E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Identificación dinámica del robot KUKA LWR IIWA mediante el diseño óptimo de experimentos

Máster Universitario en Ingeniería
Mecánica Aplicada y Computacional

Trabajo Fin de Máster

Iker Pellejero Pérez

Xabier Iriarte Goñi

Pamplona, 24 de septiembre de 2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

AGRADECIMIENTOS

Este trabajo es fruto de muchos años de estudio y considero que es una bonita forma de terminar una etapa en la que he invertido tiempo, dedicación, pasión, interés, lágrimas, alegrías… Ha sido un viaje largo, pero no lo he recorrido solo.

Gracias a mis padres por su infinita paciencia y por siempre apoyarme en estudiar lo que me gustaba, aunque ello provocara mi encierro durante horas y horas en mi habitación, guerreando con algún problema, con algún código. Alguno de muchos.

Gracias a Carlota, fiel compañera, con más paciencia que una santa y siempre con una mirada de comprensión y fuerza en los brazos para levantarme. Profesora que me enseña día a día, medio ingeniera a la fuerza.

Gracias a mis amigos, porque sin sus retorcidas formas de reírse de estas frikadas, habría sido todo más aburrido.

Gracias a Xabier Iriarte, que ha atendido mi ingente cantidad de dudas con una sonrisa siempre, fuera la hora que fuera y fuera la pregunta que fuera. Sin su conocimiento y dedicación, este trabajo se habría terminado en la página 15.

Gracias a todos los profesores del máster. Cada uno ha dejado algo en este chaval con ganas infinitas de aprender.

Lo consiguieron porque no sabían que era imposible.

(Jean Cocteau)

A veces la persona que nadie imagina capaz de nada,

es la que hace cosas que nadie imagina.

(Alan Turing)

RESUMEN

Este trabajo tiene por objeto realizar un proceso completo de identificación dinámica del robot colaborativo KUKA LWR IIWA.

El planteamiento se basa en analizar las metodologías existentes de identificación dinámica y seleccionar un método de identificación que permita obtener los parámetros dinámicos deseados.

Se deberá realizar un modelado dinámico del robot para obtener las ecuaciones que rigen el sistema y se planteará una reducción del modelo a parámetros base.

Será necesario un diseño de experimentos, definiendo la trayectoria/s óptima/s para llevar a cabo la identificación paramétrica.

Realizar los experimentos y medir los pares reales en las trayectorias reales, correlando así los datos del modelo estimado con el real.

Utilizando los valores experimentales, se realizará una estimación de los parámetros.

Por último, utilizando el modelo identificado, se estimarán los pares de una trayectoria de validación que no fuera utilizada en los experimentos para la estimación. Comparando los valores de los pares estimados y los medidos experimentalmente para esa trayectoria, se validará el modelo identificado.

PALABRAS CLAVE: KUKA, robot, identificación, modelado, dinámica, reducción, parámetros, trayectoria.

ABSTRACT

This work aims to make a complete process of dynamic identification of the collaborative robot KUKA LWR IIWA.

The approach is based on analyzing the existing dynamic identification methods and select one of them, which allows to obtain the desired dynamic parameters.

A dynamic modeling must be done in order to obtain the equations which rule the system and a reduction of the model to base parameters will be considered.

A design of experiments will be necessary, defining the optimal/s trajectory/ies for a parametric identification.

Make the experiments and measure the real torques in the real trajectories, correlating the estimated model data whit the real ones.

Using experimental values, an estimation of the parameters will be done.

Finally, using the identified model, the torques of a validation trajectory (not used in the identification experiments) will be estimated. Comparing the values of the estimated torques with the measured ones for that trajectory, the identified model will be validated.

KEY WORDS: KUKA, robot, identification, modeling, dynamic, reduction, parameters, trajectory.

1 ÍNDICE DE CONTENIDO

2 Introducción, motivación y objetivos 8
3 Metodología a emplear 10
4 Modelo cinemático 11
4.1 Descripción de la geometría del robot 12
4.2 Cinemática del robot 19
4.2.1 Parametrización cinemática 20
4.2.2 Definición de sólidos 26
5 Modelo dinámico 29
5.1 Obtención de las ecuaciones dinámicas 30
5.1.1 Principio de formulación elegido 30
5.1.2 Planteamiento de los torsores 32
5.1.3 Planteamiento de las ecuaciones 40
5.1.4 Simulaciones dinámicas (validación del modelo preliminar) 42
5.2 Estructuración simbólica 44
5.2.1 Simplificación de las ecuaciones 44
5.2.2 Técnicas simbólicas 46
5.2.3 Generación de funciones para el problema de identificación 48
6 Diseño de experimentos 49
6.1 Reducción del modelo 50
6.2 Parametrización de trayectorias 53
6.3 Optimización de trayectorias 54
6.3.1 Número de armónicos 55
6.3.2 Período fundamental 55
6.3.3 Incremento temporal 56
6.3.4 Límites del robot 57
6.3.5 Criterios de optimización 60
6.4 Obtención de trayectorias de enlace (Arranque y parada) 61
7 Identificación dinámica 66
7.1 Resultados 67
7.2 Validación 78
8 Conclusiones 85
9 Bibliografía y referencias 88
10 Anexos 89

2 INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS

Este trabajo fin de máster pone fin a una etapa de 7 años estudiando y especialización en los aspectos que más atractivos (personalmente) de la ingeniería, concretamente la ingeniería mecánica y su optimización a lo largo de los años con la implementación y desarrollo de metodologías y técnicas computacionales.

En él se ha intentado plasmar todo el conocimiento que adquirido en este máster, especialmente en el campo de este ámbito que más me interesa, los robots, el modelado y simulación de sistemas multicuerpo, y la identificación de sistemas dinámicos.

La versatilidad de estas técnicas y lo potentes que son, fueron las motivaciones principales a la hora de enfocar el trabajo final, con el que se cierra esta etapa. Las técnicas que en este trabajo se han usado y que aquí se presentan, son técnicas genéricas de modelado de sistemas, obtención de ecuaciones, simplificación de las mismas, reducción de modelos e identificación dinámica del sistema mencionado, fundamentalmente para obtener los valores de parámetros que no pueden saberse o buscarse en medios convencionales.

Se trata de realizar estimaciones sobre los parámetros que definen el sistema y su comportamiento, para conocerlo mejor, modelarlo fielmente y poder realizar trabajos de simulación, análisis y obtención de conclusiones para tomar decisiones técnicas sobre el sistema.

Este trabajo se centra en el robot KUKA LWR IIWA. Sin embargo, las técnicas que aquí se presentan, son extrapolables a más sistemas dinámicos (con leves particularizaciones y variaciones).

De forma resumida y simplificada, podría resumirse en que el problema genérico al que pretendo dar respuesta en este proyecto, es el siguiente:

Una empresa tiene un robot y que quiere construir un modelo dinámico que le permita representar fielmente la realidad de dicho robot, que teóricamente es igual a todos los robots de ese modelo, pero que no es del todo cierto, puesto que hay parámetros especiales como la viscosidad del enlace, los tensores de inercia, rigideces en los enlaces… que varían de un modelo a otro, y que no son fáciles de obtener.

Para ello, se plantea un proceso completo de modelado del sistema multicuerpo (robot KUKA LWR IIWA), se simula para validar la fiabilidad del modelo obtenido (a priori), y se realiza una reducción del modelo a parámetros base para la posterior identificación.

Tras esto se diseñan una serie de trayectorias que sirvan para hacer experimentos, identificar los parámetros y posteriormente, validar los resultados con otra trayectoria.

Esto podría servir a una empresa que sospecha que algún brazo roza más de lo que debe, o que sospecha que el robot está deteriorado, que ha perdido rigidez, que ha sido dañado, que no está lo suficientemente engrasado, que vibra… Hay multitud de campos y empresas que podrían beneficiarse de este tipo de técnicas.

Fundamentalmente, se pretende demostrar la robustez, versatilidad e interés de estas técnicas aprendidas.

El primer paso antes de comenzar con la los trabajos de construcción del modelo, obtención de las ecuaciones… es estudiar el estado del arte de esta rama de la ingeniería mecánica aplicada y computacional: las técnicas de identificación dinámica de robots de varios grados de libertad.

Para ello se han recopilado una serie de artículos de científicos prestigiosos y experimentados en el ámbito de la identificación y modelado dinámico de robots. Dichos artículos tienen cierto rigor a la hora de abordar los problemas que estas técnicas presentan, las explicaciones que en ellos se dan, son muy útiles y enriquecedoras, puesto que sus autores gozan de una amplia experiencia en este campo y se han enfrentado a problemas muy variados.

Los artículos en cuestión[1]–[6] son los que se indican en la bibliografía.

Adicionalmente se ha estudiado la tesis [7] del tutor de este trabajo fin de grado, Xabier Iriarte, quien tiene una amplia experiencia en el estudio de robots manipuladores (y de todo tipo en general) y domina técnicas de identificación de sistemas dinámicos, así como de modelado dinámico.

Sin embargo, el trabajo en el que más se basa el presente proyecto, es el redactado por Gautier, Jubien y Janot [4].

Estos trabajos servirán de orientación ante posibles dudas que puedan presentarse a lo largo del discurrir de este proyecto.

Por último, es conveniente recalcar que este trabajo había sido pensado para realizar los experimentos con un robot real que tiene la empresa ALDAKIN. Lamentablemente, no ha sido posible completar estos experimentos y los datos han tenido que ser creados por el autor, a título personal, para poder seguir adelante y realizar las tareas de identificación dinámica estudiadas en este máster.

3 METODOLOGÍA A EMPLEAR

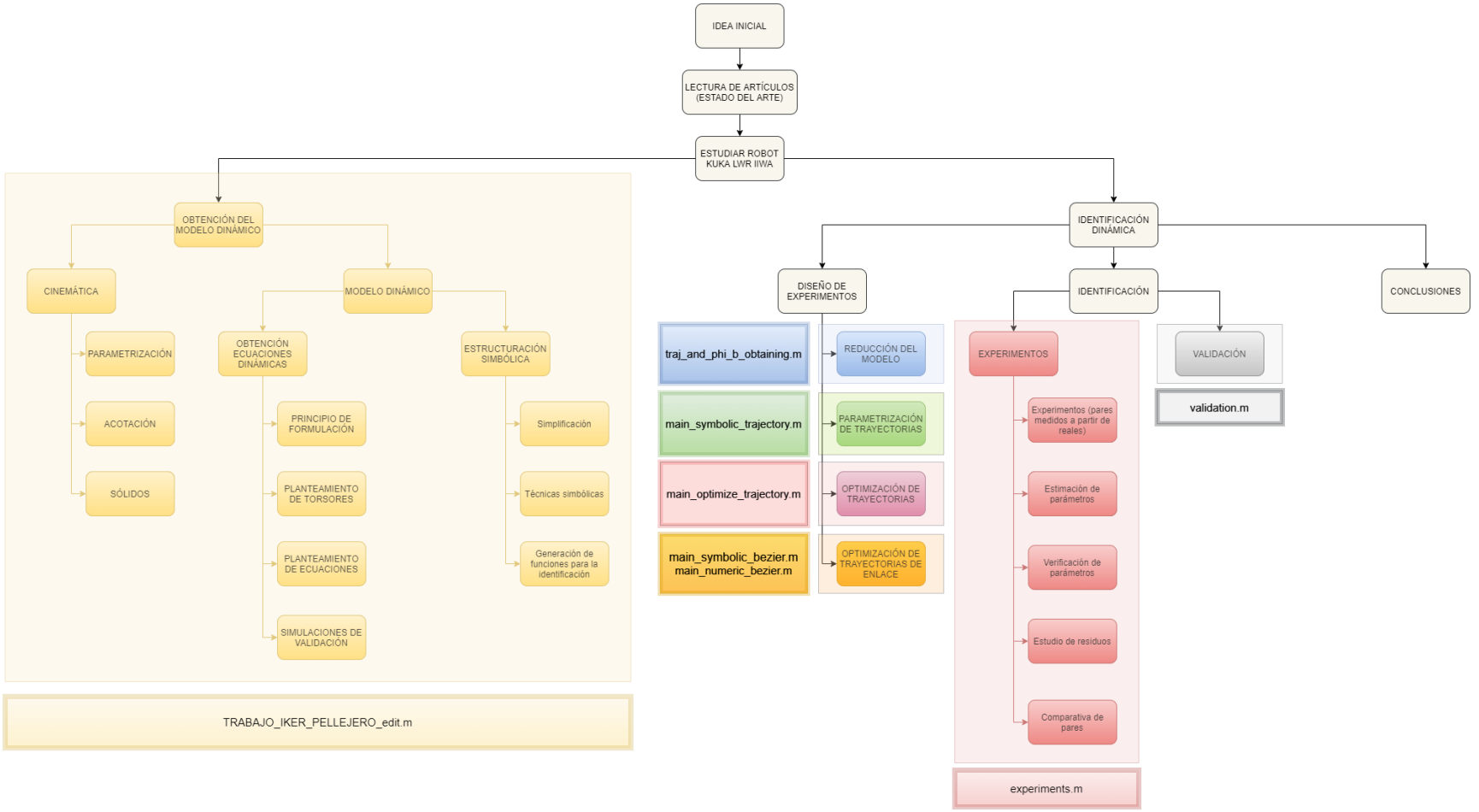


Fig 1: Resumen de la metodología a emplear

4 MODELO CINEMÁTICO

El estudio cinemático podría describirse como el estudio del movimiento al margen de sus causas. Dicho de una forma más clara y aplicada al caso que en este proyecto se trabaja, podría entenderse como el análisis de la geometría y el movimiento de un robot haciendo uso de diferentes sistemas de referencia establecidos (haciendo uso de conceptos básicos de mecánica), a lo largo del tiempo, sin tener en cuenta ni analizar las fuerzas y momentos que inducen al movimiento.

En este capítulo se detalla la resolución del problema dinámico del robot, que podría entenderse como un planteamiento y desarrollo de los problemas de montaje, posición inicial y simulación cinemática asociados.

La cinemática del sistema dictará la evolución de las coordenadas generalizadas, sujetas a determinadas restricciones cinemáticas (consecuencia de la geometría y características del robot).

Para el planteamiento del modelo cinemático, se retoman y recuerdan conceptos básicos estudiados a lo largo del máster cursado, tales como la definición de vectores para definir distancias entre los enlaces cinemáticos del robot, bases ortonormales para generar sistemas de referencia que permitan estudiar el movimiento de los diferentes cuerpos enlazados que conforman el robot…

Para plantear el modelo cinemático del robot KUKA IIWA LWR, se han planteado una serie de pasos que permitan abordar el problema con eficacia, y alcanzar soluciones correctas y óptimas para el posterior modelo dinámico.

En primer lugar será necesario conocer el robot en detalle. Cuántos grados de libertad tiene, si es de cadena abierta o cerrada, cómo está construido, cuántos cuerpos tiene y deben ser caracterizados, cómo se enlazan dichos cuerpos, qué dimensiones tiene el robot según el modelo, cuál es su espacio de trabajo... Además de conocer en detalle el robot, es conveniente revisar también el estado del arte en el modelado de robots (haciendo uso de los artículos previamente mencionados), para poder ver cómo se construyen los puntos, vectores, bases y sistemas de referencia en la obtención de modelos analíticos de robots.

En segundo lugar, teniendo claro todo lo anterior, se podrá proceder con la parametrización cinemática del robot, para por último, dibujar los sólidos y obtener una representación 3D simplificada del robot, que nos permita visualizarlo y simular su movimiento bajo diferentes supuestos de comportamiento a priori conocido, con el fin de validar los modelos que posteriormente obtengamos.

En los subcapítulos que siguen, se abordan estas etapas que permiten la obtención del modelo cinemático.

4.1 DESCRIPCIÓN DE LA GEOMETRÍA DEL ROBOT

El robot KUKA IIWA LWR, es un robot colaborativo de 7 grados de libertad de cadena abierta. Los robots colaborativos son aquellos que se emplean en la industria para cooperar con humanos en tareas generalmente repetitivas o que requieran una precisión por encima de la alcanzable para el ser humano.

Los robots como el que aquí se estudia, son cada vez más empleados y comercializados en empresas que se están adentrando en la industria 4.0 y que están buscando automatización de procesos industriales.

En algunas de las siguientes imágenes puede verse al robot KUKA IIWA LWR, trabajando con humanos en operaciones del entorno industrial.

Fig 2: Operarios trabajando en colaboración con el robot KUKA IIWA LWR [8]

Fig 3: Operaria trabajando en paralelo con un KUKA IIWA LWR [8]

Por cómo ha sido diseñado este robot, puede trabajar prácticamente en cualquier entorno industrial, y en ocasiones, para optimizar su campo de trabajo en determinadas aplicaciones (por ejemplo para alcanzar objetos distribuidos en una mesa, o para realizar operaciones en sentido de la gravedad), en las que el robot se coloca en posición cenital. En las trayectorias que vamos a emplear para la identificación dinámica, y en el modelo analítico que vamos a construir del robot para dichas trayectorias, hemos considerado que el robot trabaja en la posición estándar vertical (no cenital).

En la imagen siguiente puede verse un ejemplo de este robot trabajando en posición cenital.

Fig 4: Robot KUKA IIWA LWR trabajando en posición cenital [8]

Este robot en concreto, tiene 7 grados de libertad, lo cual le otorga una flexibilidad y versatilidad considerables y que generalmente indica que puede realizar multitud de trabajos.

Los grados de libertad son los que se detallan a continuación:

- Movimiento relativo de giro (enlace de revolución) entre base y sólido 1.
- Movimiento relativo de giro (enlace de revolución) entre base y sólido 2.
- Movimiento relativo de giro (enlace de revolución) entre base y sólido 3.
- Movimiento relativo de giro (enlace de revolución) entre base y sólido 4.
- Movimiento relativo de giro (enlace de revolución) entre base y sólido 5.
- Movimiento relativo de giro (enlace de revolución) entre base y sólido 6.
- Movimiento relativo de giro (enlace de revolución) entre base y sólido 7.

En la siguiente imagen se detallan cuáles son estos grados de libertad y sólidos a los que se refiere el punto anterior.

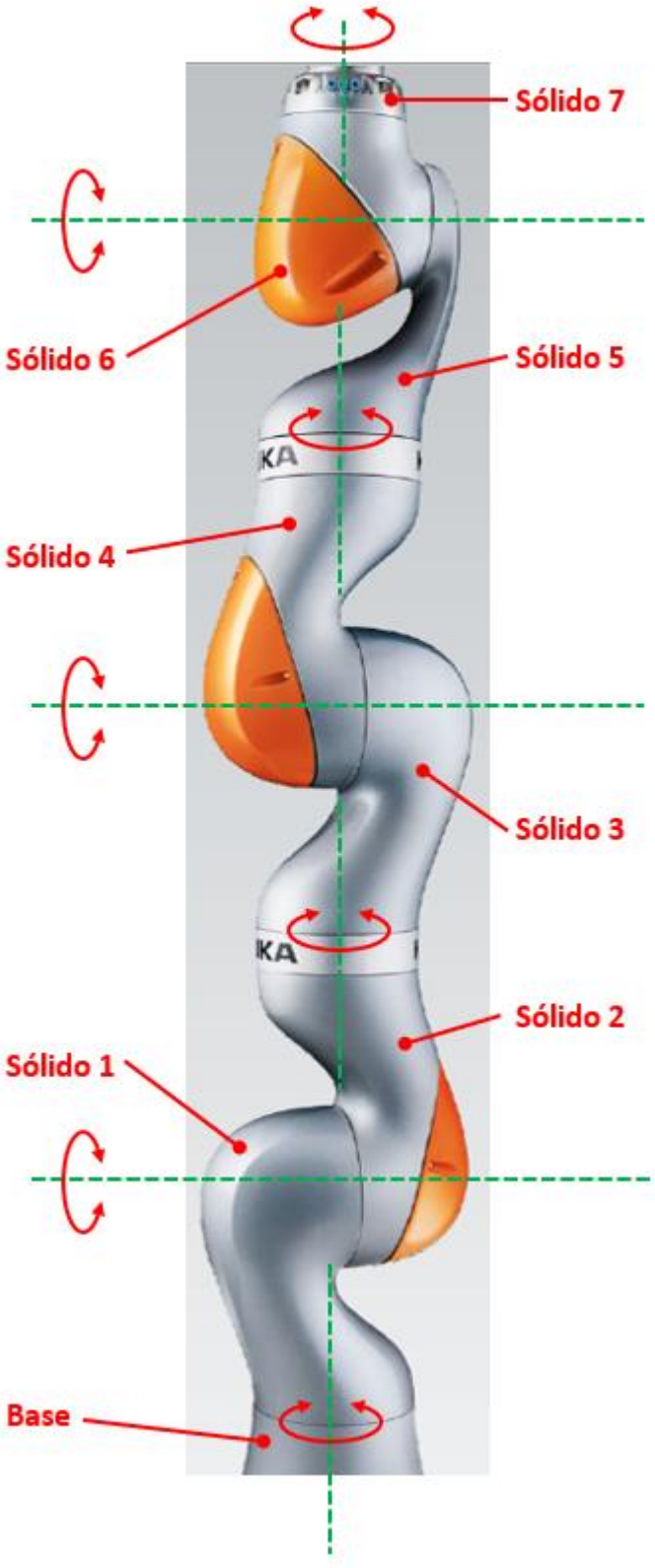
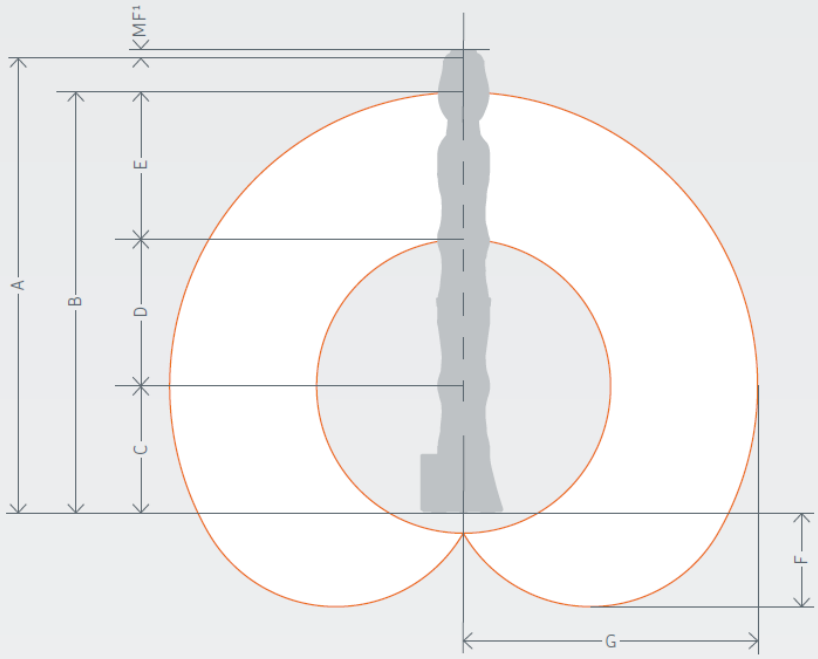


Fig 5: Sólidos y grados de libertad del robot KUKA IIWA LWR

Como puede observarse en la imagen anterior, este robot se compone de 8 sólidos, que se unen entre sí por medio de 7 enlaces sucesivos de revolución.

Como se verá más adelante, en cada uno de estos "joints" o enlaces de revolución, se tiene un encoder y sensor de par. Las configuraciones de cómo se colocan estos elementos en el robot pueden variar y de su ubicación y configuración, dependerá el posterior análisis que pueda llevarse a cabo.

