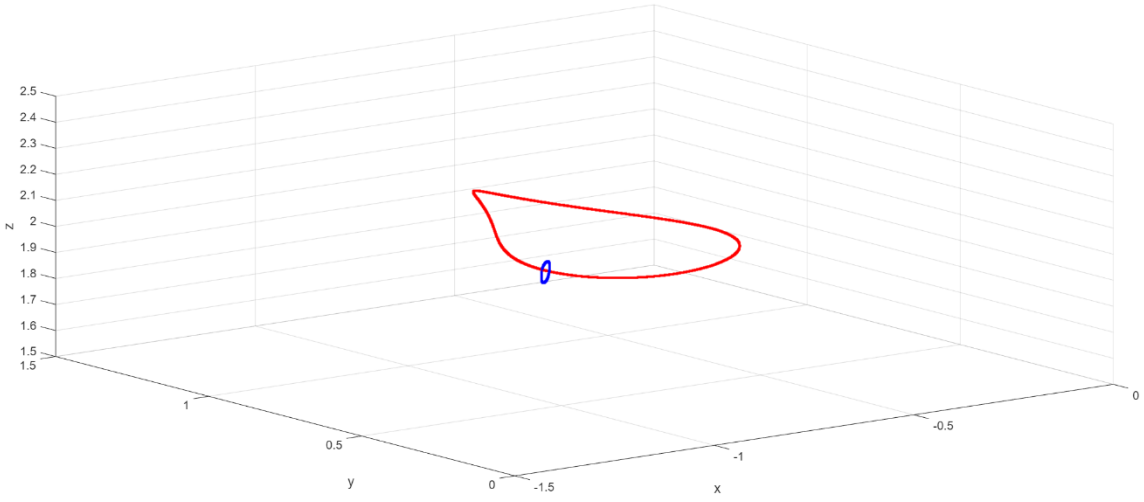

Figura 17: Nueva curva (azul) que se genera a partir de la original (roja) cuando se suprime la componente 7 del vector de funciones f.

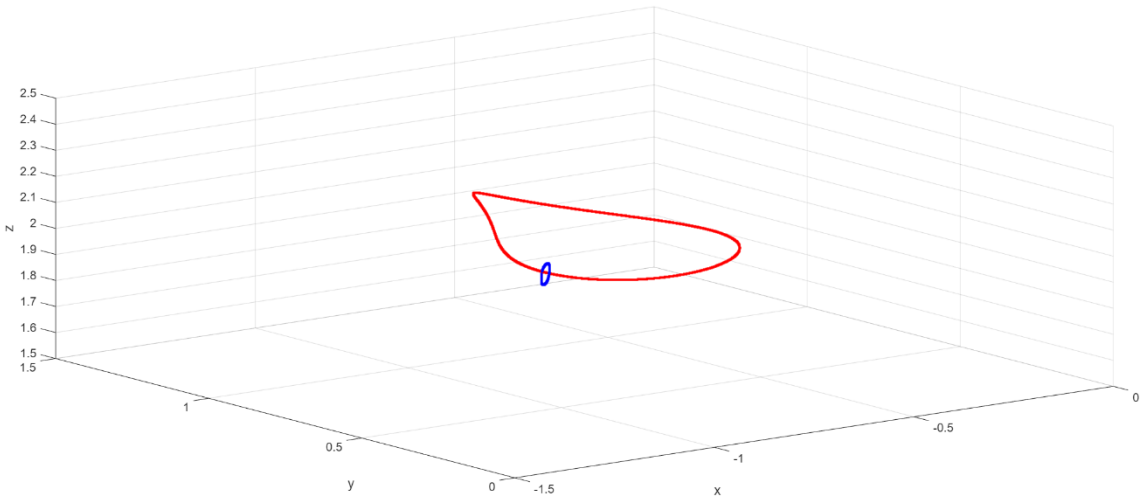

Figura 18: Nueva curva (azul) que se genera a partir de la original (roja) cuando se suprime la componente 80 del vector de funciones f.

5. CONCLUSIÓN

En este Trabajo de Fin de Grado se ha abordado la resolución de la cinemática inversa en robots modulares, con el propósito de identificar múltiples soluciones y analizar la viabilidad de distintos métodos. Inicialmente se consideró la aplicación de la homotopía como estrategia de resolución; sin embargo, se comprobó que este enfoque no resulta práctico en sistemas de gran complejidad. El número desproporcionado de soluciones generadas por el sistema auxiliar, frente al reducido número de soluciones reales del sistema objetivo, hacía inviable su aplicación, ya que la mayoría de las trayectorias divergían y no conducían a resultados útiles.

Ante esta limitación, se optó por un método más accesible basado en la resolución directa de la cinemática inversa. Este procedimiento permitió encontrar varias soluciones válidas para el primer robot y, tras el desarrollo del código, se escaló con éxito al segundo robot, confirmando también la existencia de múltiples soluciones. Aunque el método empleado no garantiza la obtención de todas las soluciones posibles, sí ha demostrado ser una herramienta eficaz para ampliar el conjunto de resultados y avanzar en el análisis de la cinemática inversa en robots modulares.

En definitiva, el trabajo ha puesto de manifiesto tanto las limitaciones del enfoque homotopía como la utilidad del método de cinemática inversa, ofreciendo una base sólida para futuras investigaciones. El perfeccionamiento del procedimiento y la incorporación de modificaciones complementarias constituirán líneas de trabajo necesarias para garantizar más soluciones posibles para robots modulares.