Para terminar con la descripción de la geometría del robot, se muestran a continuación una serie de datos, especificaciones y propiedades del robot, que han sido extraídas del catálogo oficial del robot, en la página web de KUKA:



¹ dependent on the media flange option

Workspace	Dimensions A	Dimensions B	Dimensions C	Dimensions D	Dimensions E	Dimensions F	Dimensions G	Volume
LBR iiwa 7 R800	1,266 mm	1,140 mm	340 mm	400 mm	400 mm	260 mm	800 mm	1.7 m³
LBR iiwa 14 R820	1,306 mm	1,180 mm	360 mm	420 mm	400 mm	255 mm	820 mm	1.8 m³

Fig 6: Espacio de trabajo del robot y dimensiones de los sólidos que acotan dicho espacio [8]

LBR iiwa	LBR iiwa 7 R800	LBR iiwa 14 R820
Rated payload	7 kg	14 kg
Number of axes	7	7
Wrist variant	In-line wrist	In-line wrist
Mounting flange A7	DIN ISO 9409-1-A50	DIN ISO 9409-1-A50
Installation position	any	any
Positioning accuracy (ISO 9283)	± 0.1 mm	± 0.1 mm
Axis-specific torque accuracy	± 2 %	± 2 %
Weight	23.9 kg	29.9 kg
Protection rating	IP 54	IP 54

Axis data / Range of motion		LBR iiwa 7 kg Maximum torque	LBR iiwa 7 kg Maximum velocity	LBR iiwa 14 kg Maximum torque	LBR iiwa 14 kg Maximum velocity
Axis 1 (A1)	± 170°	176 Nm	98°/s	320 Nm	85°/s
Axis 2 (A2)	± 120°	176 Nm	98°/s	320 Nm	85°/s
Axis 3 (A3)	± 170°	110 Nm	100°/s	176 Nm	100°/s
Axis 4 (A4)	± 120°	110 Nm	130°/s	176 Nm	75°/s
Axis 5 (A5)	± 170°	110 Nm	140°/s	110 Nm	130°/s
Axis 6 (A6)	± 120°	40 Nm	180°/s	40 Nm	135°/s
Axis 7 (A7)	± 175°	40 Nm	180°/s	40 Nm	135°/s

Programmable Cartesian stiffness		
Min. (X, Y, Z)	0.0 N/m	0.0 N/m
Max. (X, Y, Z)	5,000 N/m	5,000 N/m
Min. (A, B, C)	0.0 N/rad	0.0 N/rad
Max. (A, B, C)	300 Nm/rad	300 Nm/rad

KUKA Sunrise Cabinet	
Processor	Quad-core processor
Hard drive	SSD
Interfaces	USB, EtherNet, DVI-I
Protection rating	IP20
Dimensions (D x W x H)	500 mm x 483 mm x 190 mm
Weight	23 kg

Power supply connection	
Rated supply voltage	AC 110 V to 230 V
Permissible tolerance of rated voltage	± 10 %
Mains frequency	50 Hz ± 1 Hz or 60 Hz ± 1 Hz
Mains-side fusing	2 x 16 A slow-blowing

30,000 operating hours

Fig 7: Especificaciones técnicas del robot KUKA IIWA LWR [8]

Es importante tener en cuenta, que el robot que se va a estudiar en este trabajo, es el robot KUKA IIWA LWR de 14 kg, cuyo nombre comercial oficial es (del alemán):

LBR IIWA 14 R820

Por lo que se emplearán los datos correspondientes a la columna de ese modelo de robot.

En los siguientes capítulos de este documento se detalla la cinemática del robot desde un punto de vista más técnico, aplicado ya al proyecto en sí y a la obtención del modelo analítico.

Fig 8: Robots con los que iban a realizarse los experimentos

4.2 CINEMÁTICA DEL ROBOT

Este capítulo detalla los pasos que se han dado para obtener el modelo cinemático del robot mencionado. Este punto se abordará de manera descriptiva, mostrando los pasos llevados a cabo, las decisiones tomadas y el por qué de las mismas.

Es muy importante realizar una correcta definición y parametrización de la cinemática del robot, porque tiene un papel fundamental que jugar en la posterior definición del modelo dinámico, y en las simulaciones dinámicas que se llevarán a cabo para validar de forma intuitiva el modelo analítico final (con las ecuaciones dinámicas que se obtengan).

Una mala elección de coordenadas generalizadas, bases, sistemas de referencias, ecuaciones… puede complicar notablemente la generación de las ecuaciones dinámicas globales, así como la posterior simplificación, gestión y utilización de las mismas, en las etapas de identificación dinámica.

Como puede verse, la idea que subyace constantemente es realizar una identificación dinámica haciendo uso de un modelo analítico previamente obtenido (podría entenderse como una etapa de preproceso), y de unos resultados de unos experimentos diseñados. La definición de la cinemática y la dinámica del robot, entran dentro de esa primera etapa de preproceso, ardua, pero necesaria para una correcta identificación.

En los siguientes subcapítulos se detalla la parametrización cinemática realizada, así como la definición analítica, y virtual de los sólidos (necesaria tanto para la obtención de las ecuaciones, como para la realización de las simulaciones dinámicas).

Una vez conseguido este modelo, se podrá conocer la posición, velocidad y aceleración de los múltiples cuerpos que conforman el robot. Los cuerpos tienen 6 formas de moverse en el espacio (desplazamientos a lo largo de los 3 ejes cartesianos, y giros alrededor de dichos ejes. Sin embargo, a menudo existen limitaciones en los pares cinemáticos de los enlaces entre sólidos, que limitan la forma en que éstos pueden moverse. Este hecho obliga a la definición y consideración de ecuaciones de enlace cinemático.

En el caso de un robot de cadena abierta, no hay ecuaciones de enlace geométrico (por lo tanto sus derivadas inexistentes, denotan que no existirán ecuaciones de enlace cinemático holónomas). Esto se debe a que el número de ecuaciones de enlace geométrico (m), se obtiene de la diferencia entre coordenadas generalizadas que acotan el mecanismo (n) y el número de ellas que son independientes del resto (f). Así pues, en este caso:

$$m = n - f = 7 - 7 = 0$$

Tampoco existirán ecuaciones no holónomas, puesto que no existen relaciones adicionales (que no tienen origen geométrico) entre las velocidades generalizadas (un ejemplo sería condición de no deslizamiento en rodadura de un cuerpo, inexistente en este caso).

El hecho de que el robot sea de cadena abierta, con estas consideraciones, simplifica considerablemente la sección de modelado cinemático, como podrá verse más adelante.

En este capítulo se comienza a trabajar con MATLAB, creando un código de obtención del modelo analítico, estructurado por fases, según los capítulos de este proyecto. En los capítulos siguientes se muestran algunos extractos clave de varios de estos códigos, donde se aplican conceptos teóricos que se han explicado con anterioridad.

4.2.1 Parametrización cinemática

Por parametrización cinemática puede entenderse al conjunto de pasos previos a la obtención del modelo, en la que se definen coordenadas generalizadas, bases, sistemas de referencia, vectores, condiciones iniciales, ecuaciones cinemáticas (si las hubiera) …

El problema se puede plantear en términos analíticos mediante la definición de la posición del mecanismo (robot en este caso), en función de un conjunto de variables constantes en el tiempo, a las cuales se las conoce como parámetros.

$$param = \begin{bmatrix} param_1 \\ param_2 \\ \vdots \\ param_n \end{bmatrix}$$

Adicionalmente, esa posición se define también en función de variables que sí que cambian con el paso del tiempo. Estamos hablando de las coordenadas generalizadas:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix}$$

La derivada temporal de este vector anterior, da como resultado el vector de velocidades generalizadas del sistema, y estas a su vez, de forma análoga, dan como resultado de una nueva derivación el vector de aceleraciones generalizadas del sistema.

En este problema en concreto tenemos 7 coordenadas generalizadas (los 7 giros en la unión entre 2 sólidos sucesivos, del conjunto de 8 sólidos que conforman el robot). Además, es deducible que las 7 coordenadas generalizadas son independientes, es decir, su valor no depende de ninguna otra variables, coordenada o parámetro independiente, sino que el valor que toma cada una de estas coordenadas, es totalmente ajeno al resto de valores del resto de enlaces, y son gobernadas por motores y reductoras independientes, existentes en cada enlace.

De este hecho se deduce lo anteriormente comentado, en relación a la inexistencia de ecuaciones de enlace y ecuaciones cinemáticas en este problema.

Para llevar a cabo la parametrización cinemática del robot haciendo uso del software Matlab, se empleará la librería de funciones de la asignatura de este mismo máster, Dinámica de Sistemas Multicuerpo, usada anteriormente por el autor, y desarrollada por Javier Ros, profesor del máster. Dicha librería ha sido modificada por el autor para poder dar solución a problemas concretos que se han ido presentando en el desarrollo del proyecto. Más adelante se explicarán en detalle estos cambios.

Las coordenadas generalizadas se definen mediante la función “newCoord”, asignando a cada una de ellas un nombre, y unos valores de posición y velocidad iniciales.

```
newCoord('theta1',0,0);
newCoord('theta2',pi/2,0);
newCoord('theta3',0,0);
newCoord('theta4',0,0);
newCoord('theta5',0,0);
newCoord('theta6',0,0);
newCoord('theta7',0,0);
```

Fig 9: Coordenadas generalizadas definidas para el robot

Por otro lado, se definen los parámetros geométricos (longitudes, diámetros, alturas…) de los sólidos. Es importante tener en cuenta que los sólidos que se han creado, son una representación simplificada pero fidedigna de los sólidos real, para poder realizar las simulaciones posteriormente.

```
%Base
newParam('r1_base',85.5/1000);
newParam('r2_base',34.2/1000);
newParam('r_in_base',20/1000);
newParam('height_base',150/1000);
%Sol1
newParam('r_out_sol1',34.2/1000);
newParam('r_in_sol1',20/1000);
newParam('height_sol1',210/1000);
newParam('length_sol1',68/1000);
%Sol2
newParam('r_out_sol2',34.2/1000);
newParam('r_in_sol2',20/1000);
newParam('height_sol2',210/1000);
newParam('length_sol2',68/1000);
%Sol3
newParam('r_out_sol3',34.2/1000);
newParam('r_in_sol3',20/1000);
newParam('height_sol3',210/1000);
newParam('length_sol3',68/1000);
%Sol4
newParam('r_out_sol4',34.2/1000);
newParam('r_in_sol4',20/1000);
newParam('height_sol4',210/1000);
newParam('length_sol4',68/1000);
%Sol5
newParam('r_out_sol5',34.2/1000);
newParam('r_in_sol5',20/1000);
newParam('height_sol5',190/1000);
newParam('length_sol5',68/1000);
%Sol6
newParam('r_out_sol6',34.2/1000);
newParam('r_in_sol6',20/1000);
newParam('length_sol6',150/1000);
%Sol7
newParam('r_in_sol7',20/1000);
newParam('r1_out_sol7',34.2/1000);
newParam('r2_out_sol7',25/1000);
newParam('height_sol7',30/1000);
```

Fig 10: Definición de los parámetros geométricos que acotan por completo la representación simplificada del robot

Una vez definidas las coordenadas generalizadas y los parámetros que acotan el sólido (que es básicamente, conseguir la información que se plantea en la *Fig 5: Sólidos y grados de libertad del robot KUKA IIWA LWR*), se comienza con la definición de las bases y puntos, que posteriormente nos servirán para generar los sistemas de referencia.

A la hora de definir las bases hay que tener en cuenta algunos aspectos importantes. El primero de ellos es que para emplear la librería de funciones de Dinámica de sistemas multicuerpo, partimos de una base, un punto y una referencia ya creadas, que son la base "XYZ", el punto "O" y el sistema de referencia "Gr" o "IF" (suelo, Inertial Frame).

En el artículo leído al comienzo del proyecto [4], se explica que la cinemática de los robots de cadena abierta que se componen por un conjunto de sólidos enlazados en serie, se define usando la notación Modificada de Denavit y Hartenberg (notación MDH).

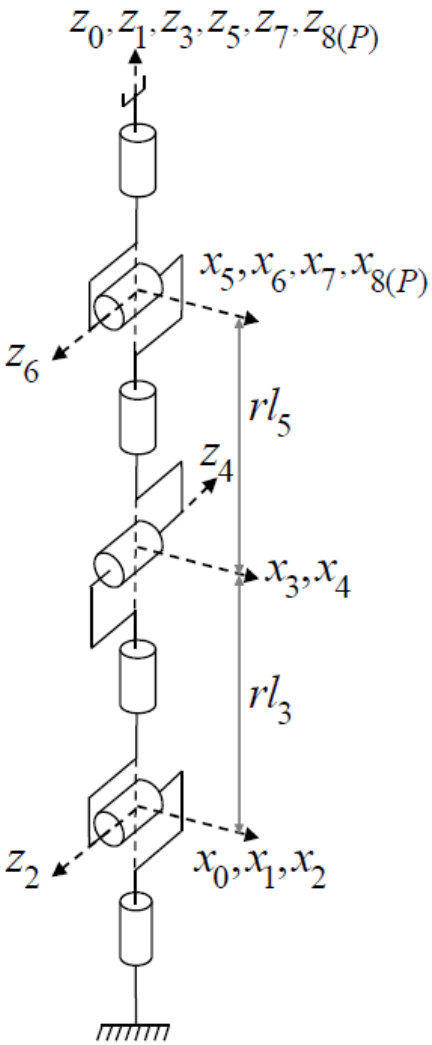


Fig 11: Boceto simplificado del robot, acotación MDH [CITAR ARTÍCULO]

Esta notación consiste en tener una referencia fija en el enlace "j", de forma que el eje "Z" del sistema de referencia, se orienta en la dirección del enlace (giro en torno a ese eje). El eje "X" se orienta en la dirección perpendicular común al "Z" de los dos sistemas de referencia de los sólidos enlazados. El ángulo "alpha" y la distancia "d", corresponden al ángulo y distancia entre los dos ejes "Z". Además existen otras variables, llamadas "theta" y "r", que se corresponden con el ángulo y la distancia entre los ejes "X" de dos sistemas de referencia enlazados.

Esta notación es la más comúnmente empleada por los científicos que se dedican a estudiar los robots de este tipo de forma analítica. Sin embargo, para poder hacer uso de la biblioteca de funciones de la asignatura de dinámica de sistemas multicuerpo, en este trabajo se ha empleado la notación clásica, en la que los sistemas de referencia se orientan de forma lógica e intuitiva partiendo de la posición de partida (definiendo por convenio, con el robot estirado, en reposo, el "Z" orientado en vertical ascendente).

A continuación se incluyen unos extractos clave del código de obtención del modelo dinámico, concretamente de la parte en la que se parametrizan las bases, puntos y sistemas y referencias del robot. Más adelante podrá consultarse una imagen más completa del robot, similar a al figura *Fig 5: Sólidos y grados de libertad del robot KUKA IIWA LWR*, donde se muestran estos aspectos de una forma más visual.

```
%Defining Bases
newBase('xyz','Base1',3,theta1);
newBase('Base1','Base2',2,theta2);
newBase('Base2','Base3',3,theta3);
newBase('Base3','Base4',2,theta4);
newBase('Base4','Base5',3,theta5);
newBase('Base5','Base6',2,theta6);
newBase('Base6','Base7',3,theta7);
```

Fig 12: Definición de las bases en Matlab

Como puede verse en la imagen anterior, las bases se definen como un movimiento relativo (en este caso son todo rotaciones) con respecto a las bases anteriores. Es una nueva forma de orientar un sólido. En este caso, la base del sólido 1 (el que va sobre la base del robot), se define como una base que gira en torno al eje "Z" de la base "XYZ", con un ángulo "theta1". De forma análoga, se definen las bases para el resto del sólido. Una forma visual y rápida de comprobar la correcta definición de bases, es que el robot KUKA IIWA LWR, tiene una sucesión de giros alternativos (en torno al eje 3 un enlace, en torno al eje 2 el siguiente enlace, entorno al 3 el siguiente… y así sucesivamente).

```
%Defining Points
newPoint('O','P1',Vector3D([0,0,height_base]','xyz'));
newPoint('P1','P2',Vector3D([0,0,height_sol1]','Base1'));
newPoint('P2','P3',Vector3D([0,0,height_sol2]','Base2'));
newPoint('P3','P4',Vector3D([0,0,height_sol3]','Base3'));
newPoint('P4','P5',Vector3D([0,0,height_sol4]','Base4'));
newPoint('P5','P6',Vector3D([0,0,height_sol5]','Base5'));
newPoint('P6','P7',Vector3D([0,0,length_sol6/3]','Base6'));
newPoint('P7','PEndEffector',Vector3D([0,0,height_sol7]','Base7'));
```

Fig 13: Definición de los puntos clave del mecanismo

En este caso, partiendo del punto "O", perteneciente al suelo, se empiezan a definir puntos clave que acotarán el mecanismo. Por ejemplo, el punto "P1", se encuentra orientado desde O, según la base "XYZ", en la dirección de "Z", a una altura sobre "O" igual al parámetro "height_base". Es decir, la altura de la base queda definida por el punto "O", que es el punto sobre el que asienta en el suelo, y el punto "P1", que se corresponde con el punto más alto de la base.

Siguiendo esta filosofía, vamos colocando puntos que permitirán ubicar los sólidos, en este caso dibujamos de forma inicial la posición de reposo, que es el robot totalmente estirado hacia arriba. La filosofía es ir construyendo nuevos puntos, partiendo de los existentes, ubicándolos distanciados el valor de un determinado parámetro previamente definido, en la dirección de uno de los ejes de las bases creadas.

Por último, se definen los sistemas de referencias.

```
% Inertial Frame/Ground
newFrame('O','xyz','Gr');
newFrame('O','xyz','IF');

% Defining Frames
newFrame('O','xyz','FBase');
newFrame('P1','Base1','FSol1');
newFrame('P2','Base2','FSol2');
newFrame('P3','Base3','FSol3');
newFrame('P4','Base4','FSol4');
newFrame('P5','Base5','FSol5');
newFrame('P6','Base6','FSol6');
newFrame('P7','Base7','FSol7');
```

Fig 14: Definición de los sistemas de referencia del robot

En este punto no hay demasiado que comentar, simplemente se definen los frames (sistemas de referencia) asociados a cada uno de los 8 sólidos, como una base situada en un punto, donde cada uno de los puntos recibe un tratamiento analítico como perteneciente a un sólido.

Previamente se ha definido el sistema inercial o suleo, orientado según la base "XYZ" en "O".

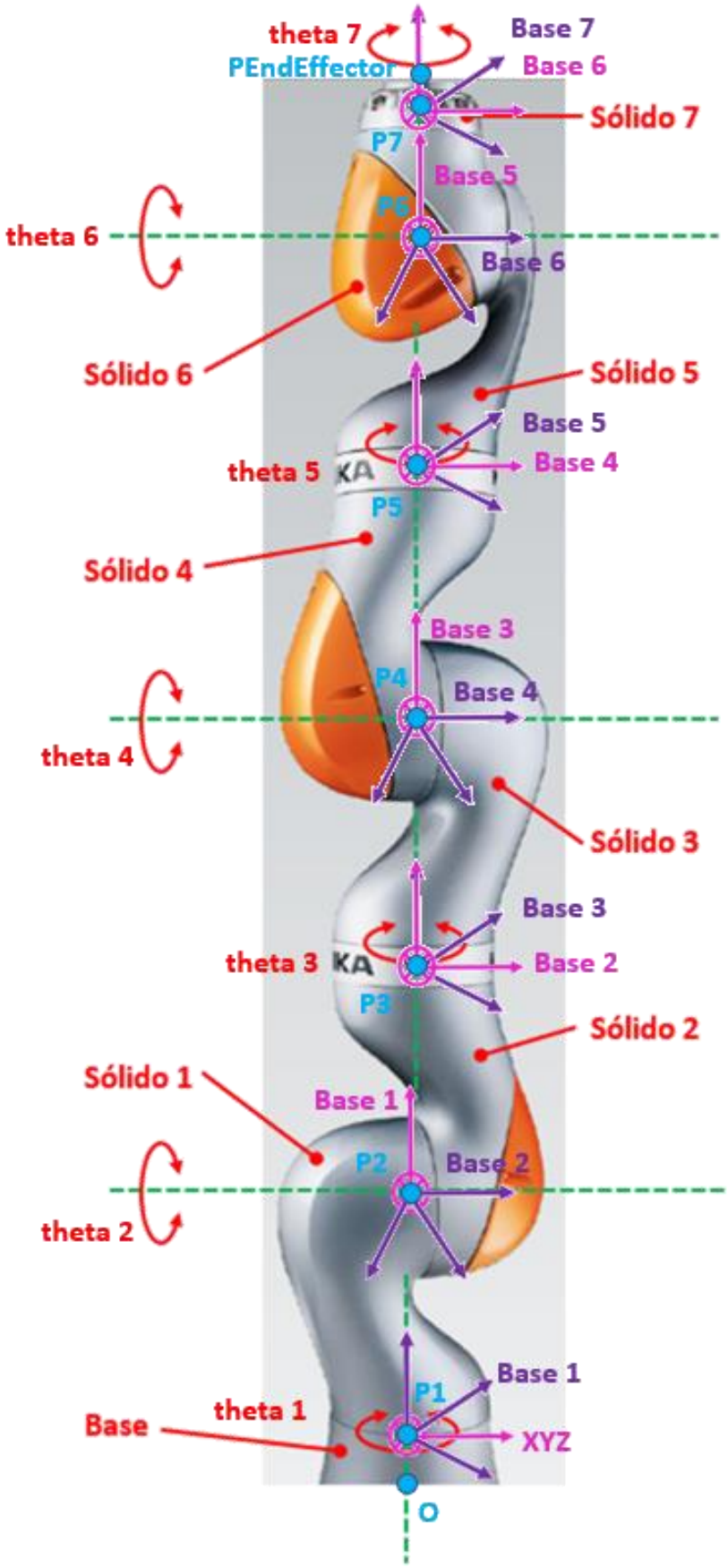


Fig 15: Acotación y parametrización completa del mecanismo

4.2.2 Definición de sólidos

Por último, se definen los sólidos de forma analítica también en Matlab. El motivo de esta definición es fundamentalmente obtener unos valores de masa, el primer momento y el tensor de inercia, que tengan sentido físico, de cara a realizar las simulaciones dinámicas.

De esta forma, se representarán visualmente los sólidos y se podrá obtener un modelo simplificado del real.

Para lograr esto, ha sido necesario modificar las funciones originales de la librería de funciones de Dinámica de sistemas multicuerpo, puesto que de lo contario, no seríamos capaces de almacenar los valores como simbólicos dentro de un vector concreto para tratar con ellos, y su manejo en las ecuaciones dinámicas sería sensiblemente más complicado.

La función modificada se llama “newSolid_without_STL”, y lo que en ella se ha modificado, es la forma en la que se introducen los nombres de bases, puntos y sistemas de referencia, así como de sólidos, para estandarizarlos y unificarlos. También esta función presentaba un problema, y es que no codificaba de forma estándar asociada al sólido las variables simbólicas creadas de masa, primeros momentos y tensor de inercia, además de que únicamente generaba aquellos no nulos.

Por ejemplo, si únicamente el sólido 2 tenía las componentes I11, I12, I22, I23, I33, generaba como salida i11, i12, i22, i23, i33; y si el sólido 3 tenía las mismas, generaba los mismos nombres (no podían ser almacenadas como pertenecientes a un sólido). El hecho de poder tener diferente número de componentes en los tensores de inercia asociados a cada sólido, dificultaba su gestión.