6.ANEXO

6.1. CÓDIGO DE LAS FUNCIONES EN MATLAB PARA EL PRIMER ROBOT

6.1.1 FUNCIÓN f:

```
function salida = f(z,C,xy,desired_cos_theta1)

x = xy(1);
y = xy(2);
a = 1;
b = 0.1;
c = 0.2;
n = 2; % relacion de reduccion (1 vuelta de theta genera 2 vueltas del pad)
x1 = z(1);
y1 = z(2);
z1 = z(3);
cos_alpha1 = z(4);
sin_alpha1 = z(5);
cos_beta1 = z(6);
sin_beta1 = z(7);
cos_gamma1 = z(8);
sin_gamma1 = z(9);
cos_theta1 = z(10);
sin_theta1 = z(11);
x2 = z(12);
y2 = z(13);
z2 = z(14);
cos_alpha2 = z(15);
sin_alpha2 = z(16);
cos_beta2 = z(17);
sin_beta2 = z(18);
cos_gamma2 = z(19);
sin_gamma2 = z(20);
cos_theta2 = z(21);
sin_theta2 = z(22);
x3 = z(23);
y3 = z(24);
z3 = z(25);
cos_alpha3 = z(26);
sin_alpha3 = z(27);
cos_beta3 = z(28);
```

```
sin_beta3 = z(29);
cos_gamma3 = z(30);
sin_gamma3 = z(31);
cos_theta3 = z(32);
sin_theta3 = z(33);

alpha1 = atan2cplx(sin_alpha1,cos_alpha1);
beta1 = atan2cplx(sin_beta1,cos_beta1);
gamma1 = atan2cplx(sin_gamma1,cos_gamma1);
theta1 = atan2cplx(sin_theta1,cos_theta1);
alpha2 = atan2cplx(sin_alpha2,cos_alpha2);
beta2 = atan2cplx(sin_beta2,cos_beta2);
gamma2 = atan2cplx(sin_gamma2,cos_gamma2);
theta2 = atan2cplx(sin_theta2,cos_theta2);
alpha3 = atan2cplx(sin_alpha3,cos_alpha3);
beta3 = atan2cplx(sin_beta3,cos_beta3);
gamma3 = atan2cplx(sin_gamma3,cos_gamma3);
theta3 = atan2cplx(sin_theta3,cos_theta3);

f0 = [];
f0 = [ f0 ; cos_theta1 - desired_cos_theta1 ];

% Matrices de rotación
T1 = [ rotx(alpha1)*roty(beta1)*rotz(gamma1) , [x1;y1;z1] ; [0 0 0 1]];
T2 = [ rotx(alpha2)*roty(beta2)*rotz(gamma2) , [x2;y2;z2] ; [0 0 0 1]];
T3 = [ rotx(alpha3)*roty(beta3)*rotz(gamma3) , [x3;y3;z3] ; [0 0 0 1]];

% Ecuaciones de restricción de robot
ecuacion3 = [x;y]-([1 0 0 0;0 1 0 0]*T3*[[rotx(pi)*rotz((n*theta3))],[(a/2);0;-
b];[0,0,0,1]]*[0;0;0;1]);
f0 = [ f0 ; ecuacion3(:) ];

ecuacion4 =
T1*[[roty(pi/2)*rotz(theta1)],[((a/2)+c);0;0];[0,0,0,1]]*[[rotx(pi)],[0;0;0];[0,0,
0,1]]*[[rotz((-n*theta2))],[(-a/2);0;b];[0,0,0,1]]^(-1)*T2^(-1)- eye(4);
ecuacion4(end,:) = [];
f0 = [ f0 ; ecuacion4(:)];

ecuacion5 =
T2*[[rotz((n*theta2))],[(a/2);0;b];[0,0,0,1]]*[[rotx(pi)],[0;0;0];[0,0,0,1]]*[[rotz(
(-n*theta3))],[(-a/2);0;b];[0,0,0,1]]^(-1)*T3^(-1)- eye(4);
ecuacion5(end,:) = [];
f0 = [ f0 ; ecuacion5(:) ];
```

```
ecuacion6 = (T1*[[roty((-pi/2))*rotz(-theta1)],[((-a/2)-c);0;0];[0,0,0,1]])-([[1
0 0;0 -1 0;0 0 -1],[0;0;0];[0,0,0,1]]);
ecuacion6(end,:) = [];
f0 = [ f0 ; ecuacion6(:) ];

fcs1 = [ cos_alpha1^2 + sin_alpha1^2 - 1 ; cos_beta1^2 + sin_beta1^2 - 1 ;
cos_gamma1^2 + sin_gamma1^2 - 1 ; cos_theta1^2 + sin_theta1^2 - 1 ];
fcs2 = [ cos_alpha2^2 + sin_alpha2^2 - 1 ; cos_beta2^2 + sin_beta2^2 - 1 ;
cos_gamma2^2 + sin_gamma2^2 - 1 ; cos_theta2^2 + sin_theta2^2 - 1 ];
fcs3 = [ cos_alpha3^2 + sin_alpha3^2 - 1 ; cos_beta3^2 + sin_beta3^2 - 1 ;
cos_gamma3^2 + sin_gamma3^2 - 1 ; cos_theta3^2 + sin_theta3^2 - 1 ];
f0 = [ f0 ; fcs1 ; fcs2 ; fcs3 ];

salida = C*f0;

end
```

6.1.2. FUNCIÓN *J* (JACOBIANA):

```
function salida = J(z,C,xy,desired_cos_theta1)

Jf = [];

incremento = 0.00001;
for i = 1 : size(z)
   dz = zeros(size(z));
   dz(i) = incremento;
   columna = f(z+dz,C,xy,desired_cos_theta1) - f(z,C,xy,desired_cos_theta1);
   Jf = [ Jf , columna ];

end
Jf = Jf/incremento;
salida = Jf;

end
```