Con estos cambios realizados, podemos generar siempre parámetros inerciales asociados a cada sólido, con un nombre que los relaciona con el sólido en cuestión, y siempre tendrán dimensiones iguales. La gestión de estos valores es simbólica a nivel de ecuaciones, numérica a nivel de simulación.

La forma que se ha considerado óptima para gestionar la obtención de los parámetros inerciales del robot, es almacenarlos dentro de una estructura de Matlab, para poder extraer de forma independiente de ella, mediante llamadas por línea de comandos, uno o varios valores.

Para obtener los valores de los parámetros inerciales, se define una densidad y se dibujan sólidos simplificados por medio del programa OPENSCAD. Con los volúmenes generados y la densidad, se tiene la masa, y el programa es capaz de calcular los tensores de inercia y centro de gravedad.

Básicamente, se obtienen estructuras independientes para cada uno de los sólidos, donde podemos encontrar su sistema de referencia, sus parámetros inerciales, y el modelo simplificado:

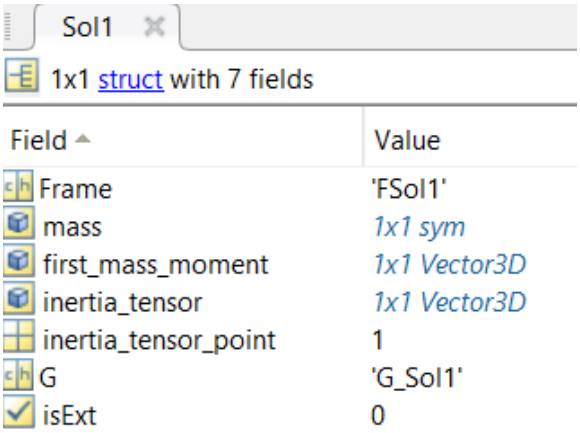


Fig 16: Estructura del sólido 1 (Sol1)

Si se teclea en línea de comandos el nombre el sólido, seguido de un punto y el nombre de la variable simbólica que queremos ver, el programa la devuelve por pantalla. Por ejemplo, si se escribe "Sol1.mass" o "Sol1.inertia_tensor", el programa entiende que se necesita esa información, y la muestra:

```
>> Sol1.mass

ans =

m_Sol1
```

```
>> Sol1.inertia_tensor

ans =

Ixx_Sol1, Ixy_Sol1, Ixz_Sol1
Ixy_Sol1, Iyy_Sol1, Iyz_Sol1
Ixz_Sol1, Iyz_Sol1, Izz_Sol1
'Base1'
```

Fig 17: Ejemplo de interacción con la estructura del sólido programado

De la imagen anterior se comprueba que los parámetros inerciales, reciben un nombre con sufijo relativo al sólido al que hacen referencia, como se ha comentado anteriormente. Además, en el caso de parámetros escritos en forma vectorial o matricial, el programa devuelve la base en la que éstos se expresen la forma que se obtiene.

En la página siguiente puede verse una comparativa entre ambos modelos, el representado de forma aproximada, y el real.

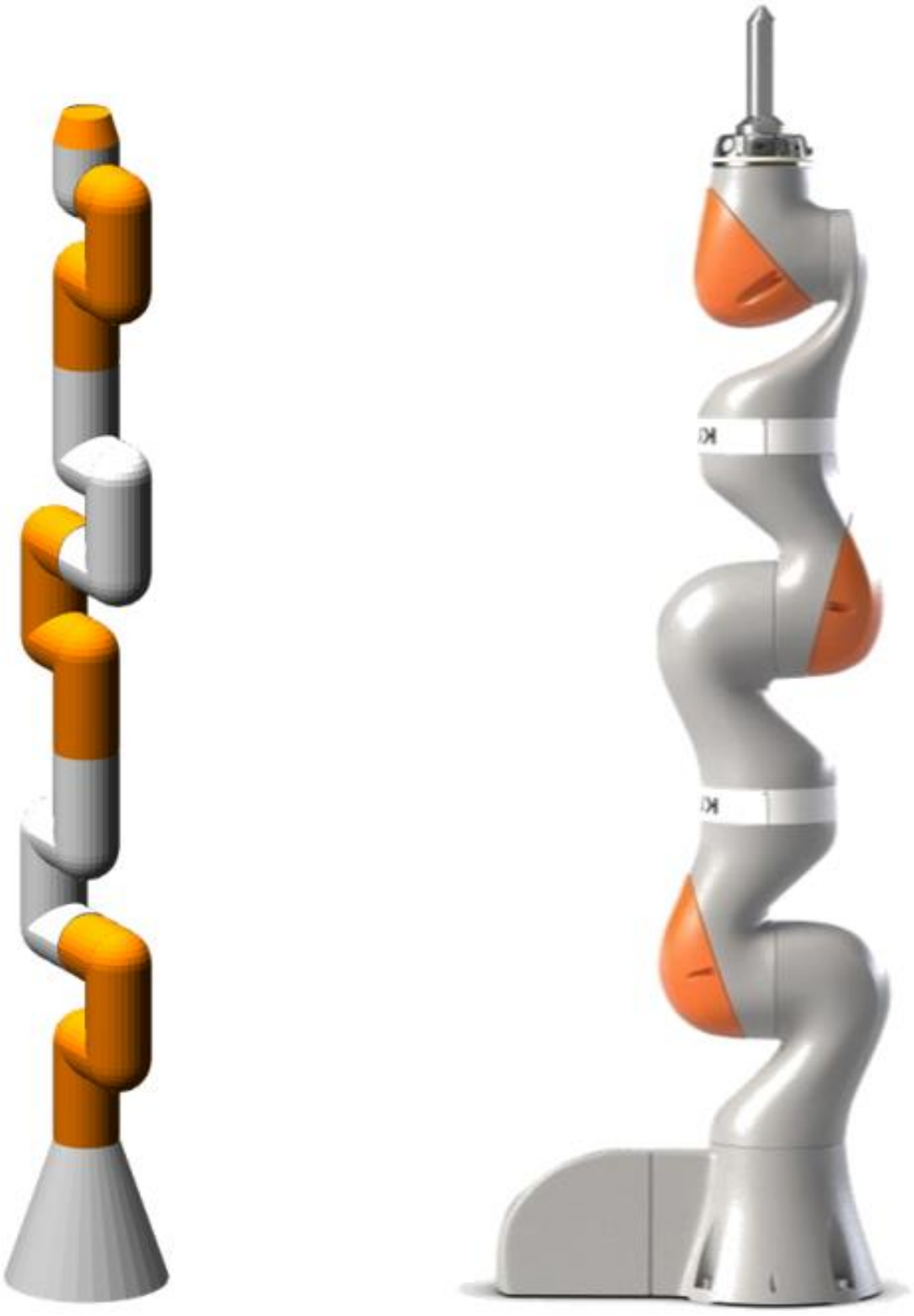


Fig 18: Comparativa entre robot simplificado, a la izquierda, y real, a la derecha

5 MODELO DINÁMICO

De forma análoga al capítulo anterior *Modelo cinemático*, el modelo dinámico puede entenderse como el estudio del movimiento, analizando las causas del mismo. Es decir, en este punto ya no es precisa la abstracción total respecto a las fuerzas y momentos ocasionadoras del movimiento en un sistema mecánico.

En este máster se ha trabajado desde una mirada vectorial, y se ha realizado una aproximación al problema dinámico puramente vectorial. Básicamente, para obtener las ecuaciones dinámicas, se aplican los principios básicos de la física (ámbito de la dinámica en sistemas mecánicos), de forma vectorial.

En el caso de los sólidos rígidos, las ecuaciones vectoriales son dos principalmente:

- La del momento lineal (Ecuaciones de Newton).
- La del momento angular o cinético (Ecuaciones de Euler).

Este conjunto de ecuaciones sirve para plantear el problema dinámico mediante el sistema de ecuaciones en notación Newton – Euler. Esta formulación es probablemente una de las más empleadas por los autores para generar modelos analíticos, básicamente porque funcionan muy bien con la acotación MDH. Una de las ventajas principales de este tipo de notación, es que el modelo dinámico, siempre será lineal con respecto a los parámetros inerciales del robot, que precisamente son algunos de los parámetros clave que se pretenden identificar.

Existe también el principio de las potencias virtuales, que puede enunciarse como: [9, p 36].

La suma de la potencia virtual de las acciones actuantes sobre un sistema (incluidas las acciones de inercia) es igual a cero. La potencia virtual de las acciones de enlace actuantes sobre el sistema es cero para velocidades virtuales compatibles con los enlaces. En este tipo de notación se trabaja con torsores, que no es más que el conjunto de fuerza y momento debidos a una acción concreta sobre un determinado sólido, como se verá más adelante.

Cada una de estas formulaciones tiene sus ventajas e inconvenientes, y son más o menos apropiadas para según qué aplicaciones. Parte del trabajo del autor, consiste en determinar cuál de ellas es idónea para el caso que se presenta en este proyecto, y aplicarla de forma eficiente.

El objetivo principal de este capítulo es dar forma final a este modelo analítico, aprovechando los pasos previamente dados, para poder determinar las acciones que actuarán sobre cada uno de los sólidos, definirlas analíticamente, integrarlas haciendo uso de la librería de funciones, y aplicando una de las formulaciones mencionadas, obtener las ecuaciones del sistema.

Por último, se escribirán las ecuaciones en un fichero, y se realizarán unas simulaciones para validar el modelo obtenido.

En los capítulos siguientes se detallan los pasos dados.

5.1 OBTENCIÓN DE LAS ECUACIONES DINÁMICAS

Este capítulo aborda la problemática básica de definir las ecuaciones dinámicas del robot, que conforman el modelo analítico, necesario para la construcción de las matrices que posteriormente se emplearán en la fase de identificación dinámica.

5.1.1 Principio de formulación elegido

Como se ha comentado con anterioridad, existen diferentes principios (en base a formulaciones) para construir modelos analíticos y aplicar los conceptos básicos de la dinámica.

En este proyecto se ha elegido la formulación de potencias virtuales, puesto que es una metodología que permite sistematizar las ecuaciones fácilmente. La principal ventaja de este tipo de formulación, es que si se escriben las ecuaciones cumpliendo ciertas condiciones, el modelo será lineal en los parámetros inerciales y de fricción.

Básicamente las condiciones principales que deben satisfacerse, son las que se enumeran a continuación:

- La posición del centro de gravedad a estimar, de cada uno de los sólidos, debe expresarse a través de los 3 primeros momentos de inercia del sólido en cuestión (que no es otra cosa que la masa por el vector que da las coordenadas del centro de gravedad en la base elegida, fija al sólido objeto de análisis). Básicamente se agrupa en diferentes parámetros, la masa y la posición del centro de gravedad, dónde este último, solo podrá ser estimado con certeza una vez se haya identificado y obtenido el valor de la masa.
- La referencia en la que se defina el torsor de inercia del sólido, se deberá definir en un punto cinemáticamente perteneciente al sólido, cuya ubicación sea conocida. El único punto no válido será el centro de gravedad, por los motivos previamente mencionados en el punto anterior, que forzosamente implican el desconocimiento de su ubicación real.

Realmente se busca identificar o estimar los parámetros inerciales y de fricción de cada uno de los sólidos, así como algunos adicionales. Las claves que se acaban de dar facilitan la construcción de un modelo adaptado y válido para tal efecto.

Pero, ¿Cómo se construyen las ecuaciones en una formulación concreta?

Las ecuaciones vienen dadas por un sumatorio de productos de torsores por velocidades virtuales [7, p 49]. En este trabajo se emplea la librería de funciones de Matlab de la asignatura de Dinámica de Sistemas Multicuerpo, para poder plantear las ecuaciones de forma directa.

En la tesis doctoral de Xabier Iriarte, la ecuación 2.6 describe la reescritura de las ecuaciones de Newton – Euler, a través de principio de las potencias virtuales:

$$\Gamma_k = \sum_{i=1}^{N_{Sol}} \left(\mathcal{F}_i \boldsymbol{V}_{ik}^* + \mathcal{M}_i \boldsymbol{\Omega}_{ik}^* \right) + \sum_{i=1}^{N_{Sol}} \sum_{j=1}^{N_{Acc}} \left(\boldsymbol{F}_{ij} \boldsymbol{V}_{ik}^* + \boldsymbol{M}_{ij} \boldsymbol{\Omega}_{ik}^* \right) = 0$$

Fig 19: Ecuación 2.6 de la tesis de Xabier Iriarte, donde se detalla la formulación de Potencias Virtuales [7]

Para más información, puede consultarse la tesis doctoral de Xabier Iriarte, debidamente referenciada en el apartado de *Bibliografía y referencias*.

Existe una forma de abordar el problema dinámico con el principio de potencias virtuales ligeramente modificado. Este modelo se conoce como Principio de las Potencias Virtuales Sistematizado.

En este modelo se definen tantas velocidades independientes como números de grados de libertad, de forma que se relacionan con las velocidades generalizadas del sistema. Este concepto permite llevar a la reescritura de las ecuaciones (que simplifican su tratamiento, puesto que el término asociado a los multiplicadores de Lagrange desaparece), mediante las siguientes etapas:

$$\mathbf{v} = [v_1, ..., v_n]^T \qquad \mathbf{M}_{\dot{\mathbf{q}}\dot{\mathbf{q}}}\ddot{\mathbf{q}} + \mathbf{V}_{\dot{\mathbf{q}}}^{\perp}\boldsymbol{\epsilon} = \mathbf{f}$$

$$\dot{\mathbf{q}} = \mathbf{B}\mathbf{v}$$

$$(\dot{\mathbf{q}}^v)^T \left(\mathbf{M}_{\dot{\mathbf{q}}\dot{\mathbf{q}}}\ddot{\mathbf{q}} + \mathbf{V}_{\dot{\mathbf{q}}}^{\perp}\boldsymbol{\epsilon} = \mathbf{f}\right) \qquad \mathbf{B}^T\mathbf{M}_{\dot{\mathbf{q}}\dot{\mathbf{q}}}\mathbf{B}\dot{\mathbf{v}} + \mathbf{B}^T\mathbf{V}_{\dot{\mathbf{q}}}^{\perp}\boldsymbol{\epsilon} = \mathbf{B}^T\mathbf{f}$$

$$\mathbf{M}_{\mathbf{vv}}\dot{\mathbf{v}} = \mathbf{B}^T\mathbf{f}$$

Fig 20: Formulación del Principio de Potencias Virtuales Sistematizado [9]

Una vez seleccionado el principio de formulación para la escritura y obtención de las ecuaciones dinámicas del robot, se puede entrar más a fondo en la dinámica que hay detrás de este robot, para definir las acciones sobre el mismo, y su comportamiento físico teórico como es debido.

Es importante recalcar que nos será posible saber si el modelo obtenido es correcto hasta realizar la identificación. Los robos de cadena abierta de muchos grados de libertad, son realmente complejos de modelar, y sobre todo de validar. Las ecuaciones, como se verá más adelante, son muy extensas y abstractas, poco intuitivas. Por estos motivos, se harán pruebas en situaciones conocidas para determinar la validez del modelo del robot.

Con esto el autor quiere explicar, que aunque posteriormente se compruebe que el modelo dinámico obtenido, no es del todo correcto, esto es algo natural y previsible. El problema fundamental en el trabajo fin de máster, es el tiempo que cuesta conseguir las ecuaciones, procesarlas y simplificarlas, para posteriormente conseguir el robot para realizar los experimentos. Pero en una situación de trabajo real, donde el tiempo no sea tan crítico como en este proyecto, el autor podría volver a revisar el modelo, con los datos obtenidos en la primera iteración de experimentos, que ayuden a dilucidar dónde se ha producido el error anteriormente, con el fin de subsanarlo y regenerar el modelo para una segunda iteración, donde posiblemente, los resultados sean más satisfactorios que en la primera.

A continuación se plantean los torsores que toman parte en la formulación del modelo, y posteriormente se plantean y obtienen las ecuaciones dinámicas.

5.1.2 Planteamiento de los torsores

El siguiente paso en el modelado dinámico, consiste en plantear los torsores del problema, de las acciones que experimente cada uno de los sólidos del sistema multicuerpo del robot.

Los torsores son unos "atajos" físicos, basados en conceptos mecánicos, que permiten la agrupación del conjunto de fuerzas y momentos relativos a una acción concreta. Por ejemplo el torsor viscoso en un enlace, agrupa la fuerza y momento que se experimenta en ese enlace (uno de los dos sólidos ejerce sobre el otro).

Es especialmente útil con la formulación de potencias virtuales, que como se ha explicado, es justamente la que se emplea en este proyecto. Básicamente se podría decir que con frecuencia, es conveniente tratar a las parejas Fuerza – Momento, como un solo bloque de acciones, que llamamos torsor (wrench en inglés).

De forma análoga, la pareja de velocidad lineal – velocidad angular, se denomina rotor (twist en inglés). Este rotor está estrechamente relacionado con el torsor, puesto que la multiplicación de torsor con rotor, lleva a una multiplicación escalar individual de fuerzas con velocidades lineales, y momentos con velocidades angulares, lo que provoca la obtención de la potencia del torsor (potencia desarrollada por una acción, sobre un sólido concreto).

Es importante no olvidarse de que no solamente existen y deben caracterizarse torsores de acciones externas sobre el robot. El torsor de inercia de D'Alembert es empleado para caracterizar en conjunto las acciones inerciales del sólido estudiado.

Ya se ha explicado en el capítulo de *Modelo cinemático*, que en este robot de cadena abierta no se requiere la escritura de ecuaciones de enlace geométrico, ni la definición de ecuaciones no holónomas para el planteamiento del problema cinemático.

Explicado esto puede deducirse que simplemente queda pasar al planteamiento de los torsores de las acciones sobre los sólidos.

Antes de comenzar creo que es importante explicar correctamente que inicialmente se habían escrito y definido los torsores de enlace en cada una de las articulaciones. Estos torsores se definen por medio de la función "newConstraint_Wrench" de la librería de funciones de Matlab, de la asignatura de Dinámica de Sistemas Multicuerpo. La definición es tal y como se muestra a continuación.

```
%% Wrench Definition

% Constraint Wrenches (Definition using helper function newConstraint_Wrench)


% Wrench_Ground_Base=newConstraint_Wrench([1,1,1],[1,1,1],'O','xyz','Base','Gr');
% Wrench_Base_Sol1=newConstraint_Wrench([1,1,1],[1,1,0],'P1','Base1','Sol1','Base');
% Wrench_Sol1_Sol2=newConstraint_Wrench([1,1,1],[1,0,1],'P2','Base2','Sol2','Sol1');
% Wrench_Sol2_Sol3=newConstraint_Wrench([1,1,1],[1,1,0],'P3','Base3','Sol3','Sol2');
% Wrench_Sol3_Sol4=newConstraint_Wrench([1,1,1],[1,0,1],'P4','Base4','Sol4','Sol3');
% Wrench_Sol4_Sol5=newConstraint_Wrench([1,1,1],[1,1,0],'P5','Base5','Sol5','Sol4');
% Wrench_Sol5_Sol6=newConstraint_Wrench([1,1,1],[1,0,1],'P6','Base6','Sol6','Sol5');
% Wrench_Sol6_Sol7=newConstraint_Wrench([1,1,1],[1,1,0],'P7','Base7','Sol7','Sol6');
```

Fig 21: Definición de os torsores de enlace entre sólidos (el cero indica el grado de libertad permitido en cada enlace)

La finalidad de estos torsores es definir los movimientos permitidos e impedidos en un enlace entre dos sólidos. Pues bien, tras haberlo hecho, en la conversación con el tutor del trabajo, se vio que la definición de estos torsores era innecesaria por ser redundante.

Lo que se vio con la explicación del profesor y director del trabajo, es que el robot lleva implícita la forma de movimiento permitida e impedida en la definición de las coordenadas generalizadas y sólidos, de la parametrización inicial. Básicamente no hace falta indicar por medio de un torsor que, por ejemplo, el sólido 3 no se traslada respecto al sólido 2 (solo rota en torno a su eje), básicamente porque no se ha definido ningún grado de libertad en esa dirección. Es decir, definir un torsor que especifique esto, es repetir algo que va implícito en la propia definición de las coordenadas generalizadas y la acotación del mecanismo.

Las direcciones en las que se tendrán fuerzas y momentos fruto de las restricciones de movimiento y giro en los enlaces, se obtendrán automáticamente de la definición del mecanismo.

Aclarado este hecho ya es posible plantear uno a uno el resto de torsores que intervienen en la dinámica del robot.

En primer lugar se plantean los torsores relativos a las acciones constitutivas. Como acción constitutiva puede entenderse cualquier respuesta de un material o sólido, provocada por la exposición del mismo a cargas externas. Básicamente se consideran para estudiar el efecto de las tensiones y deformaciones que dicha acción ha provocado sobre el material o cuerpos mencionados. Se les llama acciones constitutivas porque de alguna manera, describen el comportamiento de un cuerpo ante acciones externas, por cómo está internamente constituido dicho material.

En este caso concreto se han considerado como acciones constitutivas, las acciones relacionadas con los esfuerzos viscosos. Estos esfuerzos son los que aparecen en los enlaces del robot fruto del movimiento de los mismos. Es posiblemente la parte más compleja para definir y entender de todo el robot, puesto que son comportamientos difícilmente caracterizables y suelen de depender de parámetros desconocidos, que precisamente se pretende identificar para estimar su valor en el robot real.

Como apunte importante es necesario recalcar que existe la posibilidad de considerar dos tipos de acciones constitutivas, las asociadas a la rigidez de los enlaces entre sólidos, y las asociadas a la fricción de los enlaces entre sólidos.

Con respecto al primer tipo de acción constitutiva, se ha tomado la decisión de no caracterizarlas ni considerarlas en el modelado dinámico del robot. Estas acciones toman en consideración las fuerzas existentes en un enlace entre dos cuerpos del robot, y guardan relación con el ángulo indicado a la entrada para el giro de una articulación, y el ángulo medido a la salida de esa cadena cinemática aislada, dentro del conjunto global del mecanismo.

Para entenderlo mejor, podría pensarse en un muelle torsional, que genera unas fuerzas de reacción ante la deformación angular que sufre por mover los dos elementos que soporta. Sucede algo similar en las articulaciones del robot, dentro de las cuales existe una reductora que es la encargada de transformar la velocidad de giro del motor que acciona esa articulación en concreto (alta velocidad) en una velocidad angular de trabajo adecuada para la dinámica del robot (baja velocidad).

Dicha reductora tiene una rigidez teórica (por diseño y cálculo) y una rigidez real (por desgastes, engrases, imperfecciones), que afecta directamente al comportamiento del robot.

Todo esto plantea la problemática de modelar o no el efecto asociado a esta parte de la cadena cinemática del robot. Para resolver esta cuestión, se detectó la necesidad de contar con un sistema de doble encoder en cada articulación del robot, para poder determinar en cada instante de simulación en ángulo indicado a la entrada y a la salida de la reductora.

Con estos datos se contactó con la empresa ALDAKIN, propietaria del robot, y se resolvió la duda sobre la posibilidad o no de modelar este efecto dinámico. La respuesta por parte de ALDAKIN, fue que el robot únicamente cuenta con un encoder (sistema de medición angular instantánea) en cada articulación.

Ahora la problemática que surge es la siguiente. El dato que otorga ese encoder único en cada enlace, deberá ser tomado como real y definitivo, pero este hecho no será del todo cierto dependiendo de la posición de la articulación.