6.1.3. CONVERGENCIA_Y_TRAZADO:

```
% Primero converger en la curva
load ejemplo1
adapt_Y_to_z

incremento = 0.00001;
z0 = randn(33,1);
z = z0;
acercamiento = z';
eq_a_quitar = 26;
rango_forzado = 32;
cr = 0.1;
for i = 1 : 1000
   f0 = f(z,C,xy,desired_cos_theta1);
   f0(eq_a_quitar,:) = [];
   J0 = J(z,C,xy,desired_cos_theta1); % calcular jacobiana usando funcion
   J0(eq_a_quitar,:) = [];
   dz = -cr*pinv(J0)*f0;

% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
% Descomposición en valores singulares (SVD)
   A = J0;
   b = -f0;
   [U, S, V] = svd(A, 'econ');

% el rango efectivo
   tol = max(size(A)) * eps(max(diag(S)));
   tol = 1e-5;
   rank_A = sum(diag(S) > tol);
   rank_A = rango_forzado;
% Seleccionar ecuaciones independientes
   indepRows = 1:rank_A;
   A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
   b_red = U(:, indepRows)' * b;
% Obtener solución de mínima norma
   dz = lsqminnorm(A_red, b_red);
% FIN DEL BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES

   z = z + dz*cr;
   if norm(dz)<0.00001
      break
   end
```

```
    acercamiento = [ acercamiento ; z' ];
end
i_converger_a_curva = i;
norm(f0)
z(12:14)
cond(J0)
f_completa = f(z,C,xy,desired_cos_theta1)
norm(f_completa)

curva = z';

%% Marchar a lo largo de la curva
cr = 1;
for i = 1 : 500

    J0 = J(z,C,xy,desired_cos_theta1); % calcular jacobiana
    J0(eq_a_quitar,:) = []; % Quitamos primera fila

%%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
    A = J0;
    [U, S, V] = svd(A, 'econ');
% el rango efectivo
    tol = max(size(A)) * eps(max(diag(S)));
    tol = 1e-5;
    rank_A = sum(diag(S) > tol);
    rank_A = rango_forzado;
% Seleccionar ecuaciones independientes
    indepRows = 1:rank_A;
    A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
    [~,~,V] = svd(A_red);

    if exist('vector_tangente','var')
        vector_tangente_anterior = vector_tangente;
    end
    vector_tangente = V(:,end);
    if exist('vector_tangente_anterior','var')
        signo = vector_tangente_anterior'*vector_tangente;
        if signo<0
            vector_tangente = -vector_tangente;
        end
    end
    z = z + vector_tangente*0.1; % Esto avanza pero se sale de la curva,
reconducir por newton.
```

```
    % VOLVEMOS A PNOER AQUI EL METODO DE NEWTON
   for i2 = 1 : 200

      f0 = f(z,C,xy,desired_cos_theta1);
      f0(eq_a_quitar,:) = [];
      J0 = J(z,C,xy,desired_cos_theta1); % calcular jacobiana usando funcion
      J0(eq_a_quitar,:) = [];

      % BLOQUE PARA ELIMINAR LAS ECUACIONES REDUNDANTES
      A = J0;
      b = -f0;
      [U, S, V] = svd(A, 'econ');
       % el rango efectivo
      tol = max(size(A)) * eps(max(diag(S)));
      tol = 1e-5;
      rank_A = sum(diag(S) > tol);
      rank_A = rango_forzado;
% Seleccionar ecuaciones independientes
      indepRows = 1:rank_A;
      A_red = U(:, indepRows)' * A;% Transformación que selecciona las filas
independientes
      b_red = U(:, indepRows)' * b;
% Obtener solución de mínima norma
      dz = lsqminnorm(A_red, b_red);

      z = z + dz*cr;
      if norm(dz)<0.00001
       % rank_A
         disp(strcat('Converge con estas iteraciones: ',num2str(i2)))
         break
      end
   end
   curva = [ curva ; z' ];
   i
end

eje1 = 12;
eje2 = 13;
eje3 = 23;

close all
plot3( curva(:,eje1) , curva(:,eje2) , curva(:,eje3) )
hold on
plot3( acercamiento(:,eje1) , acercamiento(:,eje2) , acercamiento(:,eje3) , 'r' )
```

```
plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'or' )
plot3( z_original(eje1) , z_original(eje2) , z_original(eje3) , 'sg' )


close all
paso_animacion = 1;
for i = 1 : paso_animacion : size(curva,1)
   hold off
   plot3( curva(:,eje1) , curva(:,eje2) , curva(:,eje3) )
   hold on
   plot3( acercamiento(:,eje1) , acercamiento(:,eje2) , acercamiento(:,eje3) , 'r' )
   plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'or' )
   plot3( z_original(eje1) , z_original(eje2) , z_original(eje3) , 'sg' )
   f0 = f(curva(i,:)',C,xy,desired_cos_theta1);
   title(f0(eq_a_quitar))
   if f0(eq_a_quitar)>0
      plot3( curva(i,eje1) , curva(i,eje2) , curva(i,eje3) , 'ob' , 'LineWidth' , 2 )
   else
      plot3( curva(i,eje1) , curva(i,eje2) , curva(i,eje3) , 'or' , 'LineWidth' , 2 )
   end

   pause(0.1)
end
```

6.2. CÓDIGO DE LAS FUNCIONES EN MATLAB PARA EL SEGUNDO ROBOT

6.2.1. FUNCIÓN f:

```
function salida = f(z,C,xy)

xp = xy(1);
yp = xy(2);
zp = xy(3);
a = 1;
b = 0.1;
c = 0.2;
n = 2; % relacion de reduccion (1 vuelta de theta genera 2 vueltas del pad)

x1 = z(1);
y1 = z(2);
z1 = z(3);
cos_alpha1 = z(4);
sin_alpha1 = z(5);
cos_beta1 = z(6);
sin_beta1 = z(7);
cos_gamma1 = z(8);
sin_gamma1 = z(9);
cos_theta1 = z(10);
sin_theta1 = z(11);
x2 = z(12);
y2 = z(13);
z2 = z(14);
cos_alpha2 = z(15);
sin_alpha2 = z(16);
cos_beta2 = z(17);
sin_beta2 = z(18);
cos_gamma2 = z(19);
sin_gamma2 = z(20);
cos_theta2 = z(21);
sin_theta2 = z(22);
x3 = z(23);
y3 = z(24);
z3 = z(25);
cos_alpha3 = z(26);
```

```
sin_alpha3 = z(27);
cos_beta3 = z(28);
sin_beta3 = z(29);
cos_gamma3 = z(30);
sin_gamma3 = z(31);
cos_theta3 = z(32);
sin_theta3 = z(33);
x4 = z(34);
y4 = z(35);
z4 = z(36);
cos_alpha4 = z(37);
sin_alpha4 = z(38);
cos_beta4 = z(39);
sin_beta4 = z(40);
cos_gamma4 = z(41);
sin_gamma4 = z(42);
cos_theta4 = z(43);
sin_theta4 = z(44);
x5 = z(45);
y5 = z(46);
z5 = z(47);
cos_alpha5 = z(48);
sin_alpha5 = z(49);
cos_beta5 = z(50);
sin_beta5 = z(51);
cos_gamma5 = z(52);
sin_gamma5 = z(53);
cos_theta5 = z(54);
sin_theta5 = z(55);
x6 = z(56);
y6 = z(57);
z6 = z(58);
cos_alpha6 = z(59);
sin_alpha6 = z(60);
cos_beta6 = z(61);
sin_beta6 = z(62);
cos_gamma6 = z(63);
sin_gamma6 = z(64);
cos_theta6 = z(65);
sin_theta6 = z(66);
x7 = z(67);
y7 = z(68);
z7 = z(69);
```

```
cos_alpha7 = z(70);
sin_alpha7 = z(71);
cos_beta7 = z(72);
sin_beta7 = z(73);
cos_gamma7 = z(74);
sin_gamma7 = z(75);
cos_theta7 = z(76);
sin_theta7 = z(77);
x8 = z(78);
y8 = z(79);
z8 = z(80);
cos_alpha8 = z(81);
sin_alpha8 = z(82);
cos_beta8 = z(83);
sin_beta8 = z(84);
cos_gamma8 = z(85);
sin_gamma8 = z(86);
cos_theta8 = z(87);
sin_theta8 = z(88);
x9 = z(89);
y9 = z(90);
z9 = z(91);
cos_alpha9 = z(92);
sin_alpha9 = z(93);
cos_beta9 = z(94);
sin_beta9 = z(95);
cos_gamma9 = z(96);
sin_gamma9 = z(97);
cos_theta9 = z(98);
sin_theta9 = z(99);

alpha1 = atan2(sin_alpha1,cos_alpha1);
beta1 = atan2(sin_beta1,cos_beta1);
gamma1 = atan2(sin_gamma1,cos_gamma1);
theta1 = atan2(sin_theta1,cos_theta1);
alpha2 = atan2(sin_alpha2,cos_alpha2);
beta2 = atan2(sin_beta2,cos_beta2);
gamma2 = atan2(sin_gamma2,cos_gamma2);
theta2 = atan2(sin_theta2,cos_theta2);
alpha3 = atan2(sin_alpha3,cos_alpha3);
beta3 = atan2(sin_beta3,cos_beta3);
gamma3 = atan2(sin_gamma3,cos_gamma3);
theta3 = atan2(sin_theta3,cos_theta3);
```

```
alpha4 = atan2(sin_alpha4,cos_alpha4);
beta4 = atan2(sin_beta4,cos_beta4);
gamma4 = atan2(sin_gamma4,cos_gamma4);
theta4 = atan2(sin_theta4,cos_theta4);
alpha5 = atan2(sin_alpha5,cos_alpha5);
beta5 = atan2(sin_beta5,cos_beta5);
gamma5 = atan2(sin_gamma5,cos_gamma5);
theta5 = atan2(sin_theta5,cos_theta5);
alpha6 = atan2(sin_alpha6,cos_alpha6);
beta6 = atan2(sin_beta6,cos_beta6);
gamma6 = atan2(sin_gamma6,cos_gamma6);
theta6 = atan2(sin_theta6,cos_theta6);
alpha7 = atan2(sin_alpha7,cos_alpha7);
beta7 = atan2(sin_beta7,cos_beta7);
gamma7 = atan2(sin_gamma7,cos_gamma7);
theta7 = atan2(sin_theta7,cos_theta7);
alpha8 = atan2(sin_alpha8,cos_alpha8);
beta8 = atan2(sin_beta8,cos_beta8);
gamma8 = atan2(sin_gamma8,cos_gamma8);
theta8 = atan2(sin_theta8,cos_theta8);
alpha9 = atan2(sin_alpha9,cos_alpha9);
beta9 = atan2(sin_beta9,cos_beta9);
gamma9 = atan2(sin_gamma9,cos_gamma9);
theta9 = atan2(sin_theta9,cos_theta9);

f0 = [];

T1 = [ rotx(alpha1)*roty(beta1)*rotz(gamma1) , [x1;y1;z1] ; [0 0 0 1]];
T2 = [ rotx(alpha2)*roty(beta2)*rotz(gamma2) , [x2;y2;z2] ; [0 0 0 1]];
T3 = [ rotx(alpha3)*roty(beta3)*rotz(gamma3) , [x3;y3;z3] ; [0 0 0 1]];