Si el encoder se encuentra a la entrada de la reductora, o lo que es lo mismo, a la salida del motor que acciona el enlace, el dato del ángulo medido no será del todo válido, y se cometerá un ligero error en la identificación posterior. Esto se debe al hecho anteriormente comentado de ángulo real girado a la salida de la reductora, y ángulo teóricamente indicado por el sistema de accionamiento a la entrada. Podría decirse que se tiene un ligero error en los resultados, fruto de la no modelización de la reductora presente en el enlace.

Si por el contrario el enconder estuviera ubicado a la salida de la reductora, el ángulo medido y obtenido en el proceso, podría tomarse como válido con error mínimo, y el no modelar la reductora, tendría mucho menos impacto en la veracidad de los resultados obtenidos en la identificación dinámica.

Así pues, tras esta breve explicación de la no modelización de la reductora, se procede a detallar la caracterización y definición de torsores de las acciones constitutivas sobre el robot.

Las acciones constitutivas que se detallan a continuación tienen que ver con los modelos de fricción. De forma breve, se ha de tener en cuenta que para la validación del modelo y simulaciones dinámicas, se modeló exclusivamente un modelo de fricción viscosa. Este modelo contempla la aparición de fuerzas y/o momentos de acción reacción, entre dos sólidos unidos por un enlace. Si el par cinemático es de revolución, como es este caso, los pares generados en los enlaces, son proporcionales a la derivada temporal de las coordenadas generalizadas que definen el movimiento relativo entre los dos sólidos unidos. Dicho de otra forma, cuanto mayor sea la velocidad de movimiento entre ambos sólidos, mayores serán las acciones generadas de fricción viscosa en el enlace. Dicho modelo se rige por la siguiente ecuación:

$$M_{v,i}^{Pi} = -\Phi_{k,v}^{M} * \dot{q}_{k,\theta} * w_k$$

Donde $\Phi_{k,v}^{M}$ es el coeficiente de fricción viscosa del par cinemático "k", w_k dicta la dirección del movimiento relativo y de aparición de la fuerza o momento; y $\dot{q}_{k,\theta}$ hace referencia a la velocidad existente en el movimiento relativo entre los dos sólidos enlazados.

Como ya se ha explicado, las simulaciones dinámicas de validez del modelo, y la caracterización inicial, únicamente consideraban acciones constitutivas de fricción viscosa.

Posteriormente, se tomó la decisión de modelizar mejor las acciones constitutivas del mecanismo, y se amplió la caracterización de los torsores de estas acciones, modelizando un modelo de fricción conjunta viscosa – Coulomb.

La diferencia principal entre el modelo anterior y este nuevo modelo, es que el modelo de fricción de Coulomb considera la existencia de un momento de enlace permanente siempre que exista movimiento relativo entre los sólidos enlazados. Es un modelo que complementa muy bien el anterior, funciona exitosamente en la modelización de robots. Dicho modelo se rige por la siguiente ecuación:

$$M_{C,i}^{Pi} = -\Phi_{k,C}^{M} * M_C * sign\left(\dot{q}_{k,\theta}\right) * u_k$$

Donde M_C es una magnitud constante y desconocida, y por ello, se agrupa para el análisis e identificación el producto $\Phi_{k,C}^{M} * M_C$. La única variación en el torsor constante fruto del giro relativo entre sólidos en la articulación estudiada del robot, es que el signo de la velocidad relativa, determina el signo del par originado.

A continuación se muestra un extracto del código de modelado, en el que se definen los torsores de estas acciones:

```
% Constitutive
%----Begin Edit----
% Viscous Base->Sol1 (rolling)
% theta1_eq=newParam('theta1_eq',0);
k_Base_Sol1 =newParam('k_Base_Sol1',300);
c_Base_Sol1=newParam('c_Base_Sol1',0.5);
cc1=newParam('cc1',1);
sign_dt1=newParam('sign_dt1',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Base_Sol1=Vector3D([0,0,0]','Base1');
MV_Base_Sol1_P1=Vector3D([0,0,(-c_Base_Sol1*dtheta1-cc1*sign_dt1)]','Base1'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Base_Sol1_P1=Vector3D([0,0,(-c_Base_Sol1*dtheta1-cc1*sign(dtheta1))]','Base1'); %OPCIÓN 2 (COMENTAR/DESCO


WrenchV_Base_Sol1=Vector6D(FV_Base_Sol1,MV_Base_Sol1_P1,'P1','Sol1'); %ACTION
WrenchV_Sol1_Base=Vector6D(-FV_Base_Sol1,-MV_Base_Sol1_P1,'P1','Base'); %REACTION

% Viscous Sol1->Sol2 (rolling)
k_Sol1_Sol2 =newParam('k_Sol1_Sol2',300);
c_Sol1_Sol2=newParam('c_Sol1_Sol2',0.05);
cc2=newParam('cc2',1);
sign_dt2=newParam('sign_dt2',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Sol1_Sol2=Vector3D([0,0,0]','Base2');
MV_Sol1_Sol2_P2=Vector3D([0,(-c_Sol1_Sol2*dtheta2-cc2*sign_dt2),0]','Base2'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol1_Sol2_P2=Vector3D([0,(-c_Sol1_Sol2*dtheta2-cc2*sign(dtheta2)),0]','Base2'); %OPCIÓN 2 (COMENTAR/DESCO


WrenchV_Sol1_Sol2=Vector6D(FV_Sol1_Sol2,MV_Sol1_Sol2_P2,'P2','Sol2'); %ACTION
WrenchV_Sol2_Sol1=Vector6D(-FV_Sol1_Sol2,-MV_Sol1_Sol2_P2,'P2','Sol1'); %REACTION


% Viscous Sol2->Sol3 (rolling)
k_Sol2_Sol3 =newParam('k_Sol2_Sol3',300);
c_Sol2_Sol3=newParam('c_Sol2_Sol3',0.5);
cc3=newParam('cc3',1);
sign_dt3=newParam('sign_dt3',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Sol2_Sol3=Vector3D([0,0,0]','Base3');
MV_Sol2_Sol3_P3=Vector3D([0,0,(-c_Sol2_Sol3*dtheta3-cc3*sign_dt3)]','Base3'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol2_Sol3_P3=Vector3D([0,0,(-c_Sol2_Sol3*dtheta3-cc3*sign(dtheta3))]','Base3'); %OPCIÓN 2 (COMENTAR/DESCO

WrenchV_Sol2_Sol3=Vector6D(FV_Sol2_Sol3,MV_Sol2_Sol3_P3,'P3','Sol3'); %ACTION
WrenchV_Sol3_Sol2=Vector6D(-FV_Sol2_Sol3,-MV_Sol2_Sol3_P3,'P3','Sol2'); %REACTION
```

```
% Viscous Sol3->Sol4 (rolling)
k_Sol3_Sol4 =newParam('k_Sol3_Sol4',300);
c_Sol3_Sol4=newParam('c_Sol3_Sol4',0.05);
cc4=newParam('cc4',1);
sign_dt4=newParam('sign_dt4',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Sol3_Sol4=Vector3D([0,0,0]','Base4');
MV_Sol3_Sol4_P4=Vector3D([0,(-c_Sol3_Sol4*dtheta4-cc4*sign_dt4),0]','Base4'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol3_Sol4_P4=Vector3D([0,(-c_Sol3_Sol4*dtheta4-cc4*sign(dtheta4)),0]','Base4'); %OPCIÓN 2 (COMENTAR/DESCO

WrenchV_Sol3_Sol4=Vector6D(FV_Sol3_Sol4,MV_Sol3_Sol4_P4,'P4','Sol4'); %ACTION
WrenchV_Sol4_Sol3=Vector6D(-FV_Sol3_Sol4,-MV_Sol3_Sol4_P4,'P4','Sol3'); %REACTION


% Viscous Sol4->Sol5 (rolling)
k_Sol4_Sol5 =newParam('k_Sol4_Sol5',300);
c_Sol4_Sol5=newParam('c_Sol4_Sol5',0.5);
cc5=newParam('cc5',1);
sign_dt5=newParam('sign_dt5',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Sol4_Sol5=Vector3D([0,0,0]','Base5');
MV_Sol4_Sol5_P5=Vector3D([0,0,(-c_Sol4_Sol5*dtheta5-cc5*sign_dt5)]','Base5'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol4_Sol5_P5=Vector3D([0,0,(-c_Sol4_Sol5*dtheta5-cc5*sign(dtheta5))]','Base5'); %OPCIÓN 2 (COMENTAR/DESCO

WrenchV_Sol4_Sol5=Vector6D(FV_Sol4_Sol5,MV_Sol4_Sol5_P5,'P5','Sol5'); %ACTION
WrenchV_Sol5_Sol4=Vector6D(-FV_Sol4_Sol5,-MV_Sol4_Sol5_P5,'P5','Sol4'); %REACTION

% Viscous Sol5->Sol6 (rolling)
k_Sol5_Sol6 =newParam('k_Sol5_Sol6',300);
c_Sol5_Sol6=newParam('c_Sol5_Sol6',0.05);
cc6=newParam('cc6',1);
sign_dt6=newParam('sign_dt6',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Sol5_Sol6=Vector3D([0,0,0]','Base6');
MV_Sol5_Sol6_P6=Vector3D([0,(-c_Sol5_Sol6*dtheta6-cc6*sign_dt6),0]','Base6'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol5_Sol6_P6=Vector3D([0,(-c_Sol5_Sol6*dtheta6-cc6*sign(dtheta6)),0]','Base6'); %OPCIÓN 2 (COMENTAR/DESCO

WrenchV_Sol5_Sol6=Vector6D(FV_Sol5_Sol6,MV_Sol5_Sol6_P6,'P6','Sol6'); %ACTION
WrenchV_Sol6_Sol5=Vector6D(-FV_Sol5_Sol6,-MV_Sol5_Sol6_P6,'P6','Sol5'); %REACTION


% Viscous Sol6->Sol7 (rolling)
k_Sol6_Sol7 =newParam('k_Sol6_Sol7',300);
c_Sol6_Sol7=newParam('c_Sol6_Sol7',0.5);
cc7=newParam('cc7',1);
sign_dt7=newParam('sign_dt7',0); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)

FV_Sol6_Sol7=Vector3D([0,0,0]','Base7');
MV_Sol6_Sol7_P7=Vector3D([0,0,(-c_Sol6_Sol7*dtheta7-cc7*sign_dt7)]','Base7'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol6_Sol7_P7=Vector3D([0,0,(-c_Sol6_Sol7*dtheta7-cc7*sign(dtheta7))]','Base7'); %OPCIÓN 2 (COMENTAR/DESCO

WrenchV_Sol6_Sol7=Vector6D(FV_Sol6_Sol7,MV_Sol6_Sol7_P7,'P7','Sol7'); %ACTION
WrenchV_Sol7_Sol6=Vector6D(-FV_Sol6_Sol7,-MV_Sol6_Sol7_P7,'P7','Sol6'); %REACTION
```

Fig 22: Definición de los torsores de acciones constitutivas

De forma resumida, el planteamiento es el siguiente. Definir el parámetro constante a identificar de fricción viscosa en cada enlace "c_Sol1_Sol2", y definir el parámetro constante de fricción de Coulomb que agrupa $\Phi_{k,C}^{M} * M_C$, por ejemplo "cc1". Posteriormente se define el parámetro "sign_dt1", para realizar simulaciones (OPCIÓN 1). Para la obtención de las ecuaciones deberá comentarse el código asociado a la opción 1 (simulación), y descomentar el código asociado a la opción 2 (obtención de las ecuaciones).

Las fuerzas en este caso son inexistentes, y los pares generados son una combinación de ambos efectos, que tienden a oponerse al movimiento comandado por el motor de la articulación en cada uno de los enlaces. Posteriormente se crea la variables que contempla esta agrupación de fuerza y momento, torsor ("WrenchV_Sol1_Sol2", por ejemplo).

Para continuar, se modelizan los torsores asociados a la acción de cada uno de los motores presentes en cada enlace.

Básicamente los motores se modelizan como un par de entrada en cada articulación, que mueve un sólido respecto a otro. Para incluir esta modelización en las ecuaciones, se ha creado en cada enlace un parámetro, "Mi", actuante en el grado de libertad "i", que simboliza el par que introduce el conjunto motor reductora en cada enlace. Se define la dirección en cada uno de los enlaces, y se construye el torsor (únicamente par, no hay fuerzas lineales).

Para las simulaciones ese parámetro tomará el valor que se le indique, peor en las ecuaciones aparecerá como una variable simbólica.

A continuación se muestra un extracto del código de modelado, en el que se definen los torsores de estas acciones:

```
% Motors

% Motor Base->Sol1
% syms 'M1' 'real'
newParam('M1',0);
FMot_Base_Sol1=Vector3D([0,0,0]','Base1');
MMot_Base_Sol1_P1=Vector3D([0,0,M1]','Base1');%opcion1

WrenchMot_Base_Sol1=Vector6D(FMot_Base_Sol1,MMot_Base_Sol1_P1,'P1','Sol1'); %ACTION
WrenchMot_Sol1_Base=Vector6D(-FMot_Base_Sol1,-MMot_Base_Sol1_P1,'P1','Base'); %REACTION


% Motor Sol1->Sol2
% syms 'M2' 'real'
newParam('M2',0);
FMot_Sol1_Sol2=Vector3D([0,0,0]','Base2');
MMot_Sol1_Sol2_P2=Vector3D([0,M2,0]','Base2');%opcion1

WrenchMot_Sol1_Sol2=Vector6D(FMot_Sol1_Sol2,MMot_Sol1_Sol2_P2,'P2','Sol2'); %ACTION
WrenchMot_Sol2_Sol1=Vector6D(-FMot_Sol1_Sol2,-MMot_Sol1_Sol2_P2,'P2','Sol1'); %REACTION

% Motor Sol2->Sol3
% syms 'M3' 'real'
newParam('M3',0);
FMot_Sol2_Sol3=Vector3D([0,0,0]','Base3');
MMot_Sol2_Sol3_P3=Vector3D([0,0,M3]','Base3');%opcion1

WrenchMot_Sol2_Sol3=Vector6D(FMot_Sol2_Sol3,MMot_Sol2_Sol3_P3,'P3','Sol3'); %ACTION
WrenchMot_Sol3_Sol2=Vector6D(-FMot_Sol2_Sol3,-MMot_Sol2_Sol3_P3,'P3','Sol2'); %REACTION


% Motor Sol3->Sol4
% syms 'M4' 'real'
newParam('M4',0);
FMot_Sol3_Sol4=Vector3D([0,0,0]','Base4');
MMot_Sol3_Sol4_P4=Vector3D([0,M4,0]','Base4');%opcion1

WrenchMot_Sol3_Sol4=Vector6D(FMot_Sol3_Sol4,MMot_Sol3_Sol4_P4,'P4','Sol4'); %ACTION
WrenchMot_Sol4_Sol3=Vector6D(-FMot_Sol3_Sol4,-MMot_Sol3_Sol4_P4,'P4','Sol3'); %REACTION
```

```
% Motor Sol4->Sol5
% syms 'M5' 'real'
newParam('M5',0);
FMot_Sol4_Sol5=Vector3D([0,0,0]','Base5');
MMot_Sol4_Sol5_P5=Vector3D([0,0,M5]','Base5');%opcion1

WrenchMot_Sol4_Sol5=Vector6D(FMot_Sol4_Sol5,MMot_Sol4_Sol5_P5,'P5','Sol5'); %ACTION
WrenchMot_Sol5_Sol4=Vector6D(-FMot_Sol4_Sol5,-MMot_Sol4_Sol5_P5,'P5','Sol4'); %REACTION


% Motor Sol5->Sol6
% syms 'M6' 'real'
newParam('M6',0);
FMot_Sol5_Sol6=Vector3D([0,0,0]','Base6');
MMot_Sol5_Sol6_P6=Vector3D([0,M6,0]','Base6');%opcion1

WrenchMot_Sol5_Sol6=Vector6D(FMot_Sol5_Sol6,MMot_Sol5_Sol6_P6,'P6','Sol6'); %ACTION
WrenchMot_Sol6_Sol5=Vector6D(-FMot_Sol5_Sol6,-MMot_Sol5_Sol6_P6,'P6','Sol5'); %REACTION


% Motor Sol6->Sol7
% syms 'M7' 'real'
newParam('M7',0);
FMot_Sol6_Sol7=Vector3D([0,0,0]','Base7');
MMot_Sol6_Sol7_P7=Vector3D([0,0,M7]','Base7');%opcion1

WrenchMot_Sol6_Sol7=Vector6D(FMot_Sol6_Sol7,MMot_Sol6_Sol7_P7,'P7','Sol7'); %ACTION
WrenchMot_Sol7_Sol6=Vector6D(-FMot_Sol6_Sol7,-MMot_Sol6_Sol7_P7,'P7','Sol6'); %REACTION
```

Fig 23: Definición de los torsores de acciones de los conjuntos motor-reductora

Por último, se caracterizan los torsores de las últimas acciones que toman partido en la dinámica de este robot, que son los torsores de inercia y gravedad.

La definición de estos torsores la da Xabier Iriarte en su tesis doctoral [7, pp 56-58].

Los torsores de inercia:

$$\mathfrak{F}_i = -m_i \boldsymbol{a}(P_i)$$
$$\mathfrak{M}_i^{P_i} = -I_{P_i}(Sol_i)\boldsymbol{\alpha}(Sol_i) - m_i \overline{P_iG_i} \times \boldsymbol{a}(P_i)$$

Fig 24: Ecuaciones para la definición de los torsores de inercia [7]

Los torsores de las fuerzas de la gravedad:

$$\boldsymbol{F}_{\boldsymbol{g},i} = m_i \boldsymbol{g}$$
$$\boldsymbol{M}_{\boldsymbol{g},i}^{P_i} = m_i \overline{P_iG_i} \times \boldsymbol{g}$$

Fig 25: Ecuaciones para la definición de los torsores de fuerzas de la gravedad [7]

La caracterización y definición de estos torsores no requiere escritura en código, puesto que dichos torsores se caracterizan por medio de una función ya integrada en la librería de funciones de Matlab de la asignatura de Dinámica de Sistemas Multicuerpo.

Esta función toma el punto creado en la definición del sólido, y las características inerciales del mismo en su definición, para crear un vector 6D (Fuerzas y momentos, en 3 direcciones cada uno).

La caracterización de estos torsores no la hacemos directamente, pero es de las más sencillas de comprobar en cuanto a su validez, puesto que sus efectos en caso de estar bien o mal, se ven con facilidad en las simulaciones dinámica que luego se verán.

5.1.3 Planteamiento de las ecuaciones

Una vez se han descrito, planteado y definido todos los torsores involucrados en la dinámica del robot, es posible pasar a la confección de las ecuaciones dinámicas que rigen de forma analítica, el comportamiento del mismo.

El proceso es relativamente sencillo. El código que se ha preparado para obtención de las ecuaciones, contempla la posibilidad de trabajar con ecuaciones en ambas formulaciones previamente mencionadas (Newton – Euler y Potencias Virtuales).

Para ello, se plantea la suma de torsores de las acciones que se originan o sufren cada uno de los sólidos que conforman el robot. Pro hacerlo de forma genérica, y por si a futuro pretende usarse el código para otros proyectos, se ha confeccionado el código haciendo también el planteamiento de la suma de torsores de acciones sobre el sólido base (a futuro podría trasladarse sobre el suelo, pro ejemplo, para otros proyectos de la universidad).

En este planteamiento de suma de acciones sobre cada sólido, se plantea el problema dinámico y se obtienen las ecuaciones que lo gobiernan. De esta forma se obtienen las ecuaciones dinámicas en la formulación de Newton – Euler.

Para obtener las ecuaciones en la formulación de Potencias Virtuales, basta con realizar el producto escalar de todas las expresiones obtenidas en los sumatorios de torsores de acciones actuantes sobre los sólidos, con las componentes asociadas de velocidades en los “Twist”, para cada velocidad generalizada.

Tras esto tendremos las 7 ecuaciones asociadas a cada velocidad generalizada de los 7 grados de libertad.

En los siguientes extractos de códigos se ilustra y comenta de forma más clara esta idea.

```
% Base
Sum_Wrenches_Base_NE= WrenchIG_Base+WrenchV_Sol1_Base+WrenchMot_Sol1_Base; %+Wrench_Ground_Base-Wrench_Base_Sol1;

% Sol1
Sum_Wrenches_Sol1_NE=  WrenchIG_Sol1+WrenchV_Base_Sol1+WrenchV_Sol2_Sol1+WrenchMot_Base_Sol1+WrenchMot_Sol2_Sol1;

% Sol2
Sum_Wrenches_Sol2_NE=  WrenchIG_Sol2+WrenchV_Sol1_Sol2+WrenchV_Sol3_Sol2+WrenchMot_Sol1_Sol2+WrenchMot_Sol3_Sol2;

% Sol3
Sum_Wrenches_Sol3_NE=  WrenchIG_Sol3+WrenchV_Sol2_Sol3+WrenchV_Sol4_Sol3+WrenchMot_Sol2_Sol3+WrenchMot_Sol4_Sol3;

% Sol4
Sum_Wrenches_Sol4_NE=  WrenchIG_Sol4+WrenchV_Sol3_Sol4+WrenchV_Sol5_Sol4+WrenchMot_Sol3_Sol4+WrenchMot_Sol5_Sol4;

% Sol5
Sum_Wrenches_Sol5_NE=  WrenchIG_Sol5+WrenchV_Sol4_Sol5+WrenchV_Sol6_Sol5+WrenchMot_Sol4_Sol5+WrenchMot_Sol6_Sol5;

% Sol6
Sum_Wrenches_Sol6_NE=  WrenchIG_Sol6+WrenchV_Sol5_Sol6+WrenchV_Sol7_Sol6+WrenchMot_Sol5_Sol6+WrenchMot_Sol7_Sol6;

% Sol7
Sum_Wrenches_Sol7_NE=  WrenchIG_Sol7+WrenchV_Sol6_Sol7-WrenchMot_Sol6_Sol7; %+Wrench_Sol6_Sol7;
```

Fig 26: Suma de torsores sobre cada sólido. Paso inicial planteamiento de las ecuaciones

Como puede apreciarse, se plantean todo como una suma de torsores, porque los signos de las acciones están contemplados dentro del propio torsor. Se contemplan acciones – reacciones.

```
Dyn_eq_NE=[Sum_Wrenches_Base_NE;
    Sum_Wrenches_Sol1_NE;
    Sum_Wrenches_Sol2_NE;
    Sum_Wrenches_Sol3_NE;
    Sum_Wrenches_Sol4_NE;
    Sum_Wrenches_Sol5_NE;
    Sum_Wrenches_Sol6_NE;
    Sum_Wrenches_Sol7_NE]
```

Fig 27: Obtención de las ecuaciones dinámicas en formulación Newton - Euler

Como se observa en la imagen anterior, para obtener las ecuaciones en la formulación de Newton – Euler, basta con agrupar todas las expresiones vectoriales de suma de acciones y reacciones sobre cada sólido.

Sin embargo, para obtener las ecuaciones en formulación de Potencias Virtuales, se definen los twist asociados a los giros entre bases, y posteriormente se plantea el producto escalar anteriormente mencionado.

```
Dyn_eq_VP=sym(zeros(n_q,1));

Twist_Base = Twist('Base');
Twist_Sol1 = Twist('Sol1');
Twist_Sol2 = Twist('Sol2');
Twist_Sol3 = Twist('Sol3');
Twist_Sol4 = Twist('Sol4');
Twist_Sol5 = Twist('Sol5');
Twist_Sol6 = Twist('Sol6');
Twist_Sol7 = Twist('Sol7');

for i=1:n_q
    Dyn_eq_VP(i,1)=dot( Sum_Wrenches_Base_NE , diff( Twist_Base , dq(i) ) )+...
        dot( Sum_Wrenches_Sol1_NE , diff( Twist_Sol1 , dq(i) ) )+...
        dot( Sum_Wrenches_Sol2_NE , diff( Twist_Sol2 , dq(i) ) )+...
        dot( Sum_Wrenches_Sol3_NE , diff( Twist_Sol3 , dq(i) ) )+...
        dot( Sum_Wrenches_Sol4_NE , diff( Twist_Sol4 , dq(i) ) )+...
        dot( Sum_Wrenches_Sol5_NE , diff( Twist_Sol5 , dq(i) ) )+...
        dot( Sum_Wrenches_Sol6_NE , diff( Twist_Sol6 , dq(i) ) )+...
        dot( Sum_Wrenches_Sol7_NE , diff( Twist_Sol7 , dq(i) ) );
end

save Dyn_eq_VP Dyn_eq_VP
```

Fig 28: Obtención de las ecuaciones dinámicas en formulación de Potencias Virtuales

Por último, tras guardar las ecuaciones, se cargan y se almacenan bien diferenciadas unas de otras en un fichero .txt, que llamaremos “Dyn_eq_VP.txt”. Este paso es necesario básicamente porque no podemos obtener las ecuaciones por pantalla con Matlab (son extremadamente extensas y complejas). Hay que escribirlas en algún fichero para la posterior obtención de matrices previa a la identificación.

Por lo que respecta al modelo dinámico, únicamente resta validar el modelo analítico obtenido con las ecuaciones dinámicas en formulación de Potencias Virtuales, con alguna simulación dinámica.

5.1.4 Simulaciones dinámicas (validación del modelo preliminar)

Para las simulaciones dinámicas se han llevado a cabo una serie de pasos clave que permiten evaluar la validez del modelo con relativa facilidad y progresión.

Tras consultar con el director del trabajo, se vio que no hay formas claras y veraces de validar un modelo dinámico antes de realizar los experimentos y análisis de los datos, por lo que, para tratar de obtener una idea de la validez del modelo lo más real posible, se planteó la posibilidad de realizar simulaciones, con pocos sólidos, y luego con muchos, suponiendo que los brazos del robot se comportan como un péndulo.

La decisión fue esta porque los péndulos son elementos de comportamiento conocido en condiciones normales de trabajo, y además muy visual. Para ello únicamente se deben poner a cero todos los pares de los conjuntos motor – reductora de cada enlace, y dar unos valores ficticios para las simulaciones de coeficientes de fricción en los enlaces, suficientemente pequeños para que en la ventana gráfica, podamos ver movimiento relativo de los sólidos ante la acción de la gravedad sobre su propio peso.

Para estas simulaciones, se han planteado diferentes situaciones de partida, y se estudian primero para un robot de 3 sólidos (base y sólidos 1 y 2), después para uno de 5 sólidos (desde la base hasta el sólido 4); y finalmente para el robot completo.

El planteamiento de ir fraccionando las simulaciones en función del número de sólidos involucrados, es porque siempre es más sencillo, predecir y evaluar el comportamiento de un péndulo simple, que uno más complejo, como un péndulo triple o cuádruple, por ejemplo.

Las posiciones iniciales que dictaminan las posiciones de partida del robot, son:

- Ángulo entre sólidos 1 y 2 de:
    - 90º → Debería oscilar lentamente hasta detenerse, con el paso del tiempo, fruto de la fricción.
    - ≈0º (totalmente vertical, posición más alta, algo perturbada) → Debería bajar lentamente, acelerarse, y oscilar como en el caso anterior hasta detenerse.
    - 180º (posición más baja) → Debería permanecer inmóvil, la gravedad no ejerce momento en el enlace y todos los pares de accionamiento están desactivados (a cero).

Se han grabado unos vídeos de estas simulaciones, y se han analizado conjuntamente con el profesor y director del trabajo. Se han dado por válidos por obtener los resultados teóricamente correctos y esperados.

Esto no quiere decir que no puedan surgir errores en el modelo a futuro, tras realizar los experimentos; pero por lo menos queda claro que se ha revisado y analizado la veracidad del lo obtenido hasta este mundo, dentro de lo que es posible, con las herramientas disponibles.

Pueden consultarse los vídeos en los archivos adjuntados a continuación, donde se indican en el nombre el número de sólidos involucrados, y las posiciones iniciales. En caso de no poder visualizarlos correctamente, solicitarlos al autor.


5S-pi_2.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR


5S-0.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR


5S_pi.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR


3S-pi_2.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR


3S-pi.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR


3S-0.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR


8S-pi_2.mp4

SOLICITAR VÍDEO AL AUTOR Y REPRODUCIR

5.2 ESTRUCTURACIÓN SIMBÓLICA

En el presente capítulo se detallan de forma breve pero concreta, las técnicas y metodologías simbólicas que se han llevado a cabo con Matlab, para adaptar y mejorar las expresiones de las ecuaciones dinámicas y los resultados obtenidos hasta este momento con los modelados cinemático y dinámico.

Las técnicas que a continuación se muestran, son el punto de partida para obtener las matrices necesarias para la identificación dinámica posterior, que se dará más adelante.

Puede entenderse como un paso previo de preproceso, donde necesitamos adaptar la información existente, de forma que cumpla una serie de condiciones.

5.2.1 Simplificación de las ecuaciones

El primero de los pasos a dar, es simplificar las ecuaciones que ya han sido obtenidas. Por simplificación, se entiende la reescritura de las variables, parámetros y otros nombres, que son excesivamente largos en el código de Matlab, y que hacen muy pesado el tratamiento y procesamiento del fichero de texto que contiene las ecuaciones dinámicas resultantes.

Por ejemplo, en el caso de los parámetros geométricos, se ha trabajado con nombres de variables bastante largos, pero que en el código resultan útiles para identificar rápidamente de qué parámetro físico se trata, sin necesidad de acudir al glosario de nombres que maneja el diseñador, ni al catálogo del robot.

La pega de trabajar con nombres tan largos, es que cuando se construyen las matlabFunction, partiendo de las expresiones de las ecuaciones (muy extensas y complejas) que serán evaluadas posteriormente en varios instantes en la identificación, complica mucho y alarga de forma exponencial, la duración del proceso de obtención de estas funciones evaluables de Matlab.

Tras valorarlo en conjunto con el director del trabajo, se tomó la decisión de realizar un paso intermedio de reescritura y reformulación de las ecuaciones en un nuevo fichero de texto, para agilizar la obtención de las funciones “matlabFunction” de Matlab.

La forma elegida de renombrar las variables, sigue las siguientes normas:

- theta1 → t1
- theta2 → t2
- dtheta1 → t1
- m_Sol1 → m1
- mx_Sol1 → mx1
- c_Sol1_Sol2 → c2
- r1_base → r1b
- height_Sol1 → h1
- length_Sol1 → l1
- …etc.

Para ello se construyó un código que se ejecutó externamente mediante la ventana de comandos de Windows, con PowerShell, que entraba en el fichero que contenía las ecuaciones dinámicas originales, y realizaba las sustituciones de los nombres de las variables que el código indicaba.

Un pequeño extracto de este código se muestra a continuación:

```
Get-Content "C:\Users\Usuario\Desktop\Dyn_eq_VP.txt" | ForEach-Object { $_ -replace "theta1","t1" } |
ForEach-Object { $_ -replace "theta2","t2" } |
ForEach-Object { $_ -replace "theta3","t3" } |
ForEach-Object { $_ -replace "theta4","t4" } |
ForEach-Object { $_ -replace "theta5","t5" } |
ForEach-Object { $_ -replace "theta6","t6" } |
ForEach-Object { $_ -replace "theta7","t7" } |

ForEach-Object { $_ -replace "c_Base_Sol1","c1" } |
ForEach-Object { $_ -replace "c_Sol1_Sol2","c2" } |
ForEach-Object { $_ -replace "c_Sol2_Sol3","c3" } |
ForEach-Object { $_ -replace "c_Sol3_Sol4","c4" } |
ForEach-Object { $_ -replace "c_Sol4_Sol5","c5" } |
ForEach-Object { $_ -replace "c_Sol5_Sol6","c6" } |
ForEach-Object { $_ -replace "c_Sol6_Sol7","c7" } |

ForEach-Object { $_ -replace "r1_base","rb1" } |
ForEach-Object { $_ -replace "r2_base","rb2" } |
ForEach-Object { $_ -replace "r_in_base","rib" } |
ForEach-Object { $_ -replace "height_base","hb" } |

ForEach-Object { $_ -replace "r_out_sol1","ro1" } |
ForEach-Object { $_ -replace "r_in_sol1","ri1" } |
ForEach-Object { $_ -replace "height_sol1","h1" } |
ForEach-Object { $_ -replace "length_sol1","l1" } |

ForEach-Object { $_ -replace "r_out_sol2","ro2" } |
ForEach-Object { $_ -replace "r_in_sol2","ri2" } |
ForEach-Object { $_ -replace "height_sol2","h2" } |
ForEach-Object { $_ -replace "length_sol2","l2" } |

ForEach-Object { $_ -replace "r_out_sol3","ro3" } |
ForEach-Object { $_ -replace "r_in_sol3","ri3" } |
ForEach-Object { $_ -replace "height_sol3","h3" } |
ForEach-Object { $_ -replace "length_sol3","l3" } |

ForEach-Object { $_ -replace "r_out_sol4","ro4" } |
ForEach-Object { $_ -replace "r_in_sol4","ri4" } |
ForEach-Object { $_ -replace "height_sol4","h4" } |
ForEach-Object { $_ -replace "length_sol4","l4" } |
```

Fig 29: Script de PowerShell para reescritura de nombres de variables en el fichero de ecuaciones dinámicas

5.2.2 Técnicas simbólicas

Este apartado de técnicas simbólicas, únicamente recoge las operaciones en notación simbólica que se han programado en los códigos de Matlab, para optimizar la forma en la que se construyen las matrices y funciones.

El problema principal que se detectó inicialmente con la ejecución del código que construía las funciones y matrices, partiendo de las ecuaciones dinámicas, era el tiempo de simulación. El código estuvo simulando 3 días enteros y no llegó a terminar; siendo detenido por el autor del trabajo.

Ello llevó en primer lugar a una reescritura de las ecuaciones, simplificando y aligerando el tratamiento del fichero; ya explicado en *Simplificación de las ecuaciones*; pero también, fueron necesarias técnicas adicionales que aligeraran la carga computacional del proceso en sí.

Las técnicas simbólicas empleadas, son las que se muestran a continuación:

- División del fichero simbólico en 2: Esta técnica es bien sencilla, y consiste en fragmentar el código simbólico en dos diferentes. En el primero de ellos, se escriben las ecuaciones del fichero que las agrupa, de forma independiente, y se le asigna un nombre a cada ecuación, para poder tratarlas de forma independiente o agruparlas a posteriori. Esto se hace para aligerar el otro código, que es el que se simula y llama a este primero (que es especialmente pesado y complejo de manejar, por la gran cantidad de texto que maneja). Este primer fichero es “Dynamic_Equations.m”.

  El segundo fichero, es “get_symbolic_K_symplify.m”, que contiene la segunda de las técnicas simbólicas, y que básicamente obtiene las dos matlabFunction necesarias para realizar la identificación dinámica posterior.

- Reducción del coste computacional de generación de funciones: El proceso de obtención de las matlabFunction necesarias, es realmente largo y tedioso. Gracias a la orientación y experiencia del director del máster, se halló una forma de realizar este paso de forma óptima y eficiente, reduciendo considerablemente la duración de las simulaciones. La técnica en cuestión, consiste en lo siguiente:

  Para sacar el jacobiano de “Eq_Dyn” con respecto a “param” (parámetros inerciales y de fricción a estimar), se ha hecho lo siguiente:

  - Crear un vector de parámetros que tenga todos menos el parámetro i.
  - Sustituir en el vector “Eq_Dyn” ese vector por ceros (así sólo aparecerá el parámetro i.
  - Derivar parcialmente el resultado respecto del parámetro i.
  - Guardar el vector columna resultante en la columna i-ésima de K.
  - Una vez hecho esto con todos los parámetros: crear la matlabFunction.

  Básicamente lo que se consigue es una confección del jacobiano parámetro a parámetro, con una matriz repleta de ceros salvo lo interesante en cada momento, para aligerar mucho el proceso. Esto se hace mediante un bucle, que se muestra en el siguiente extracto con más detenimiento.

```
Dynamic_Equations
%  symbolic definition of the coordinates and their derivatives
syms t1 t2 t3 t4 t5 t6 t7 real
syms dt1 dt2 dt3 dt4 dt5 dt6 dt7 real
syms ddt1 ddt2 ddt3 ddt4 ddt5 ddt6 ddt7 real
    q=[t1 t2 t3 t4 t5 t6 t7]';
    dq=[dt1 dt2 dt3 dt4 dt5 dt6 dt7]';
    ddq=[ddt1 ddt2 ddt3 ddt4 ddt5 ddt6 ddt7]';

%  symbolic definition of the sign of the generalized velocities
syms s1 s2 s3 s4 s5 s6 s7 real
s_dq=[s1 s2 s3 s4 s5 s6 s7]';

%  symbolic definition of the input torques
syms M1 M2 M3 M4 M5 M6 M7 real
    torques=[M1 M2 M3 M4 M5 M6 M7]';

%  symbolic definition of the geometric parameters
syms rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3 ri3 h3 l3 ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri7 r1o7 r2o7
%  symbolic definition of the gravity constant
syms g real
    param_geom=[rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3 ri3 h3 l3 ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri


%  symbolic definition of the inercial parameters
syms m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 m2 mx2 my2 mz2 Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3
    param=[ m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 m2 mx2 my2 mz2 Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 m

load Eq Dyn.mat
for i=1:length(param)
    tic
    param_i=[param(1:i-1,1);param(i+1:end)];
    fprintf(['Subsitute all params but ',num2str(i),' by 0\n']);
    Eq_i=subs(Eq_Dyn,param_i,zeros(size(param_i)));
    fprintf('Simplifying Eq_Dyn\n');
    Eq_i=simplify(Eq_i);
    fprintf('Calculating Ki\n');
    Ki=jacobian(Eq_i,param(i,1));
    fprintf('Simplifying Ki\n');
    Ki=simplify(Ki);
    K(:,i)=Ki;
    save K K
    toc
end

tau= - subs(Eq_Dyn,param,zeros(size(param)));
save tau tau;

tic,fprintf('Generating evalK\n');
matlabFunction(K,'File','evalK','vars',{q,dq,ddq,s_dq,param_geom})
toc

tic,fprintf('Generating evaltau\n');
matlabFunction(tau,'File','evaltau','vars',{q,torques})
toc
```

Fig 30: Extracto del código donde se aplican las técnicas simbólicas mencionadas

5.2.3 Generación de funciones para el problema de identificación

Este subcapítulo es muy breve ya que únicamente se centra en la generación de las funciones necesarias para la posterior identificación dinámica.

Lo primero es tener claro qué se necesita, y para ello se acude a los apuntes de la asignatura de Identificación de Sistemas Dinámicos, del Máster.

En ellos se explica claramente que el problema de identificación habitual, se plantea en forma matricial de la siguiente manera.

Partiendo de los conceptos básicos de regresión lineal y estadística vistos en el máster [10], es demostrable que :

$$y_m = y + \varepsilon = K(x)\phi + \varepsilon$$

Donde "x" se mide sin error, "K" es una matriz que depende de "x", "ϕ" es el vector de parámetros, y "ε" representa el error al medir "y".

En este caso, de las ecuaciones que rigen el sistema, se realiza el jacobiano para obtener la matriz que contiene todo lo que multiplica a los parámetros a estimar, "K". Esa matriz, ha de ser evaluada a cada instante, pues representa todo lo que podremos conocer en el tiempo, y que guarda relación con los parámetros a estimar. La evaluación temporal de K se registra, como filas de una matriz global, "W", que se usará para la estimación.

Por último, las mediciones realizadas en los experimentos (Ym), nosotros la llamamos "τ", vector que contiene los pares en ellos enlaces en cada momento, por ejemplo. Son datos que podemos extraer de la sensórica que integra el robot.

Todo ello servirá para construir un sistema tal que, de forma simplificada:

$$\tau = W * \phi$$

Por ello se deberán obtener 2 matlabFunction, "evalK" y "evalTau", que serán funciones simbólicas a evaluar numéricamente para tener valores en cada experimento realizado, que sirvan para construir las matrices que se emplean en la identificación dinámica.

El código asociado, puede verse en la figura *Fig 30: Extracto del código donde se aplican las técnicas simbólicas mencionadas*.

6 DISEÑO DE EXPERIMENTOS

En este capítulo se lleva a cabo el diseño de los experimentos que se requieren para poder hacer la identificación dinámica con el robot. Básicamente, ésta es una etapa de preparación y adecuación del modelo y las trayectorias, que el robot real debe describir.

La idea que subyace es emplear los modelos analíticos creados, combinándolos con los datos registrados en los experimentos (trayectorias indicadas y realizadas por el robot). Fusionando ambos conceptos se alcanzarán resultados de estimación de parámetros que queremos estimar, y posteriormente deberá comprobarse la validez de los resultados obtenidos con esta idea.

Lo primero y más fundamental, será llevar a cabo una reducción del modelo con alguna de las metodologías y técnicas existentes.

Lo que sucede es que, cuando se pretende llevar a cabo la identificación de los parámetros inerciales del robot, hay que hacer unos ajustes en el modelo analítico original que se ha obtenido, para asegurar que se va a poder realizar la estimación como es debido.

Los procedimientos existentes para llevar a cabo esta tarea, dan como resultado modelos que dependen de menos parámetros que el modelo analítico original. Por ello esa re-parametrización del modelo suele denominarse como “reducción del modelo”.

Posteriormente será necesario parametrizar unas trayectorias cualesquiera, para poder ensamblar la matriz “W” en función de unos instantes preestablecidos discretizando la trayectoria, y así poder obtener las expresiones de los parámetros base con un código muy sencillo.

A continuación, habrá que dar paso a un proceso de optimización de trayectorias, puesto que ya no vale con cualquier trayectoria que creemos, sino que habrá que buscar unas trayectorias que cumplan una serie de requisitos para que el robot pueda hacerlas correctamente, y que a su vez nos proporcionen datos válidos y de interés.

En esta parte del diseño de experimentos, se nombrarán y explicarán algunas técnicas adoptadas y decisiones tomadas, con el fin de especificar cada uno de los puntos pasos que se han ido dando en el proyecto.

Para finalizar con el diseño de experimentos, se crearán para las trayectorias óptimas obtenidas, unas trayectorias de enlace, de arranque y parada, que básicamente conectan la posición de reposo inicial (robot descansando en su posición más estirada), con la posición de partida de la trayectoria óptima; y la posición final de la trayectoria óptima, con la posición de reposo final (robot descansando en su posición más estirada).

Una vez diseñados los experimentos, se deberían haber llevado a cabo en las instalaciones de ALDAKIN, y con los resultados obtenidos podrá dar lugar la fase de identificación dinámica.

6.1 REDUCCIÓN DEL MODELO

Después de plantear las fases del diseño de experimentos, se comienza con la reducción del modelo a parámetros base. Ya se ha explicado previamente en qué consiste esta técnica de identificación dinámica, esa re-parametrización necesaria para asegurar la identificabilidad del modelo.

Previamente se ha obtenido la función de Matlab, o matlabFunction, que permite evaluar K numéricamente. Aprovechando este concepto, al escribir las ecuaciones de la dinámica inversa en "n" instantes de tiempo, y ensamblando cada uno de esos instantes en una matriz como filas (cada fila corresponde a una evaluación de "K" para un instante concreto), se obtiene la matriz de observación, que se suele denominar con la letra "W".

El motivo de la reducción del modelo a menos parámetros (parámetros base), donde muchos de estos nuevos parámetros, son combinaciones de dos o más parámetros anteriormente independientes.

La necesidad de esta reducción se debe a que cuando se evalúa "K" en "n" instantes (generalmente bastantes instantes), la matriz "W" que se obtiene, tiende a ser deficiente en rango (o lo que es lo mismo, no es de rango máximo). Este hecho se traduce en que haya columnas, que sean combinaciones lineales de otras columnas, provocando que sea imposible estimar los parámetros asociados a esas columnas.

Existen diferentes metodologías para abordar el problema de la reducción del modelo a parámetros base, cada una con sus ventajas e inconvenientes. Los dos métodos más comúnmente empleados son el método de las columnas independientes (no muy eficiente desde el punto de vista computacional), y el método QR, que es el que en este caso emplearemos [10].

Este proyecto no tiene en ningún caso como objetivo explicar las diferentes técnicas y opciones que hay para realizar la identificación dinámica, pero sí pretende explicar las técnicas en él empleadas y los motivos por los cuáles han sido seleccionadas frente al resto de opciones existentes. Por este motivo, la primera de las metodologías mencionadas para reducir modelos, no será explicada, mientras que la segunda de ellas (la seleccionada en este caso), sí.

El método QR es uno de los mejores métodos numéricos, a nivel computacional, para solucionar el problema de la reducción de modelos dinámicos a menor número de parámetros a identificar. El método se llama así, porque emplea un tipo de descomposición de las ecuaciones en forma matricial, llamado "descomposición QR".

W es la matriz de observación del sistema, tal que $\tau = W * \phi$, donde W es de dimensiones n x p (instantes y parámetros). Este método trabaja con 3 matrices nuevas (Q, R, E), tal que:

$$W_{nxp} * E_{pxp} = Q_{nxn} * R_{nxp}$$

Q es una matriz cuadra ortonormal, R es una matriz triangular superior y E es la llamada matriz de permutación. Esta descomposición se obtiene directamente de un comando de Matlab.

El rango de W será igual al rango de R, pero si el rango de W es menor que el número de sus columnas, entonces los últimos p-r elementos de la diagonal de R serán nulos, y la matriz WE podrá escribirse como:

$$W * E = [W1, W2] = [Q1, Q2] \begin{bmatrix} R1 & R2 \\ 0_{(n-r)\,x\,r} & 0_{(n-r)\,x\,(p-r)} \end{bmatrix} = [Q1 * R1, Q1 * R2]$$

R1 tiene rango máximo y es regular (r x r). De esta forma se dividen las matrices W*E y Q en función del rango de R:

$$W1 = Q1 * R1 \quad y \quad W2 = Q1 * R2$$

Operando se tiene que:

$$W2 = W1 * \beta; donde\ \beta = R1^{-1} * R2$$

Reordenando la ecuación original, se llega a una nueva expresión:

$$\tau = W * \phi \rightarrow W_b * \phi_b = \tau$$

Donde $\phi_b = \phi_1 + \beta * \phi_2$.

El algoritmo que debe programarse, es el que se muestra en los apuntes de la asignatura de Identificación de Sistemas Dinámicos, elaborados por Xabier Iriarte, [10, p 70] en los que nos hemos apoyado para la explicación de estas técnicas:

- Calcular $\boldsymbol{Q}$, $\boldsymbol{R}$ y $\boldsymbol{E}$ de la descomposición QR de $\boldsymbol{W}$.
- Calcular el rango de $\boldsymbol{W}$: r
- Determinar el número de columnas de $\boldsymbol{W}$: p.
- Extraer las matrices $\boldsymbol{R}_1$ y $\boldsymbol{R}_2$ de $\boldsymbol{R}$.
- Calcular la matriz $\boldsymbol{\beta}$.
- Reordenar las filas de $\boldsymbol{W}$ con $\boldsymbol{WE}$.
- Extraer las primeras r columnas de $\boldsymbol{WE}$.