T4 = [ rotx(alpha4)*roty(beta4)*rotz(gamma4) , [x4;y4;z4] ; [0 0 0 1]];
T5 = [ rotx(alpha5)*roty(beta5)*rotz(gamma5) , [x5;y5;z5] ; [0 0 0 1]];
T6 = [ rotx(alpha6)*roty(beta6)*rotz(gamma6) , [x6;y6;z6] ; [0 0 0 1]];
T7 = [ rotx(alpha7)*roty(beta7)*rotz(gamma7) , [x7;y7;z7] ; [0 0 0 1]];
T8 = [ rotx(alpha8)*roty(beta8)*rotz(gamma8) , [x8;y8;z8] ; [0 0 0 1]];
T9 = [ rotx(alpha9)*roty(beta9)*rotz(gamma9) , [x9;y9;z9] ; [0 0 0 1]];

ecuacion = [xp;yp;zp]-[T8(1,4);T8(2,4);T8(3,4)];
f0 = [ f0 ; ecuacion ];

ecuacion1 =
T1*center2port(2,theta1)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(2,theta4))
^(-1)*T4^(-1)- eye(4);
```

```
ecuacion1(end,:) = [];
f0 = [ f0 ; ecuacion1(:) ];

ecuacion2 =
T4*center2port(1,theta4)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(1,theta7))
^(-1)*T7^(-1)- eye(4);
ecuacion2(end,:) = [];
f0 = [ f0 ; ecuacion2(:) ];

ecuacion3 =
T7*center2port(6,theta7)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(1,theta9))
^(-1)*T9^(-1)- eye(4);
ecuacion3(end,:) = [];
f0 = [ f0 ; ecuacion3(:) ];

ecuacion4 =
T9*center2port(8,theta9)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(8,theta8))
^(-1)*T8^(-1)- eye(4);
ecuacion4(end,:) = [];
f0 = [ f0 ; ecuacion4(:) ];

ecuacion5 =
T8*center2port(7,theta8)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(6,theta6))
^(-1)*T6^(-1)- eye(4);
ecuacion5(end,:) = [];
f0 = [ f0 ; ecuacion5(:) ];

ecuacion6 =
T6*center2port(7,theta6)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(6,theta3))
^(-1)*T3^(-1)- eye(4);
ecuacion6(end,:) = [];
f0 = [ f0 ; ecuacion6(:) ];

ecuacion7 =
T3*center2port(1,theta3)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(1,theta5))
^(-1)*T5^(-1)- eye(4);
ecuacion7(end,:) = [];
f0 = [ f0 ; ecuacion7(:) ];

ecuacion8 =
T5*center2port(2,theta5)*[[rotx(pi)],[0;0;0];[0,0,0,1]]*(center2port(2,theta2))
^(-1)*T2^(-1)- eye(4);
ecuacion8(end,:) = [];
f0 = [ f0 ; ecuacion8(:) ];
```

```
%ENTRE MÓDULO Y BASE 1 2
 ecuacion9 = (T1*center2port(9,theta1))-([[1 0 0;0 -1 0;0 0 -
 1],[0;0;0];[0,0,0,1]]);
 ecuacion9(end,:) = [];
 f0 = [ f0 ; ecuacion9(:) ];

 ecuacion10 = (T2*center2port(9,theta2))-([[1 0 0;0 -1 0;0 0 -
 1],[2;0;0];[0,0,0,1]]);
 ecuacion10(end,:) = [];
 f0 = [ f0 ; ecuacion10(:) ];

 fcs1 = [ cos_alpha1^2 + sin_alpha1^2 - 1 ; cos_beta1^2 + sin_beta1^2 - 1 ;
 cos_gamma1^2 + sin_gamma1^2 - 1 ; cos_theta1^2 + sin_theta1^2 - 1 ];
 fcs2 = [ cos_alpha2^2 + sin_alpha2^2 - 1 ; cos_beta2^2 + sin_beta2^2 - 1 ;
 cos_gamma2^2 + sin_gamma2^2 - 1 ; cos_theta2^2 + sin_theta2^2 - 1 ];
 fcs3 = [ cos_alpha3^2 + sin_alpha3^2 - 1 ; cos_beta3^2 + sin_beta3^2 - 1 ;
 cos_gamma3^2 + sin_gamma3^2 - 1 ; cos_theta3^2 + sin_theta3^2 - 1 ];
 fcs4 = [ cos_alpha4^2 + sin_alpha4^2 - 1 ; cos_beta4^2 + sin_beta4^2 - 1 ;
 cos_gamma4^2 + sin_gamma4^2 - 1 ; cos_theta4^2 + sin_theta4^2 - 1 ];
 fcs5 = [ cos_alpha5^2 + sin_alpha5^2 - 1 ; cos_beta5^2 + sin_beta5^2 - 1 ;
 cos_gamma5^2 + sin_gamma5^2 - 1 ; cos_theta5^2 + sin_theta5^2 - 1 ];
 fcs6 = [ cos_alpha6^2 + sin_alpha6^2 - 1 ; cos_beta6^2 + sin_beta6^2 - 1 ;
 cos_gamma6^2 + sin_gamma6^2 - 1 ; cos_theta6^2 + sin_theta6^2 - 1 ];
 fcs7 = [ cos_alpha7^2 + sin_alpha7^2 - 1 ; cos_beta7^2 + sin_beta7^2 - 1 ;
 cos_gamma7^2 + sin_gamma7^2 - 1 ; cos_theta7^2 + sin_theta7^2 - 1 ];
 fcs8 = [ cos_alpha8^2 + sin_alpha8^2 - 1 ; cos_beta8^2 + sin_beta8^2 - 1 ;
 cos_gamma8^2 + sin_gamma8^2 - 1 ; cos_theta8^2 + sin_theta8^2 - 1 ];
 fcs9 = [ cos_alpha9^2 + sin_alpha9^2 - 1 ; cos_beta9^2 + sin_beta9^2 - 1 ;
 cos_gamma9^2 + sin_gamma9^2 - 1 ; cos_theta9^2 + sin_theta9^2 - 1 ];
 f0 = [ f0 ; fcs1 ; fcs2 ; fcs3; fcs4; fcs5; fcs6; fcs7; fcs8; fcs9 ];

 salida = C*f0;

 end
```