- Crear un vector simbólico de parámetros $\boldsymbol{\phi} = \{p_1; p_2; \ldots p_n\}$.
- Reordenar las filas de $\boldsymbol{\phi}$ con $\boldsymbol{E'\phi}$.
- Extraer $\boldsymbol{\phi}_1$ y $\boldsymbol{\phi}_2$ a partir de $\boldsymbol{E'\phi}$.
- Determinar las expresiones simbólicas de los parámetros base $\boldsymbol{\phi}_b$.

Fig 31: Pasos y etapas del algoritmo para reducir el modelo dinámico a parámetros base

Para realizar la reducción del modelo, se ha creado el código de Matlab "QR_reduction.m", que podrá consultarse en el capítulo *Anexos*, y que básicamente devuelve el vector "v", que contiene las columnas independientes extraídas de la descomposición QR.

A continuación se muestran las expresiones de los parámetros base:

Ixy7
Ixz7
Ixy6
Iyz7
Ixz6
Iyz6
Ixx6 + Iyy7 - 1.0*Izz6 - 0.001764*m7 - 0.084*mz6
Izz7
Ixy5
Iyz5 + 0.19*my5
Ixz5
Ixz4
my7
Iyy6 - 1.0*Ixx6 + Izz6
mx7
Ixy4
0.042*m7 + mz6 + mz7
Iyz4
mx6
Ixx6 + Ixx7 - 1.0*Izz6 - 0.001764*m7 - 0.084*mz6
Izz5 + Izz6
Ixy3
Ixx5 + Iyy4 + Izz6 - 0.0441*m5 - 0.008*m6 - 0.008*m7 - 0.42*mz4
Ixx3 + Ixx4 + Iyy2 - 1.0*Iyy4 + 0.0441*m3 + 0.1764*m4 + 0.1764*m5 + 0.1764*m6 + 0.1764*m7 + 0.42*mz3
mx5
Ixz3
my5 + my6
Iyy4 + Iyy5 - 0.0441*m5 - 0.008*m6 - 0.008*m7 - 0.42*mz4
Iyz3 + 0.21*my3
mx4
0.21*m5 + 0.4*m6 + 0.4*m7 + mz4 + mz5
Ixx3 + Iyy2 + Izz4 + 0.0441*m3 + 0.1764*m4 + 0.1764*m5 + 0.1764*m6 + 0.1764*m7 + 0.42*mz3
Izz3 - 1.0*Iyy2 - 1.0*Ixx3 - 0.0441*m3 - 0.1764*m4 - 0.1764*m5 - 0.1764*m6 - 0.1764*m7 - 0.42*mz3
Ixz2
Ixy2
Iyz2
Iyy2 + Iyy3 + 0.0441*m3 + 0.1764*m4 + 0.1764*m5 + 0.1764*m6 + 0.1764*m7 + 0.42*mz3
Izz1 + Izz2
c3
Ixx2 - 1.0*Iyy2 + Izz1
c5
c4
c7
c6
c2
c1
mx3
0.21*m3 + 0.42*m4 + 0.42*m5 + 0.42*m6 + 0.42*m7 + mz2 + mz3
my3 + my4
mx2
cc2
cc1
cc4
cc6
cc3
cc5
cc7

Fig 32: Expresiones de los parámetros base para este modelo dinámico (elaboración propia)

6.2 PARAMETRIZACIÓN DE TRAYECTORIAS

El siguiente paso en la fase del diseño de experimentos, será la parametrización de trayectorias para poder, precisamente, construir una matriz W en "n" instantes, que permita encontrar los parámetros base del modelo dinámico.

Esta fase consiste en parametrizar una trayectoria, creándola con alguno de los métodos que existen para crear y definir trayectorias de forma simbólica y automática, para emplearla posteriormente en el código previamente explicado de reducción del modelo.

La idea que subyace en este punto, es conseguir una matriz W asociada a "n" instantes en los que se discretiza la trayectoria continua diseñada, que permitirá aplicar la descomposición QR.

Existen multitud de formas de parametrizar trayectorias para realizar este paso. Uno de los mas habituales, es construir una ecuación haciendo uso de series de Fourier, que en el tiempo se traduzca en una onda o trayectoria que se compone de varias funciones armónicas simples.

Mediante la parametrización de trayectorias, somos capaces de obtener unos valores de coordenadas en función del tiempo y de unos parámetros que denominaremos parámetros de trayectoria.

Las ecuaciones en coordenadas de posición, velocidad y aceleración que pueden tomarse para la definición de las trayectorias de forma paramétrica, son las siguientes (extraídas de los apuntes de la asignatura de Identificación de Sistemas Dinámicos del máster IMAC, elaborados por Xabier Iriarte [10].

$$q_i(t) = q_{i0} + \sum_{j=1}^{N_H} \left[\frac{a_{ij}}{2\pi f j} \sin(2\pi f j t) - \frac{b_{ij}}{2\pi f j} \cos(2\pi f j t) \right]$$

$$\dot{q}_i(t) = \sum_{j=1}^{N_H} \left[a_{ij} \cos(2\pi f j t) + b_{ij} \sin(2\pi f j t) \right]$$

$$\ddot{q}_i(t) = \sum_{j=1}^{N_H} \left[-a_{ij} 2\pi f j \sin(2\pi f j t) + b_{ij} 2\pi f j \cos(2\pi f j t) \right]$$

Fig 33: Ecuaciones de parametrización de trayectorias mediante series de Fourier [10]

Básicamente habrá que fijar el número de armónicos que tendrán las funciones que describen las trayectorias (N_H), y el número de grados de libertad del mecanismo (i).

El número de parámetros de los que dependerá la trayectoria, se calcula por medio de la siguiente expresión:

$$Nº parámetros\ trayectoria = (2 * N_H + 1) * N_{DOF}$$

Por último habrá que definir aspectos como período fundamental, incremento temporal… Y otros aspectos que juegan un papel fundamental para definir aspectos simbólicos previos a la optimización.

6.3 OPTIMIZACIÓN DE TRAYECTORIAS

Llegados a este punto se ha tratado el tema de parametrización de trayectorias con otros fines, para plantear la forma en la que se construyen las trayectorias del robot en los experimentos de forma automatizada y simbólica, haciendo uso de técnicas computacionales.

Bajo el planteamiento anteriormente comentado, en el que se va ensamblando una matriz "W", y se mide un vector de fuerzas o pares, la precisión con la que se pueden estimar los parámetros del robot (que forman parte de su modelo dinámico), depende notablemente de cómo se lleve a cabo ese experimento. Una mala definición de las trayectorias de los experimentos, por ejemplo, podría llevar a una mala estimación de los parámetros.

La forma en la que se diseñan los experimentos es crucial, y las decisiones tomadas en este punto son vitales. Una de las formas en las que se puede planificar y perfeccionar el diseño de experimentos, es mediante la optimización de trayectorias.

La optimización de trayectorias consiste en elegir la forma más apropiada y eficiente en la que el robot deberá moverse en los experimentos, para que todos los parámetros a estimar sean los suficientemente excitados como para realizar una identificación dinámica precisa.

Esta optimización de trayectorias puede hacerse en base a diferentes variables y criterios, que se explicarán en los siguientes capítulos.

Gracias a la parametrización ya mencionada, la trayectoria se define en función de un determinado número de parámetros de trayectoria, que permiten obtener la trayectoria óptima siguiendo el siguiente planteamiento.

$$\boldsymbol{\vartheta}^* = \arg\min_{\boldsymbol{\vartheta}}\{\mathcal{F}(\boldsymbol{W}(\boldsymbol{\vartheta}))\}; \quad \text{sujeto a:} \quad \boldsymbol{h}_R(\boldsymbol{\vartheta}) \leq \mathbf{0}$$

Fig 34: Ecuación de obtención de parámetros de trayectoria óptima [10]

Se dice que una trayectoria es óptima según el criterio, si se cumple la ecuación anterior (h_R es un conjunto de inecuaciones restrictivas que imponen condiciones a esas trayectorias).

6.3.1 Número de armónicos

Por número de armónicos se entiende el número de combinación de sinusoides en base Fourier se emplean para la parametrización de trayectorias.

Si se recuerdan las ecuaciones expuestas en *Fig 33: Ecuaciones de parametrización de trayectorias mediante series de Fourier*, el número de armónicos define el número de sumas de senos menos cosenos (con sus correspondientes amplitudes y frecuencias) que construyen la expresión simbólica paramétrica de la trayectoria a optimizar para los experimentos.

La decisión que compete a la selección del número de armónicos óptimos para la parametrización de trayectorias de robots, se toma en base a los criterios adoptados por los científicos desarrolladores de las metodologías de identificación dinámica en los artículos referenciados en *Bibliografía y referencias* y citados previamente.

En los artículos que se han leído para plantear el trabajo y para resolver dudas, se toman de forma habitual cinco armónicos para la construcción de las trayectorias parametrizadas del robot.

Por este hecho, y dada la dilatada experiencia que tienen los autores de dichos artículos en el campo de la identificación dinámica en robots, se ha decidido tomar el mismo número de armónicos para la parametrización y optimización de trayectorias que permitirán diseñar los experimentos.

6.3.2 Período fundamental

El período fundamental se define como el inverso de la frecuencia fundamental que modula la frecuencia de los senos y cosenos de los armónicos que conforman la expresión paramétrica de la trayectoria en base a series de Fourier.

Es necesario procurar que este período fundamental no sea ni demasiado pequeño ni demasiado grande, un valor similar al de los artículos referenciados, puede ser suficiente.

Para este proyecto se ha adoptado un valor de frecuencia fundamental de 0.1, que es lo mismo que decir que se ha definido un período fundamental de 10.

Este período fundamental, define la duración del experimento de forma directa.

6.3.3 Incremento temporal

Para que la identificación se haga correctamente, hay que seleccionar un incremento temporal que sea suficiente para discretizar la trayectoria en un número de puntos que permita captar y representar correctamente puntos clave de la onda en el período fundamental elegido.

En este trabajo, el incremento temporal se obtiene automáticamente en función del período fundamental y número de puntos seleccionados. Puede verse en el siguiente extracto de código.

```
% main_optimize_trajectory
clear all

% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% number of points
n=100;

% fundamental frequency
f=0.1;

% crit='cond';
crit='dopt';

% time instants
times=0:(1/f)/(n-1):(1/f);
```

Fig 35: Extracto en el que se definen número de puntos, frecuencia fundamental, criterios e instantes

6.3.4 Límites del robot

Por construcción, el robot tiene una serie de limitaciones en posición (movimiento de cada una de las siete articulaciones que tiene) y en velocidad.

Esos límites del robot han de ser considerados en la definición de las trayectorias, para garantizar que el robot podrá describir una trayectoria suficientemente excitable sin resultar dañado.

La optimización de trayectorias para robots está condicionada al cumplimiento de una serie de restricciones que se definen con unas inecuaciones en concreto, y que ponen límites al movimiento del robot (en posición, velocidad…).

Las restricciones de los actuadores en los enlaces, se definen por medio de las siguientes ecuaciones (extraídas de los apuntes de la asignatura de Identificación de Sistemas Dinámicos [10, p 75]

$$q_{i_{\text{mín}}} \leq q_i(\vartheta, t) \leq q_{i_{\text{máx}}}$$
$$\dot{q}_{i_{\text{mín}}} \leq \dot{q}_i(\vartheta, t) \leq \dot{q}_{i_{\text{máx}}}$$
$$\ddot{q}_{i_{\text{mín}}} \leq \ddot{q}_i(\vartheta, t) \leq \ddot{q}_{i_{\text{máx}}}$$

Fig 36: Planteamiento de las inecuaciones de restricción en posición, velocidad y aceleración, respectivamente [10]

Si se recuerdan los pasos anteriores, las posiciones, velocidades y aceleraciones que adopta cada grado de libertad del mecanismo, pueden expresarse en función de los parámetros de trayectoria que se han creado previamente, y por ende, estos límites pueden expresarse en función de estos parámetros:

$$lb \leq R\vartheta \leq ub$$

Fig 37: Reescritura de las inecuaciones de restricciones en función de los parámetros de trayectoria [10]

Los valores de “lb” y “up” (lower boundary y upper boundary), agrupan los límites inferiores y superiores de una variable del robot en cada enlace. “R” se corresponde con una matriz, y el vector “ϑ” contiene los parámetros de trayectoria.

El coste computacional asociado a la optimización de trayectorias, depende en gran parte del número de parámetros de trayectorias que se tienen y de el número de puntos en los que éstas son discretizadas.

Para este robot, los límites son los siguientes:

	Range of motion (º)	**Maximum velocity (rpm)**
AXIS 1	±170	±2267
AXIS 2	±120	±2267
AXIS 3	±170	±2267
AXIS 4	±120	±2000
AXIS 5	±170	±2167
AXIS 6	±120	±3600
AXIS 7	±175	±3600

Fig 38: Límites del robot en posición y velocidad (elaboración propia)

En el siguiente extracto de código se muestra la programación de los límites del robot para la optimización de trayectorias sujeta a restricciones:

```
% security factor
sf= 0.95;

qmax=   sf*[170;120;170;120;170;120;175]*pi/180;
qmin= - qmax;
dqmax=  sf*[2267;2267;2267;2000;2167;3600;3600]*2*pi/60/160;
```

Fig 39: Extracto del código de Matlab en el que se programan los límites del robot

Como puede observarse en la imagen anterior, se ha aplicado un factor de seguridad a la hora de definir los límites del robot (estamos diciendo que en realidad puede moverse a posiciones y velocidades inferiores de las reales). Este factor de seguridad es imprescindible, porque los actuadores no son instantáneos; por lo que seguramente, cuando se acerque al límite de robot, es posible (de no existir este factor) que se sobrepasara, dañando el robot.

Además se ha considerado la relación de multiplicación de la reductora, y se ha realizado la conversión a unidades del Sistema Internacional.

Como apunte, es importante recalcar que las funciones habitualmente empleadas de optimización, buscan mínimos locales y globales en dicha optimización. Las dos funciones más típicas para la obtención de esos mínimos (el mínimo error entre iteraciones) son “fminunc” y “fmincon”. “fminunc” encuentra mínimos en la función que no está sometida a restricciones (unconstrained), mientras que “fmincon” encuentra mínimos en la función, sujeta a restricciones (constrained). En este caso particular, se usará la segunda función de las mencionadas al comienzo (para tener en cuenta los límites del robot, y generar la variable “options”, y posteriormente la primera, cuando simplemente se busquen minimos con las opciones de minimización generadas con las restricciones.

A continuación se muestra un extracto del bucle programado para este efecto.

```
cont=0;
while true
    cont=cont+1;
    rng('shuffle');
    param_traj_0=2*pi*rand((2*n_h+1)*n_dof,1);

%
options=optimoptions(@fmincon,'Display','iter','MaxFunctionEvaluations',1000*((2*n_h+1)*n_dof));
    options=optimoptions(@fmincon,'Display','iter','MaxFunctionEvaluations',inf,'maxiter',10000);

    fun=@(param_traj)objective_function(param_traj,n,f,crit);
%
[param_traj_opt,value]=fminunc(@(param_traj)objective_function(param_traj,n,f,crit),param_traj_0,options);
%     [param_traj_opt,value]=fminunc(fun,param_traj_0,options);
    [param_traj_opt,value]=fmincon(fun,param_traj_0,A,b,[],[],[],[],[],options);
%     x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)

        value_str=num2str(value);
        cont_str=num2str(cont);

        if crit=='cond'
                s=['param_cond_',cont_str,'_',value_str,'.mat'];
        elseif crit=='dopt'
                s=['param_dopt_',cont_str,'_',value_str,'.mat'];
        else
                s=['param_crit_',cont_str,'_',value_str,'.mat'];
        end

        command_str=['save ',s,' param_traj_opt;'];
        eval(command_str);

end
param_traj_opt
```

Fig 40: Extracto del código en el que se programa el bucle que optimiza las trayectorias sujetas a restricciones

6.3.5 Criterios de optimización

Las optimizaciones de trayectorias pueden hacerse de diferentes formas. Uno de los factores que más determinan el tipo de optimización respecto a los demás, es cómo se va a cuantificar la validez, idoneidad o importancia de un resultado, frente a otros (de cara a tomarlo como bueno y válido, o malo y desechable).

La forma en la que se cuantifica esto, se llama criterio de optimización de trayectorias, y generalmente existen dos muy comúnmente utilizados [10].

- **Criterio en base a la transmisión de errores (número de condición):** Este criterio considera que no hay error en las mediciones de posiciones, velocidades y aceleraciones, y que los errores en las medidas, son los asociados a la medición de las fuerzas. Este hecho lleva a realizar una estimación de parámetros (tras ensamblar las matrices del problema), de forma que dichos parámetros sean los que minimicen las discrepancias entre mediciones y predicciones (resuelven el problema de la optimización). De este criterio se deduce que para un error en la medición de fuerzas concreto, el experimento con el cual se estimarán los parámetros, deberá tener asociada una matriz de observación con un número de condición lo más pequeño posible (límite de desviación bajo).

- **Criterio en base a la información (D-optimality):** Se basa en un planteamiento de la estimación en términos estadísticos, asumiendo que en la realización de un experimento suele ser posible conocer el ruido de las mediciones. Si se calcula la matriz de varianzas y covarianzas de la estimación, podrá plantearse una búsqueda de trayectorias que proporcionen las menores varianzas posibles en la estimación. Es un modelo robusto (sobre todo si las magnitudes del modelo son de diferente índole, orden de magnitud…). En resumen, es un criterio que permite obtener trayectorias óptimas en base al criterio de minimización la varianza en la estimación de parámetros de una determinada matriz de observación.

En este trabajo, se emplean ambos métodos para obtener trayectorias óptimas. En el código programado, basta con comentar o descomentar la línea en la que se adopta uno u otro criterio (ver imagen).

```
% crit='cond';
crit='dopt';
```

Fig 41: Extracto del código de optimización de trayectorias en el que se selecciona el criterio de optimización

6.4 OBTENCIÓN DE TRAYECTORIAS DE ENLACE (ARRANQUE Y PARADA)

De forma breve, el último de los puntos que queda por concretar antes de realizar los experimentos, es la definición de trayectorias de enlace entre las óptimas obtenidas, y las posiciones de reposo del robot.

El robot tiene una posición de reposo, con sus ángulos puestos a "cero", que básicamente hacen que el robot adopte una posición totalmente extendida hacia arriba. Esa es la posición en la que el robot permanece inactivo, por lo que habrá que diseñar 2 trayectorias adicionales, para cada trayectoria optimizada anteriormente: una que enlace del reposo al inicio de la trayectoria optimizada, y otra que enlace del final de la trayectoria optimizada al reposo.

Para hacer esto, se ha generado un código que permite cargar una trayectoria cualquiera anteriormente optimizada, definir una serie de parámetros (tiempo que dura la trayectoria optimizada, tiempo que dura la trayectoria de enlace, número de armónicos, número de grados de libertad, número de bucles de enlace…), y generar la trayectoria.

La teoría en la que se basa la obtención de estas trayectorias, es la teoría de las curvas de Bézier. Estas curvas se basan en la definición de puntos clave, que permiten definir continuidad en posición, posición y velocidad, o combinadas con estas, también en aceleraciones y sobreaceleraciones.

En primer lugar, se genera un código simbólico en el que se definen las expresiones de las curvas de Bézier y los puntos clave que definen las condiciones de continuidad. Posteriormente, se ejecuta el código numérico, en el que se particulariza esta teoría para cada trayectoria.

Un ejemplo de una trayectoria de enlace, de reposo a condiciones iniciales de trayectoria optimizada, puede verse en la siguiente imagen (ver página siguiente en horizontal con más detalle).

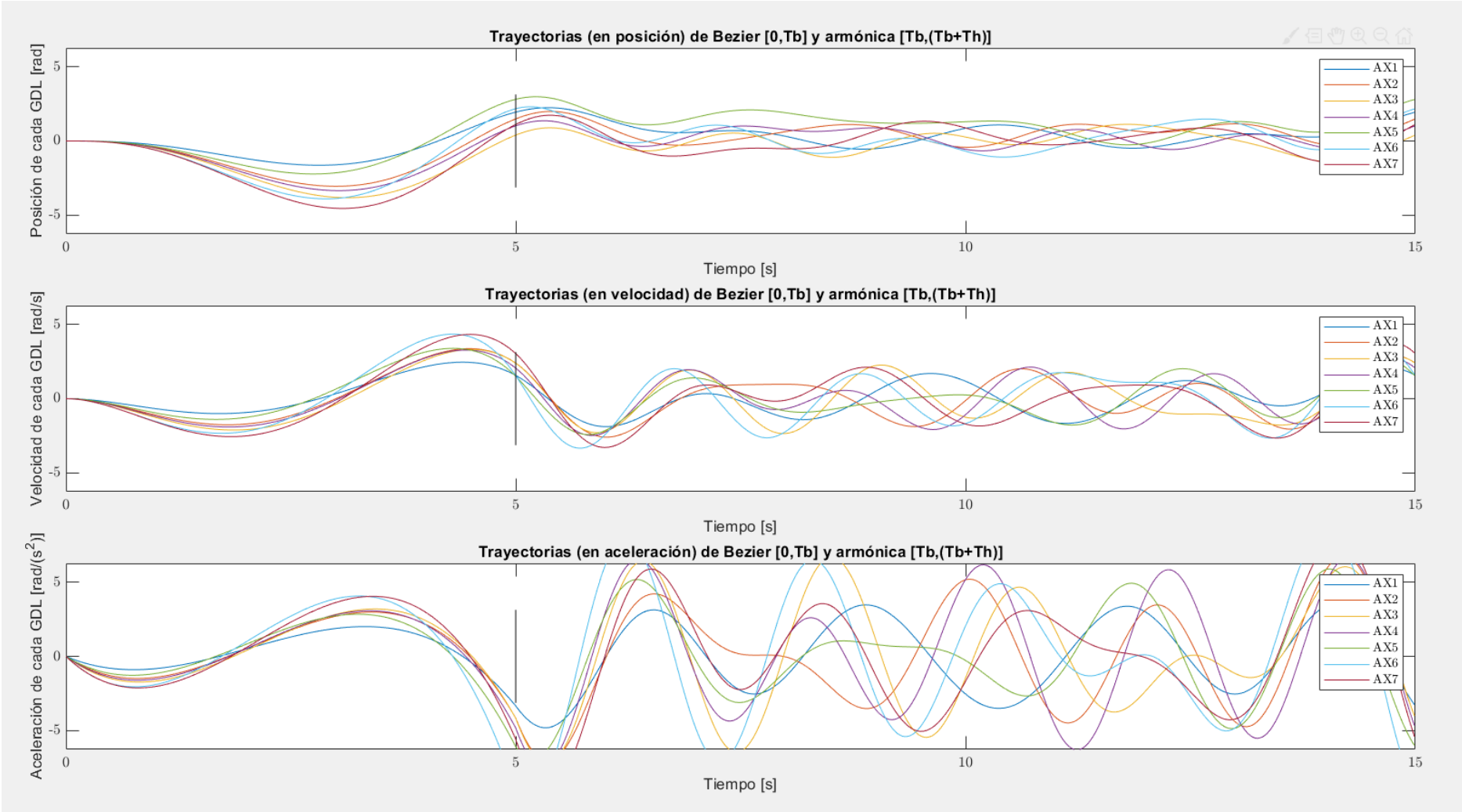


Fig 42: Ejemplo de un enlace entre posición de reposo, y condiciones iniciales de trayectoria optimizada.