6.2.2. FUNCIÓN *J* (JACOBIANA):

```
function salida = J(z,C,xy)
Jf = [];
incremento = 0.00001;
for i = 1 : size(z)
   dz = zeros(size(z));
   dz(i) = incremento;
   columna = f(z+dz,C,xy) - f(z,C,xy);
   Jf = [ Jf , columna ];
end
Jf = Jf/incremento;
salida = Jf;

end
```

6.2.3. CONVERGENCIA_Y_TRAZADO:

```
%% Sección 1


% Primero converger en la curva

% Probar otros, variando x, y, manteniendo z = 2
% Para cada (x,y) que se pruebe, tienemos que ejecutar esto varias veces, porque
no siempre
% convergera a la misma curva.
% Guardar resultado de cada ejecucion (tanto para distintos xy como para
distintos z0)
%save 'prueba_1'

adapt_Y_to_z
xy = [ 0 1 2 ];

clear J

incremento = 0.00001;
z0 = 10*randn(99,1); % Dependiendo del z0 aleatorio inicial, tendremos unas
curvas u otras.
% z0 = z_original
% load('semilla_z0') % Un ejemplo que converge a una curva con varias
soluciones.
z = z0;
acercamiento = z';
```

```
eq_a_quitar = 13;
% eq_a_quitar = [];
% rango_forzado = 33;
rango_forzado = 98;
cr = 0.1;

for i = 1 : 1000
   i
   f0 = f(z,C,xy);
   f0(eq_a_quitar,:) = [];
   J0 = J(z,C,xy); % calcular jacobiana usando función
   J0(eq_a_quitar,:) = [];
   dz = -cr*pinv(J0)*f0;

   %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
   % Descomposición en valores singulares (SVD)
   A = J0;
   B = -f0;
   [U, S, V] = svd(A, 'econ');
   % el rango efectivo
   tol = max(size(A)) * eps(max(diag(S)));
   tol = 1e-5;
   rank_A = sum(diag(S) > tol);
   rank_A = rango_forzado;
   % Seleccionar ecuaciones independientes
   indepRows = 1:rank_A;
   A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
   b_red = U(:, indepRows)' * B;
   % Obtener solución de mínima norma
   dz = lsqminnorm(A_red, b_red);
   %%% FIN DEL BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES

   % dz = linsolve(J0,-f0);
   % det(J0'*J0)
   z = z + dz*cr;
   norm_dz = norm(dz)
   if norm_dz < 0.5
      cr = 1;
   end
   if norm_dz<0.00001
      break
   end
   acercamiento = [ acercamiento ; z' ];
end
i_converger_a_curva = i;
norm(f0)
% z(12:14)
cond(J0)
```

```
f_completa = f(z,C,xy)
norm(f_completa)

curva = z';


%% Sección 2: Marchar a lo largo de la curva
cr = 1;
while 1

   J0 = J(z,C,xy); % calcular jacobiana
   J0(eq_a_quitar,:) = []; % Quitamos primera fila

   %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
   A = J0;
   [U, S, V] = svd(A, 'econ');
   % el rango efectivo
   tol = max(size(A)) * eps(max(diag(S)));
   tol = 1e-5;
   rank_A = sum(diag(S) > tol);
   rank_A = rango_forzado;
   % Seleccionar ecuaciones independientes
   indepRows = 1:rank_A;
   A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
   [~,~,V] = svd(A_red);

   if exist('vector_tangente','var')
      vector_tangente_anterior = vector_tangente;
   end
   vector_tangente = V(:,end);
   % [U,S,V] = svd(J0);
   % vector_tangente = V(:,end);
   if exist('vector_tangente_anterior','var')
      signo = vector_tangente_anterior'*vector_tangente;
      if signo<0
         vector_tangente = -vector_tangente;
      end
   end
   z = z + vector_tangente*0.1; % Esto avanza pero se sale de la curva, reconducir
por newton.
   % VOLVEMOS A PONER AQUI EL METODO DE NEWTON
   for i2 = 1 : 200

      f0 = f(z,C,xy);
      f0(eq_a_quitar,:) = [];
      J0 = J(z,C,xy); % calcular jacobiana usando funcion
      J0(eq_a_quitar,:) = [];
```

```
    %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
    A = J0;
    B = -f0;
    [U, S, V] = svd(A, 'econ');
    % el rango efectivo
    tol = max(size(A)) * eps(max(diag(S)));
    tol = 1e-5;
    rank_A = sum(diag(S) > tol);
    rank_A = rango_forzado;
    % Seleccionar ecuaciones independientes
    indepRows = 1:rank_A;
    A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
    b_red = U(:, indepRows)' * B;
    % Obtener solución de mínima norma
    dz = lsqminnorm(A_red, b_red);

    z = z + dz*cr;
    if norm(dz)<0.00001
      % rank_A
      disp(strcat('Converge con estas iteraciones: ',num2str(i2)))
      break
    end
  end
  curva = [ curva ; z' ];

  norma_vuelta = norm(z-curva(1,:)');
  clc
  f0 = f(curva(end,:)',C,xy);
  [ size(curva) , norma_vuelta , f0(eq_a_quitar) ]
  save 'prueba_11'
  if norm_vuelta<0.001
    break
  end

end
```