A continuación se muestra un ejemplo de una de las trayectorias óptimas diseñadas:

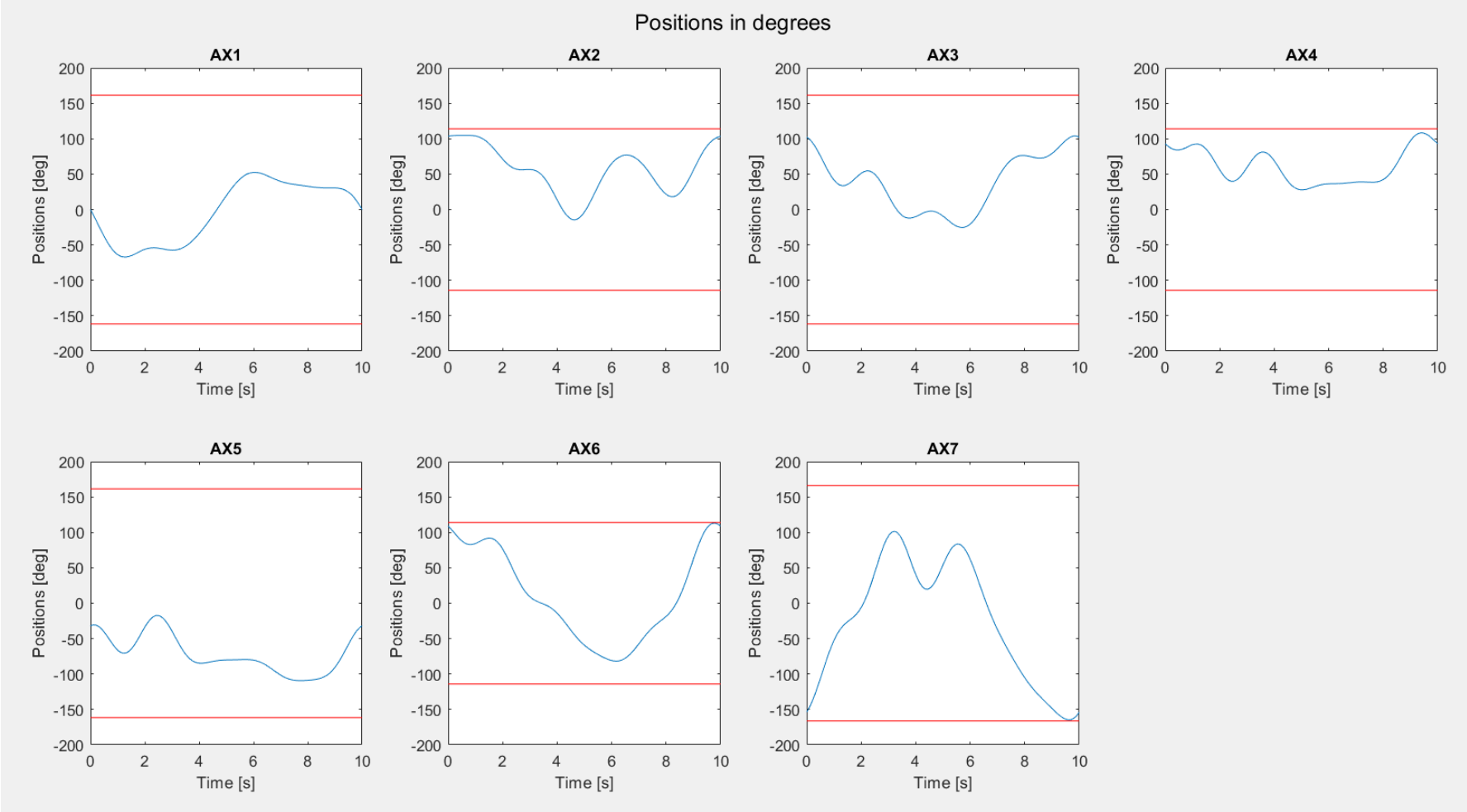
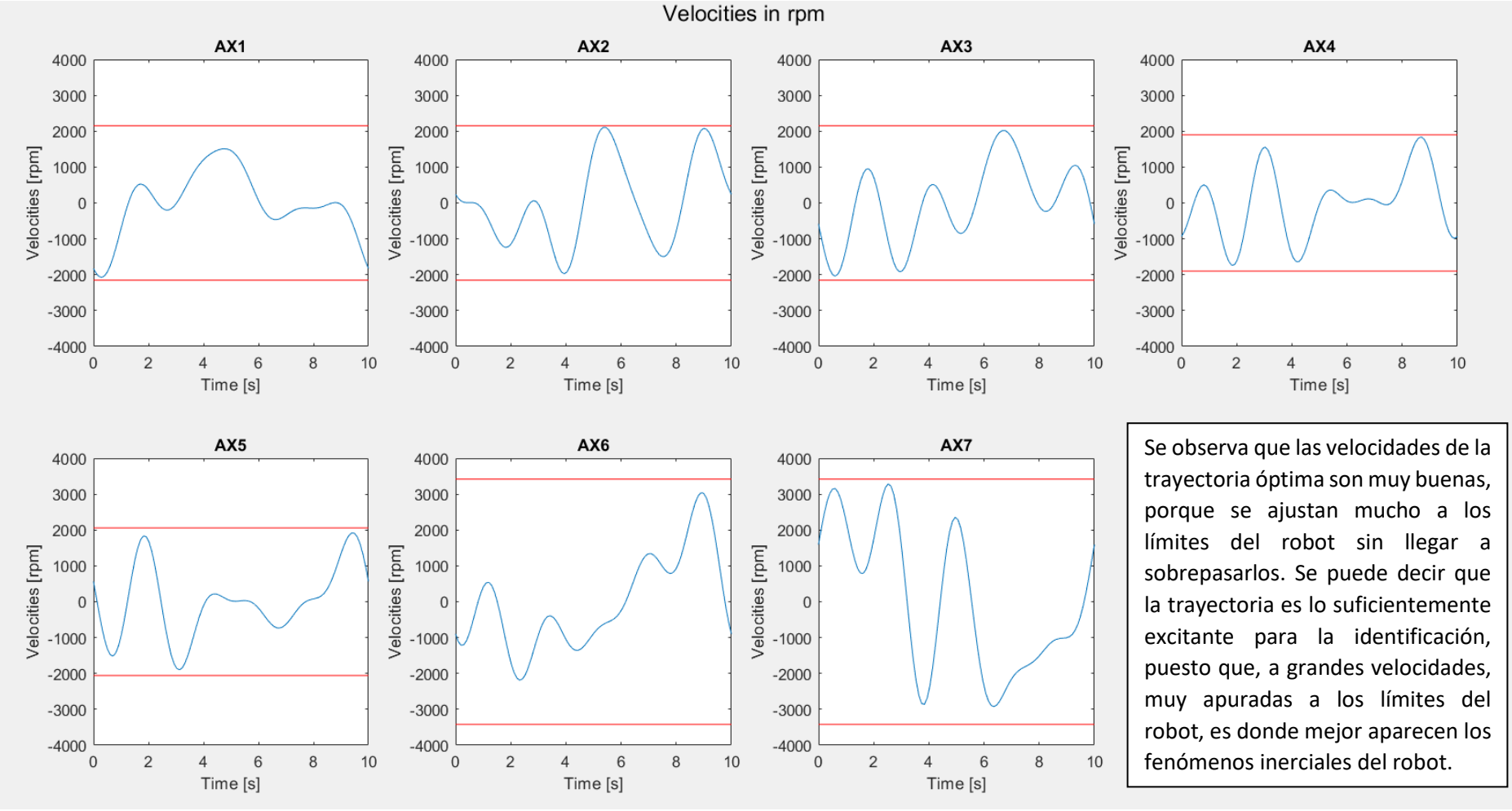


Fig 43: Posiciones de los ejes en la trayectoria óptima programada



Se observa que las velocidades de la trayectoria óptima son muy buenas, porque se ajustan mucho a los límites del robot sin llegar a sobrepasarlos. Se puede decir que la trayectoria es lo suficientemente excitante para la identificación, puesto que, a grandes velocidades, muy apuradas a los límites del robot, es donde mejor aparecen los fenómenos inerciales del robot.

Fig 44: Velocidades de los ejes en la trayectoria óptima programada

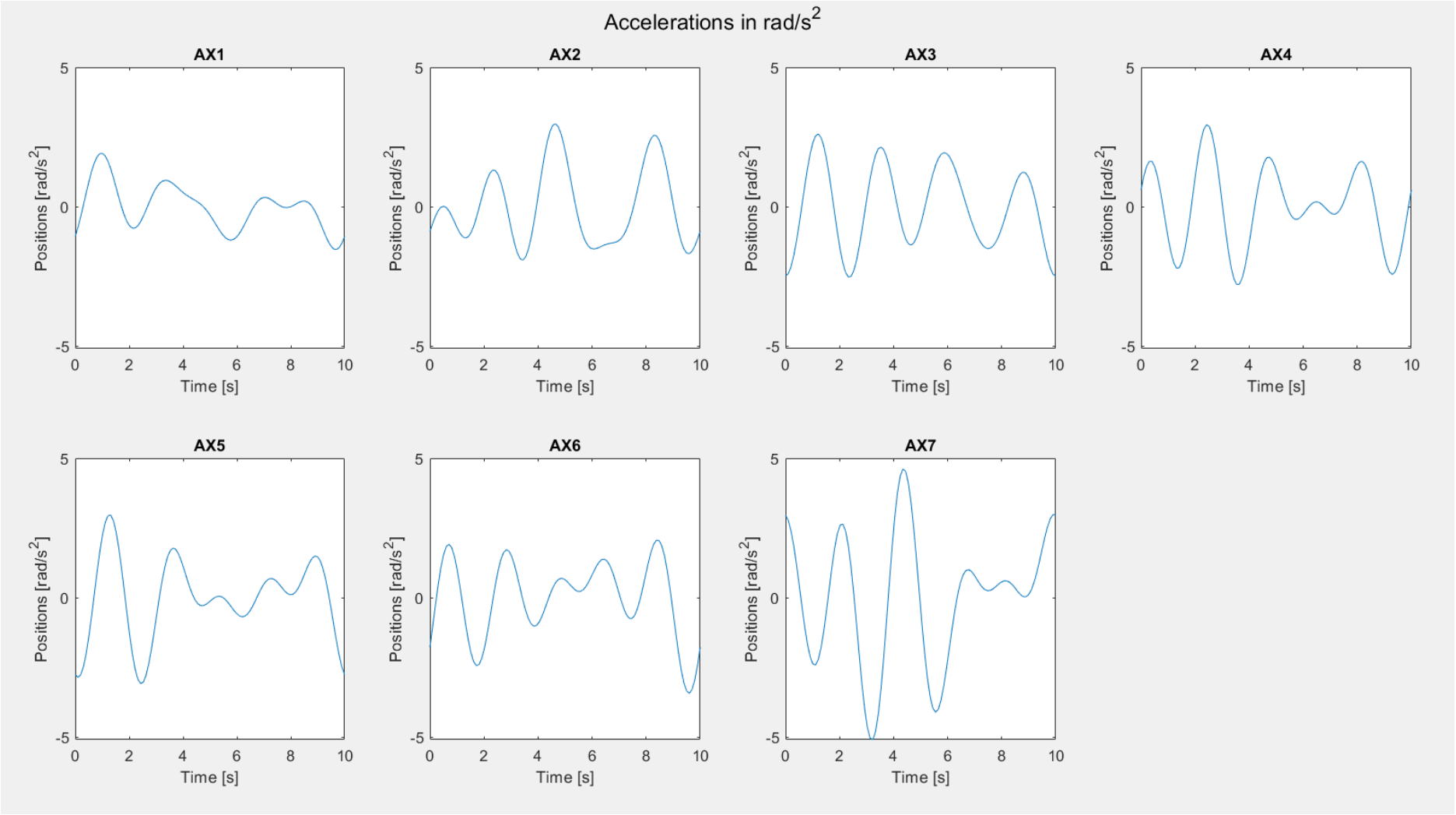


Fig 45: Aceleraciones de los ejes en la trayectoria óptima programada

7 IDENTIFICACIÓN DINÁMICA

Una vez se ha completado el diseño de los experimentos, se procede a la realización de los mismos. Tal y cómo se ha comentado anteriormente, el planteamiento de este trabajo final de máster, era llevar a cabo los experimentos con un robot real, para llevar a cabo un proceso de identificación dinámica lo más realista posible.

Finalmente no han podido llevarse a cabo los experimentos con el robot real, puesto que a pesar de tener preparados los ficheros con las trayectorias varios meses antes, no ha sido posible lograr realizarlos con la empresa propietaria del robot debido a la alta carga de trabajo existente.

Para poder seguir avanzando en este trabajo, se acordó con el tutor del mismo, que la mejor solución sería realizar los experimentos de manera ficticia, generando nosotros mismos unos datos con funciones randomizadoras de Matlab, siguiendo criterios estadísticos estudiados.

Este hecho nos permitirá tener unos datos que nos permitan realizar la identificación dinámica (básicamente interesa la estimación de parámetros y pares).

Posteriormente, para la fase de validación, seleccionaremos otra trayectoria optimizada en la fase de diseño de experimentos, y la moldearemos de igual forma que antes, para poder verificar que la estimación de parámetros es similar, y que sigue siendo aceptable para otra trayectoria diferente. Los pasos a dar son los que se muestran a continuación:

- Evaluar la matriz W para una de las trayectorias optimizadas (experimento).
- Obtener los valores de los parámetros reales (suponiendo unos valores que tengan sentido, logrando que se asemejen a los del robot real). Esta parte es realmente compleja, pero para ello nos apoyaremos en el modelo 3D simplificado previamente construido para obtener las ecuaciones dinámicas.
- Calcular los valores de los pares reales en las articulaciones.
- Distorsionar los valores de los pares reales con funciones randomizadoras de Matlab, para simular unos valores de pares medidos (con algo de ruido) en los experimentos reales (de haberlos podido realizar).
- Obtener la matriz de observación “base”, del modelo reducido.
- Realizar la estimación de los parámetros base, con los pares medidos y la matriz de observación del modelo reducido.
- Comparar los valores de los parámetros estimados y reales, y determinar si son correctos o no.
- Realizar la validación de los experimentos y obtener gráficos que permitan sacar conclusiones.

En los siguientes apartados se detallan los resultados y la validación, y se muestran extractos del código programado para lograr llevar a cabo la identificación dinámica.

7.1 RESULTADOS

Para obtener los resultados de los experimentos, se ha seleccionado una de las trayectorias optimizadas en base al criterio del número de condición.

Cargando esa trayectoria en Matlab, se obtienen los parámetros de trayectoria que hacen que dicha trayectoria parametrizada, sea óptima desde el punto de vista de la identificación dinámica.

Para evaluar y montar la matriz de observación necesaria, se ejecuta la función "get_W_with_param_traj.m", que se encarga de evaluar la trayectoria paramétrica en función de los parámetros de trayectoria óptimos, y devuelve la matriz ensamblada.

```
%% LIMPIAR ESPACIO DE TRABAJO

clear all
close all
clc

%% TRAYECTORIA
% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% number of points
n=100;

% fundamental frequency
f=0.1;

% crit='cond';
crit='dopt';

load param_cond_3_51.8841.mat

%% ENSAMBLAR MATRIZ DE OBSERVACIÓN
W=get_W_with_param_traj(param_traj_opt,n,f,crit);

save W W
```

Fig 46: Extracto del código "experiments.m" donde se ensambla la matriz de observación

A la matriz obtenida, se le aplica el método de reducción QR (función "QR_reduction_exp.m", y se cargan los valores de los parámetros reales. Estos valores se importan desde los datos obtenidos en el código de obtención del modelo. En dicho código original, se crearon unas funciones que leían el fichero de OPENSCAD dibujado y obtenían los valores de los parámetros inerciales asociados a cada sólido (masa, primeros momentos de inercia y tensor de inercia). Con esos datos, y dando unos valores aleatorios (pero realistas) a los coeficientes de fricción (0.5 en los parámetros asociados al modelo de fricción viscosa, y 0.8 a

los parámetros de fricción asociados al modelo de fricción de Coulomb), se obtienen los valores de los parámetros "reales", que simulan los del robot real en el supuesto de haber podido llevar a cabo los experimentos.

```
%% OBTENER MODELO REDUCIDO
[phi,phi_b,Wb,E,r,beta]=QR_reduction_exp(W);
phi_b;
phi_b_definitivo=vpa(phi_b,4)
```

Fig 47: Extracto de las líneas de comando del script, donde se realiza la reducción del modelo

De los valores obtenidos de los parámetros inerciales, se eliminan los duplicados (por ejemplo si Iyz = Izy), y se calculan los valores de los pares reales, haciendo uso de la fórmula básica estudiada en Identificación de Sistemas Dinámicos.

$$Y = W * \phi \rightarrow \tau = W * \phi$$

En este caso el vector Y, es el vector de los pares (τ). Esto nos lleva a obtener la variable que recoge los valores reales de estos pares. Estudiando los valores obtenidos, se extrae el valor cuyo valor absoluto es máximo, y se calcula la desviación típica en la medición de los pares como un 5% del valor de la amplitud máxima de dichos pares. Este paso es necesario para obtener un valor de desviación típica en la medición realista (podría haberse considerado un 10% de la amplitud), para así obtener resultados válidos y similares a los del caso real del experimento.

```
%% CALCULAR LOS PARES ASOCIADOS A ESA TRAYECTORIA CON LOS PARÁMETROS TEÓRICAMENTE REALES

tau_real=W*phi_real;

%% OBTENER LA DESVIACIÓN TÍPICA DE LOS PARES MEDIDOS (5-10% DE AMP MAX

for i=1:7
    max_torque=max(tau_real)
    min_torque=min(tau_real)
end

if abs(max_torque)>abs(min_torque)
    sigma=0.05*abs(max_torque);
else
    sigma=0.05*abs(min_torque);
end
```

Fig 48: Extracto del código programado "experiments.m" para calcular valor de desvición típica realista

Con ese valor de desviación típica, se calcula el vector de pares medidos (distorsionando de forma aleatoria los reales) y se comparan los reales y los medidos, para comprobar que éstos tiene sentido.

```
%% OBTENER VALORES DE PARES MEDIDOS (RANDOMIZANDO LOS REALES)

tau_m=tau_real+sigma*randn(size(tau_real));

%% COMPARAR PARES REALES CON LOS MEDIDOS (COMO SI HUBIÉRAMOS HECHO EXPERIMENTOS)

figure
for i=1:7
    cont=0;
    for j=i:7:length(tau_real)
        cont=cont+1;
        tau_ax_real(cont)=tau_real(j);
        tau_ax_m(cont)=tau_m(j);
    end
    ax_real(:,i)=tau_ax_real';
    ax_m(:,i)=tau_ax_m';
end

for i=1:7
    ax(i)=subplot(2,4,i);
    plot(ax_real(:,i))
    hold on
    plot(ax_m(:,i))
    xlabel('TIME [s]')
    ylabel('TORQUE [Nm]')
    legend('\tau_{real}','\tau_{medida}')
    title(['REAL VS. ESTIMATED \tau AX ',num2str(i)])
end
```

Fig 49: Obtención de los pares medidos en la trayectoria óptima del experimento, y representación de la comparativa

En la página siguiente se muestran las gráficas comparativas de pares reales y medidos.

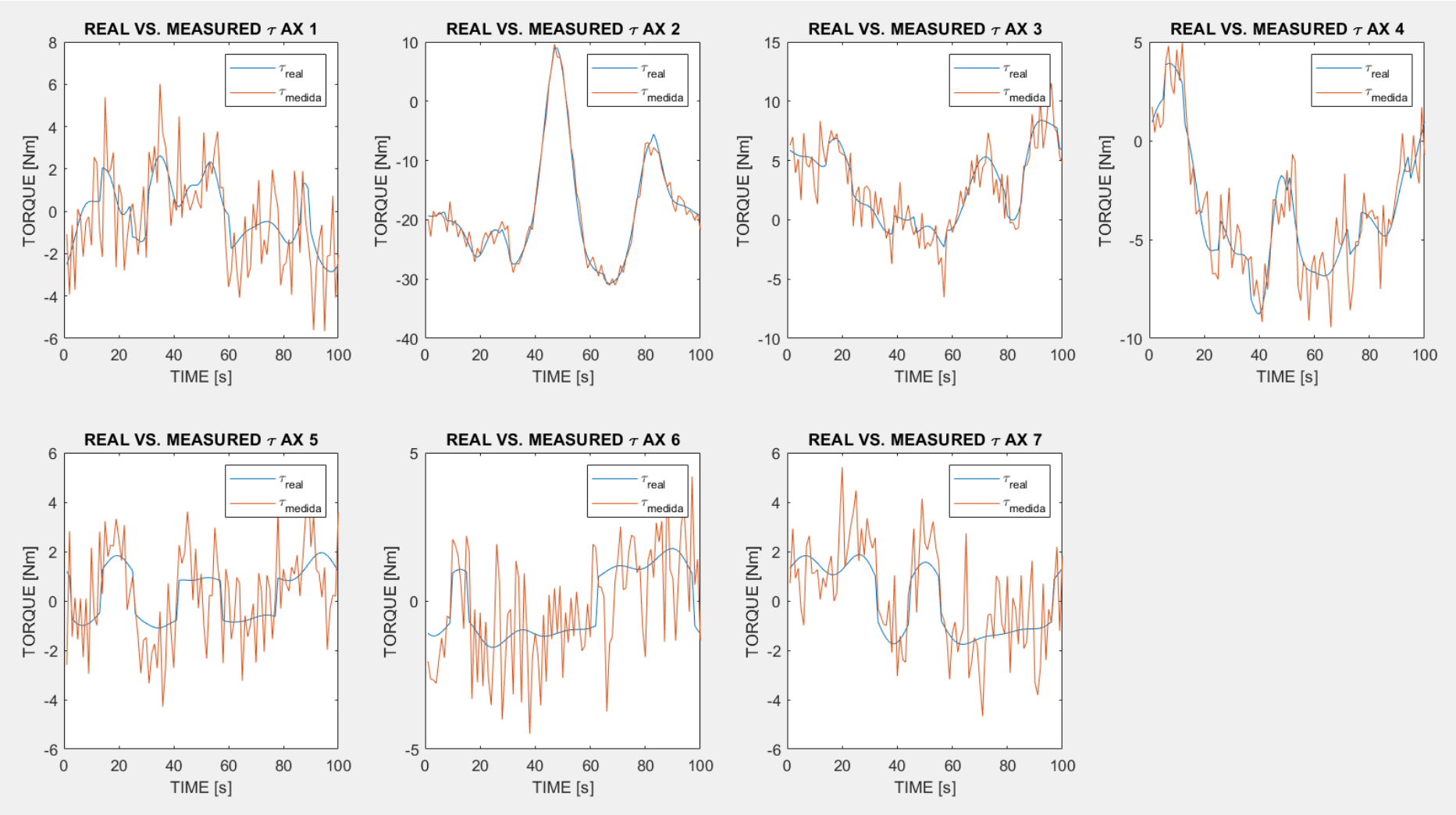


Fig 50: Comparativa entre pares reales para lograr la trayectoria óptima, y pares medidos en el experimento

En la figura anterior se observa la comparativa entre pares reales (necesarios para llevar a cabo la trayectoria diseñada) y los pares medidos. Como puede observarse, los valores medidos se ajustan bien a los reales, pero con una oscilación en torno al valor teórico, que simboliza el ruido de la medida.