%% Sección 3: Para dibujar la curva obtenida

```
% En rojo: los movimientos hasta converger a la curva
% En azul: la curva trazada, después de haber convergido en ella

eje1 = 67;
eje2 = 68;
eje3 = 69;

close all
plot3( curva(:,eje1) , curva(:,eje2) , curva(:,eje3) )
```

```
hold on
plot3( acercamiento(:,eje1) , acercamiento(:,eje2) , acercamiento(:,eje3) , 'r' )
plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'or' )
plot3( z_original(eje1) , z_original(eje2) , z_original(eje3) , 'sg' )


%% Sección 4: Muestra la animación del movimiento a lo largo de la curva
% y va detectando soluciones cuando cambia el signo (soluciones = puntos
negros)

soluciones = [];
close all
paso_animacion = 1;
clear f0
for i = 1 : paso_animacion : size(curva,1)
  hold off
  plot3( curva(:,eje1) , curva(:,eje2) , curva(:,eje3) )
  hold on
  plot3( acercamiento(:,eje1) , acercamiento(:,eje2) , acercamiento(:,eje3) , 'r' )
  plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'or' )
  plot3( z_original(eje1) , z_original(eje2) , z_original(eje3) , 'sg' )
  if exist('f0','var')
    f0_previo = f0;
  end
  f0 = f(curva(i,:)',C,xy);
  if exist('f0_previo','var') && f0_previo(eq_a_quitar)*f0(eq_a_quitar)<0
    soluciones = [ soluciones ; curva(i,:) ];
  end
  title(f0(eq_a_quitar))
  if f0(eq_a_quitar)>0
    plot3( curva(i,eje1) , curva(i,eje2) , curva(i,eje3) , 'ob' , 'LineWidth' , 2 )
  else
    plot3( curva(i,eje1) , curva(i,eje2) , curva(i,eje3) , 'or' , 'LineWidth' , 2 )
  end
  plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'm+' )
  if size(soluciones,1)>0
    plot3( soluciones(:,eje1) , soluciones(:,eje2) , soluciones(:,eje3) , '.k' )
  end

  pause(0.1)
end
```

En este apartado se incluye el código correspondiente a la última modificación realizada en el punto 4.4, el cual refleja los ajustes finales implementados en el desarrollo.

6.2.4. CONVERGENCIA_Y_TRAZADO: DE CAPÍTULO 4.4

```
%% Sección 1
% Primero converger en la curva

%% format long

% Probar otros, variando x, y, manteniendo z = 2
% Para cada (x,y) que se pruebe, tienemos que ejecutar esto varias veces, porque no siempre
% convergera a la misma curva.
% Guardar resultado de cada ejecucion (tanto para distintos xy como para distintos z0)
%save 'prueba_1'

adapt_Y_to_z
xy = [ 0 1 2 ];

clear J

incremento = 0.00001;
z0 = 10*randn(99,1); % Dependiendo del z0 aleatorio inicial, tendremos unas curvas u otras.
% z0 = z_original
% load('semilla_z0') % Un ejemplo que converge a una curva con varias soluciones.
z = z0;
acercamiento = z';
eq_a_quitar = 13;
% eq_a_quitar = [];
% rango_forzado = 33;
rango_forzado = 98;
cr = 0.1;
for i = 1 : 1000
   i
   f0 = f(z,C,xy);
   f0(eq_a_quitar,:) = [];
   J0 = J(z,C,xy); % calcular jacobiana usando funcion
   J0(eq_a_quitar,:) = [];
   dz = -cr*pinv(J0)*f0; %

   %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
```

```
    % Descomposición en valores singulares (SVD)
    A = J0;
    B = -f0;
    [U, S, V] = svd(A, 'econ');
    % el rango efectivo
    tol = max(size(A)) * eps(max(diag(S)));
    tol = 1e-5;
    rank_A = sum(diag(S) > tol);
    rank_A = rango_forzado;
    % Seleccionar ecuaciones independientes
    indepRows = 1:rank_A;
    A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
    b_red = U(:, indepRows)' * B;
    % Obtener solución de mínima norma
    dz = lsqminnorm(A_red, b_red);
    %%% FIN DEL BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES

    % dz = linsolve(J0,-f0);
    % det(J0'*J0)
    z = z + dz*cr;
    norm_dz = norm(dz)
    if norm_dz < 0.5
        cr = 1;
    end
    if norm_dz<0.00001
        break
    end
    acercamiento = [ acercamiento ; z' ];
end
i_converger_a_curva = i;
norm(f0)
% z(12:14)
cond(J0)
f_completa = f(z,C,xy)
norm(f_completa)

curva = z';


%% Sección 2: Marchar a lo largo de la curva
format long
cr = 1;
eq_a_quitar =123 ; % con 37 sale otra curva con las mismas 2 soluciones
while 1 %

    J0 = J(z,C,xy); % calcular jacobiana
    J0(eq_a_quitar,:) = []; % Quitamos primera fila
```

```
    %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
    A = J0;
    [U, S, V] = svd(A, 'econ');
    % el rango efectivo
    tol = max(size(A)) * eps(max(diag(S)));
    tol = 1e-5;
    rank_A = sum(diag(S) > tol);
    rank_A = rango_forzado;
    % Seleccionar ecuaciones independientes
    indepRows = 1:rank_A;
    A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
    [~,~,V] = svd(A_red);

    if exist('vector_tangente','var')
        vector_tangente_anterior = vector_tangente;
    end
    vector_tangente = V(:,end);
    % [U,S,V] = svd(J0);
    % vector_tangente = V(:,end);
    if exist('vector_tangente_anterior','var')
        signo = vector_tangente_anterior'*vector_tangente;
        if signo<0
            vector_tangente = -vector_tangente;
        end
    end
    z = z + vector_tangente*0.1; % Esto avanza pero se sale de la curva, reconducir
por newton.
    % VOLVEMOS A PONER AQUI EL METODO DE NEWTON
    for i2 = 1 : 200

        f0 = f(z,C,xy);
        f0(eq_a_quitar,:) = [];
        J0 = J(z,C,xy); % calcular jacobiana usando funcion
        J0(eq_a_quitar,:) = [];

        %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
        A = J0;
        B = -f0;
        [U, S, V] = svd(A, 'econ');
        % el rango efectivo
        tol = max(size(A)) * eps(max(diag(S)));
        tol = 1e-5;
        rank_A = sum(diag(S) > tol);
        rank_A = rango_forzado;
        % Seleccionar ecuaciones independientes
        indepRows = 1:rank_A;
        A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
```

```
    b_red = U(:, indepRows)' * B;
    % Obtener solución de mínima norma
    dz = lsqminnorm(A_red, b_red);

    z = z + dz*cr;
    if norm(dz)<0.00001
      % rank_A
      disp(strcat('Converge con estas iteraciones: ',num2str(i2)))
      break
    end
  end
  curva = [ curva ; [z',eq_a_quitar] ];

  norma_vuelta = norm(z-curva(1,1:end-1)');
  clc
  f0 = f(curva(end,1:end-1)',C,xy);
  [ size(curva) , norma_vuelta , f0(eq_a_quitar) ]
  % save 'prueba_13'
  if norma_vuelta<0.001
    break
  end
 save 'prueba_12_eq_123!'
end
```



```
%% Sección 3: Para dibujar la curva obtenida
% En rojo: los movimientos hasta converger a la curva
% En azul: la curva trazada, despues de haber convergido en ella

eje1 = 67;
eje2 = 68;
eje3 = 69;

close all
load("prueba_12.mat");
curva_origenal= curva;
load("prueba_12_eq_80!.mat");
plot3( curva(:,eje1) , curva(:,eje2) , curva(:,eje3) , 'b.' )
hold on
plot3( curva_origenal(:,eje1) , curva_origenal(:,eje2) , curva_origenal(:,eje3) , 'r.' )
xlim([-1.5 0]);   % Solo muestra X entre -2 y 2
ylim([0 1.5]);
zlim([1.5 2.5])
hold on
plot3( acercamiento(:,eje1) , acercamiento(:,eje2) , acercamiento(:,eje3) , 'r' )
plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'or' )
plot3( z_original(eje1) , z_original(eje2) , z_original(eje3) , 'sg' )
```

```
%% Sección 4: Muestra la animacion del movimiento a lo largo de la
curva
% y va detectando soluciones cuando cambia el signo (soluciones = puntos
negros)

soluciones = [];
close all
paso_animacion = 1;
clear f0 f0_previo
parar = false;
for i = 1 : paso_animacion : size(curva,1)
  i
  hold off
  plot3( curva(:,eje1) , curva(:,eje2) , curva(:,eje3) , '.' )
  hold on
  % plot3( acercamiento(:,eje1) , acercamiento(:,eje2) , acercamiento(:,eje3) , 'r' )
  plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'or' )
  plot3( z_original(eje1) , z_original(eje2) , z_original(eje3) , 'sg' )
  if exist('f0','var')
    f0_previo = f0;
  end
  f0 = f(curva(i,:)',C,xy);
  if exist('f0_previo','var') && f0_previo(curva(i,end))*f0(curva(i,end))<0
    soluciones = [ soluciones ; curva(i,:) ];
    parar = true;
  end
  title(f0(curva(i,end)))
  if f0(curva(i,end))>0
    plot3( curva(i,eje1) , curva(i,eje2) , curva(i,eje3) , 'ob' , 'LineWidth' , 2 )
  else
    plot3( curva(i,eje1) , curva(i,eje2) , curva(i,eje3) , 'or' , 'LineWidth' , 2 )
  end
  plot3( z0(eje1) , z0(eje2) , z0(eje3) , 'm+' )
  if size(soluciones,1)>0
    plot3( soluciones(:,eje1) , soluciones(:,eje2) , soluciones(:,eje3) , '+k' ,
'LineWidth' , 2 , 'MarkerSize' , 10)
  end
  %plot3( z_otra(eje1) , z_otra(eje2) , z_otra(eje3) , 'm+' )
  pause(0.1)
   if parar
     z = curva(i,1:end-1)';
     break
   end
end
```

```
%%  Sección 5: converger a solucion
for i = 1 : 1000
   f0 = f(z,C,xy);
   J0 = J(z,C,xy); % calcular jacobiana usando funcion

   %%% BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES
   % Descomposición en valores singulares (SVD)
   A = J0;
   B = -f0;
   [U, S, V] = svd(A, 'econ');
   % el rango efectivo
   tol = max(size(A)) * eps(max(diag(S)));
   tol = 1e-5;
   rank_A = sum(diag(S) > tol);
   rank_A = rango_forzado+1;
   % Seleccionar ecuaciones independientes
   indepRows = 1:rank_A;
   A_red = U(:, indepRows)' * A; % Transformación que selecciona las filas
independientes
   b_red = U(:, indepRows)' * B;
   % Obtener solución de mínima norma
   dz = lsqminnorm(A_red, b_red);
   %%% FIN DEL BLOQUE, PARA ELIMINAR LAS ECUACIONES REDUNDANTES

   % dz = linsolve(J0,-f0);
   % det(J0'*J0)
   z = z + dz;
   norm_dz = norm(dz)
   if norm_dz<0.00001
      break
   end
end
```

7. REFERENCIAS

Peidró, A., et al. (2025). Kinematics and Workspace of Closed-Chain Modular Reconfigurable Robots (Artículo en elaboración para su envío a una revista científica).

Peidró, A., Gallego, J., Payá, L., Marín, J. M., Reinoso, Ó. (2019b). Trajectory analysis for the MASAR: A new modular and single-actuator robot. Robotics, 8(3), 78.

Peidró, A., Payá, L., Cebollada, S., Román, V., & Reinoso, Ó. (2020, July). Solution of the forward kinematics of parallel robots based on constraint curves. In *International Conference on Informatics in Control, Automation and Robotics* (pp. 386-409). Cham: Springer International Publishing.

Prieto, Freddy Alexander (2025). Modelado dinámico de robots modulares MASAR. Trabajo de Fin de Máster, Universidad Miguel Hernández de Elche.

Romanishin, J. W., Gilpin, K., Claici, S., & Rus, D. (2015). 3D M-Blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In Robotics and Automation (ICRA), 2015 IEEE International Conference on (pp. 1925-1932). IEEE.

Shen, W. M., Salemi, B., & Will, P. (2000). Hormones for self-reconfigurable robots. In Proceedings of the 6th International Conference on Intelligent Autonomous Systems.