En general el ajuste es bueno, en los 7 ejes del robot. Sin embargo, puede verse cómo el efecto del ruido, al haberse definido la desviación típica de las medidas como un 5% de la amplitud máxima de los pares, tiene mayor influencia en aquellos ejes en los que el par es menor (1, 5, 6, 7), y una menor influencia en aquellos ejes en los que el par es mayor (2, 3 y 4).

Tras calcular los pares medidos, se puede realizar la estimación de los parámetros haciendo uso de la alguna de las siguientes expresiones (se ha probado con ambas y los resultados son prácticamente idénticos):

$$\hat{\Phi} = W_{base} \backslash \tau_m$$

$$\hat{\Phi}_{LS} = (W' * W)^{-1} * W' * \tau_m = W^{+} * \tau_m \rightarrow \hat{\Phi}_{LS} = pinv(W) * \tau_m$$

```
%% REALIZAR ESTIMACIÓN DE LOS PARÁMETROS

phi_base_est=Wbase\tau_m;
% phi_base_est=inv(Wbase'*Wbase)*Wbase'*tau_m;

E_prima_phi_real=E'*phi_real;
phi_1_real=E_prima_phi_real(1:r,:);
phi_2_real=E_prima_phi_real(r+1:end,:);

phi_base_real = phi_1_real + beta * phi_2_real;

save phi_base_est phi_base_est
```

Fig 51: Expresiones y extracto del código donde se realiza la estimación de los parámetros

A continuación, se calcula la matriz de varianzas y covarianzas de la estimación de parámetros, cuya diagonal principal contiene las desviaciones típicas de cada parámetro estimado. Para ello será necesario recurrir a la desviación típica de las mediciones (calculado antes como el 5% de la amplitud el par medido de mayor amplitud).

Una vez calculadas dichas desviaciones típicas, se podrá verificar que los valores de los parámetros estimados son aceptables, siempre y cuando estén dentro de los siguientes rangos:

$$\left[\Phi_{base} - 3 * \sigma_{\hat{\Phi}_{base}}, \Phi_{base} + 3 * \sigma_{\hat{\Phi}_{base}}\right]$$

Representando estos resultados en un gráfico “errorbar” de Matlab, podemos ver el valor real de ese parámetro, y si cae dentro de los límites aceptables. Con los valores de parámetros estimados, se calculan los valores de pares estimados.

```
%% CALCULAR MATRIZ DE VARIANZAS Y COVARIANZAS EN ESTIMACIÓN DE PARÁMETROS

VAR_phi_base_est = sigma^2 *inv(Wbase'*Wbase);
sigma_phi_base_est= sqrt(diag(VAR_phi_base_est));

%% REPRESENTAR PARÁMETROS ESTIMADOS DENTRO DE RANGOS ACEPTABLES (VER VALIDEZ)

x=1:length(phi_base_est);
for i=1:length(phi_base_est)
    err_neg(i)=phi_base_real(i) - 3*sigma_phi_base_est(i);
    err_pos(i)=phi_base_real(i) + 3*sigma_phi_base_est(i);
end

figure
errorbar(x,phi_base_est,err_neg,err_pos,'o','markersize',5)
xlabel('Base parameters')
ylabel('Value')
title('VALIDATION OF ESTIMATED PARAMETERS (ACCEPTABLE RANGES)')

%% ESTIMAR LOS PARES

tau_est=Wbase*phi_base_est;
```

Fig 52: Extracto del código programado para validar parámetros estimados y calcular pares estimados

En la página siguiente puede verse en apaisado, el gráfico donde se representan los valores de parámetros estimados comparándolos con el rango aceptable para cada uno de ellos. Se observa en dicha gráfica que todos los parámetros han sido estimados satisfactoriamente, puesto que caen dentro del rango aceptable anteriormente mencionado.

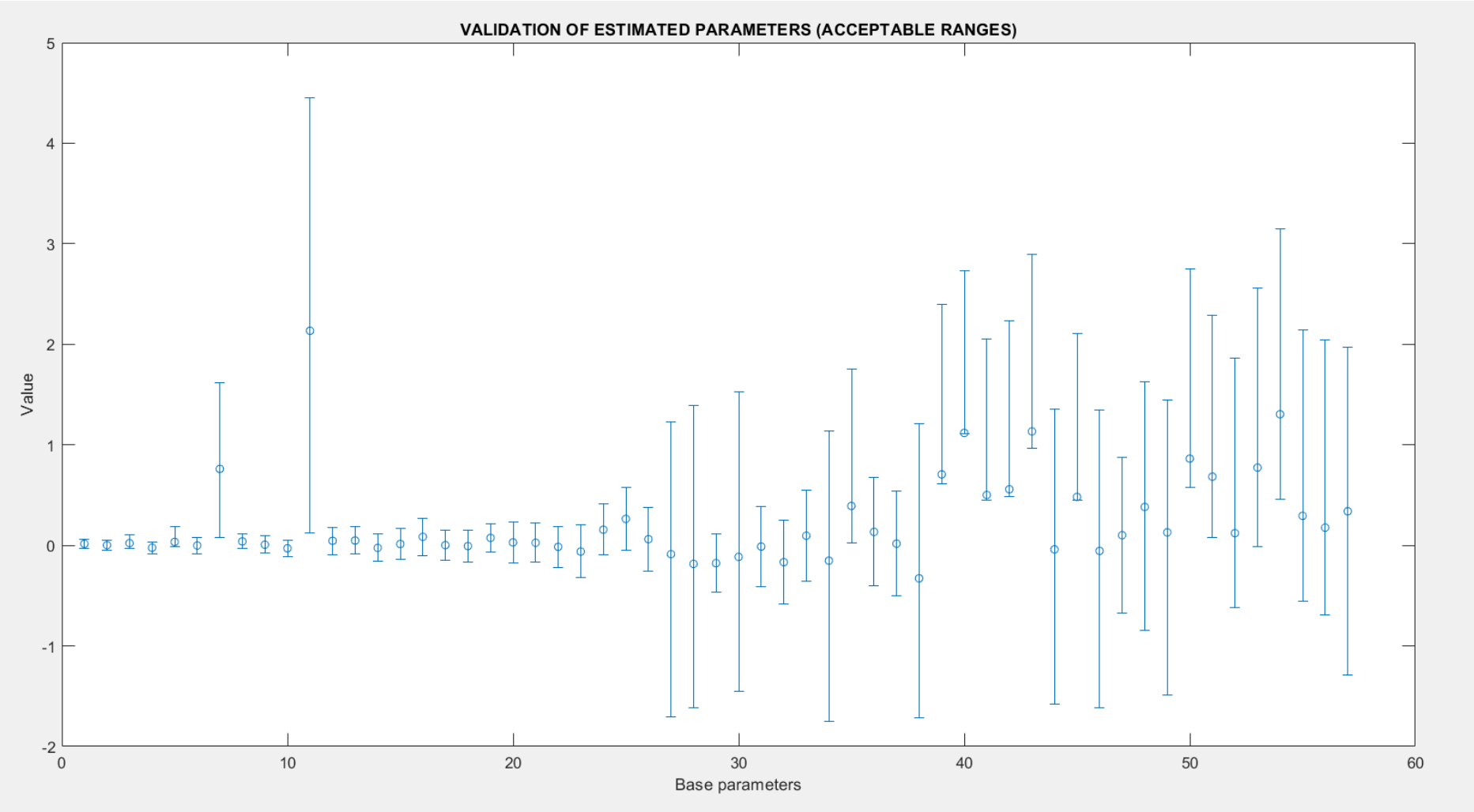


Fig 53: Validación de parámetros estimados, comprobando que se encuentran dentro del rango aceptable

Para seguir adelante con los resultados, se calculan los residuos (error de estimación) de los pares en los enlaces para esta trayectoria, haciendo uso de la siguiente expresión extraída de la teoría estudiada en el máster:

$$\hat{\varepsilon} = Y_m - \hat{Y} = Y_m - W(x) * \hat{\Phi}$$

Los valores de esta variable residuo, nos dan una idea de las diferencias entre los valores medidos y estimados. Si se dibuja un histograma de los residuos, puede verse con claridad si estos siguen una distribución normal, y puede estimarse igualmente la media y desviación típica que se ven en el diagrama.

Para este caso se ha hecho exactamente esto, puesto que una de las partes más interesantes del trabajo, es conseguir un modelo que permita estimar con mucha precisión (o bastante precisión) los pares que se obtienen en los enlaces, puesto que estos dan una idea sobre cuán forzados están los ejes del robot, en determinadas posiciones.

Por ejemplo, si se está estudiando una determinada aplicación, en la que la velocidad es crucial y los pares la limitan, quizá nos permita hacernos una idea de una relación entre pesos a manipular, pares desarrollados al manipular dicho peso en la trayectoria, y velocidades que se alcanzan, para, sin forzar el robot, comprobar que lo planteado es viable o no.

Para lograr estudiar los residuos, se han programado las siguientes líneas de código:

```
%% CÁLCULOS ASOCIADOS A LOS RESIDUOS

% RESIDUO
epsilon=tau_real-tau_est;

% MEDIA
media_epsilon=mean(epsilon);
sigma_epsilon_est=std(epsilon);

% VARIANZA
m=size(tau_real,1);
var_epsilon_est=sigma_epsilon_est^2;

% HISTOGRAMA EPSILON
figure
histfit(epsilon)
title('Histograma de los residuos epsilon')
```

Fig 54: Extracto de código programado para estudiar los residuos en la estimación de los pares

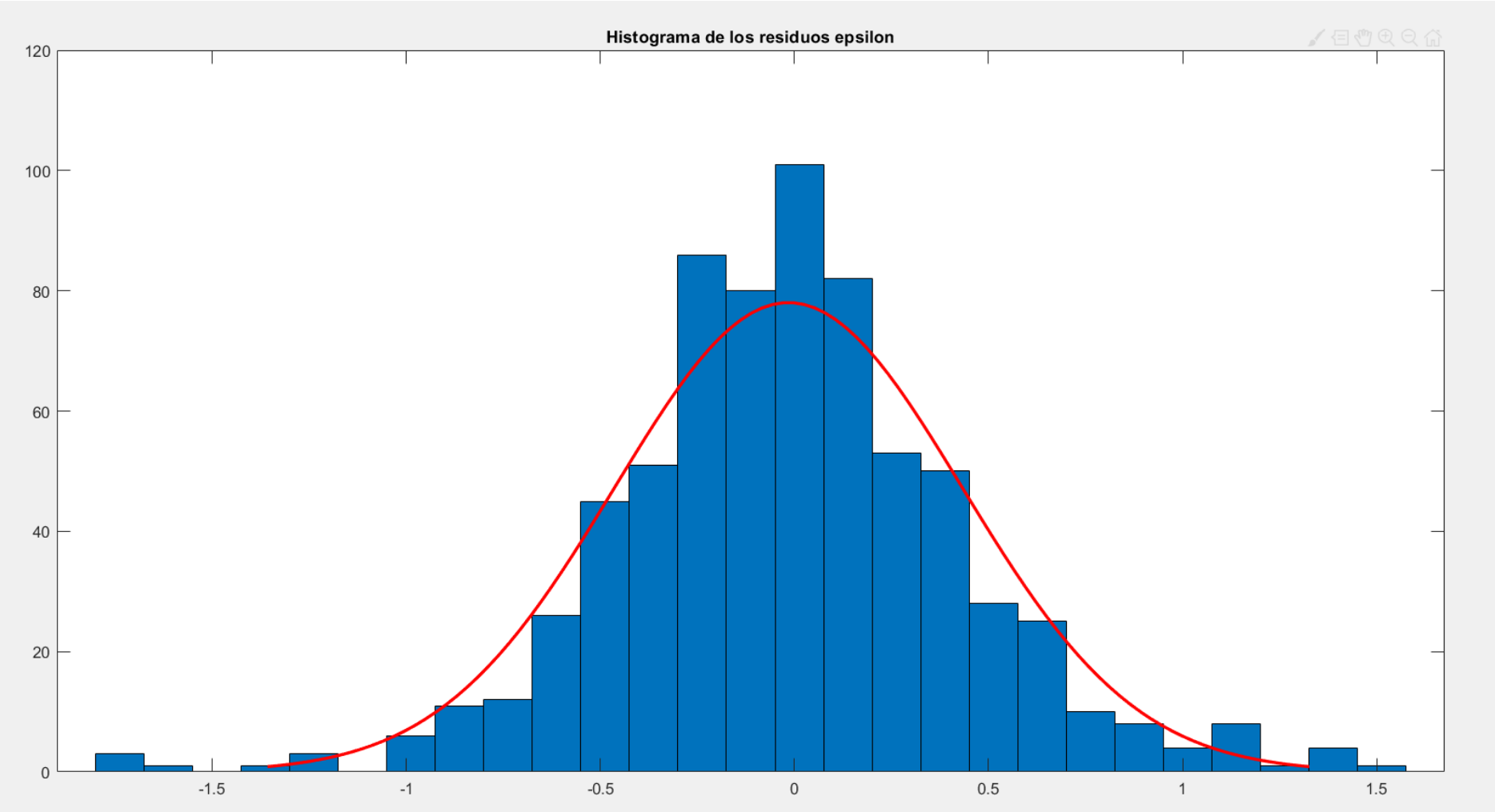


Fig 55: Histograma de los residuos en la estimacion de los pares

Para terminar con los experimentos, se calculan los intervalos de confianza para la estimación de los pares (interesa obtener estos valores para determinadas trayectorias al hacer simulaciones en Matlab, en lugar de en el robot (no forzarlo)), al 95% (con una confianza del 95%, el intervalo calculado contendrá el verdadero valor el par estimado).

En el siguiente extracto de código se ven las líneas programadas para calcular estos intervalos:

```
%% GRAFICAR PARES MEDIDOS, ESTIMADOS Y SUS INTERVALOS DE CONFIANZA

figure
for i=1:7
    cont=0;
    for j=i:7:length(tau_real)
        cont=cont+1;
        tau_ax_est(cont)=tau_est(j);
        IC_ax_min(cont)=IC_min_tau(j);
        IC_ax_max(cont)=IC_max_tau(j);
    end
    ax_est(:,i)=tau_ax_est';
    ax_IC_min(:,i)=IC_ax_min';
    ax_IC_max(:,i)=IC_ax_max';

end

for i=1:7
    ax(i)=subplot(2,4,i);
    plot(ax_m(:,i),'b.')
    hold on
    plot(ax_est(:,i),'m')
    plot(ax_IC_min(:,i),'g')
    plot(ax_IC_max(:,i),'k')
    xlabel('TIME [s]')
    ylabel('TORQUE [Nm]')
    legend('\tau_{medida}','\tau_{estimada}','IC_{min}','IC_{max}')
    title(['MEDIDO VS. ESTIMATED \tau AX ',num2str(i)])
end
```

Fig 56: Código programado para calcular y graficar los intervalos de confianza

En la siguiente figura puede comprobarse como los intervalos de confianza son ajustados a los valores estimados, pero garantiazando que con una confianza del 95%, dentro de esos intervalos, se encontrarán los valores del par.

Las gráficas parecen más disepersas cuanto menor es el valor del par.

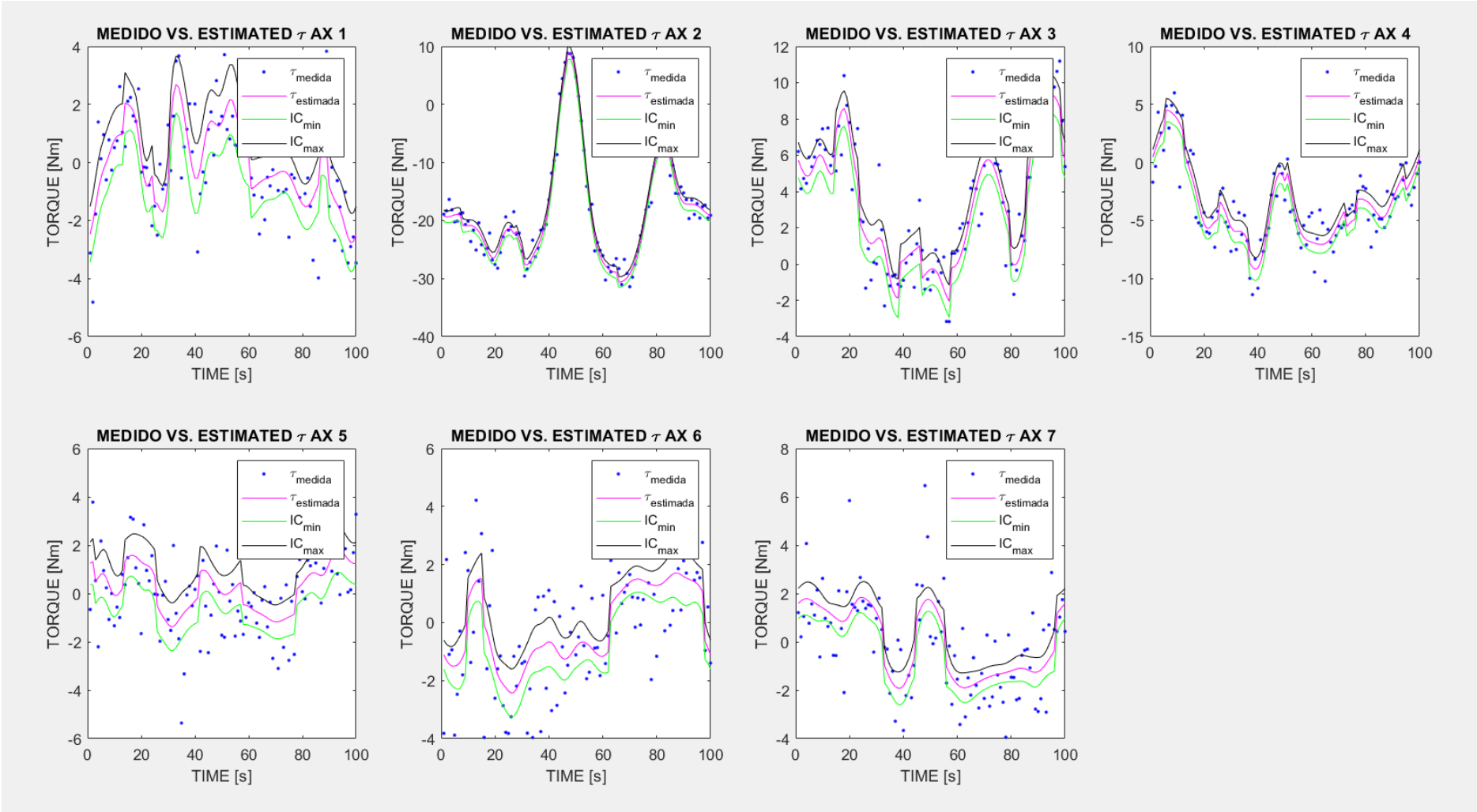


Fig 57: Intervalos de confianza al 95% junto con los pares medidos y estimados

7.2 VALIDACIÓN

El capítulo de validación está estrechamente relacionado con el de los resultados de los experimentos. En él se pretende analizar otra trayectoria, para obtener los pares asociados a la misma y su nueva desviación típica en la medida (dependiendo de su amplitud).

Con estos nuevos datos, se puede realizar la nueva estimación de los parámetros base, que se comparará con la estimación inicial, para ver si con ambos modelos se obtienen resultados parecidos.

El resto de pasos son iguales, se realiza una comparativa entre pares medidos y reales para la nueva trayectoria, se analiza si los nuevos parámetros estimados están dentro de los intervalos aceptables (en función de la desviación típica asociada a la estimación de cada uno de ellos), y se analizan los residuos y se calculan los intervalos de confianza para la estimación de los pares.

A continuación se presentan las gráficas que de él se obtienen.

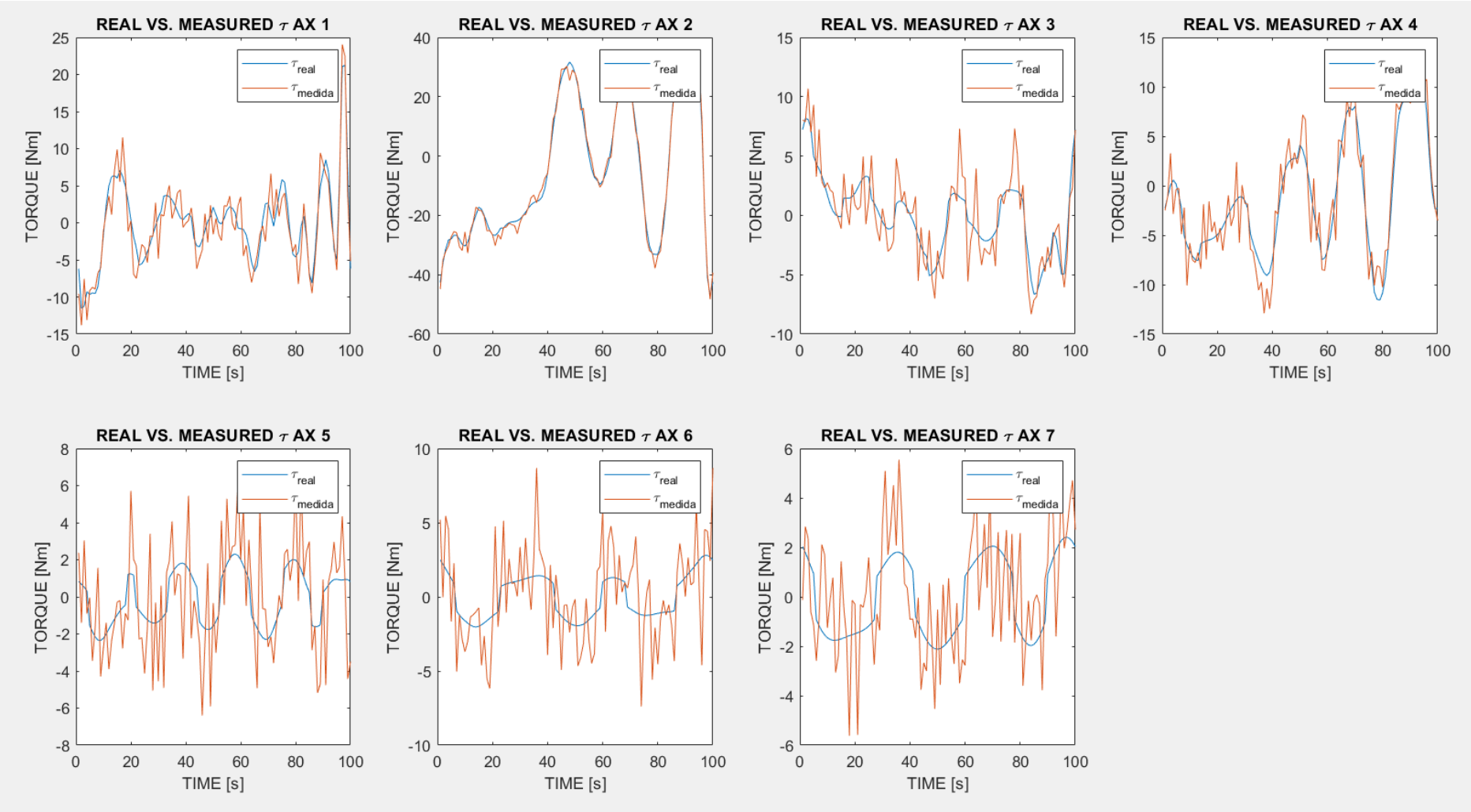


Fig 58: Comparativa entre pares reales y medidos para la segunda trayectoria (de validación)

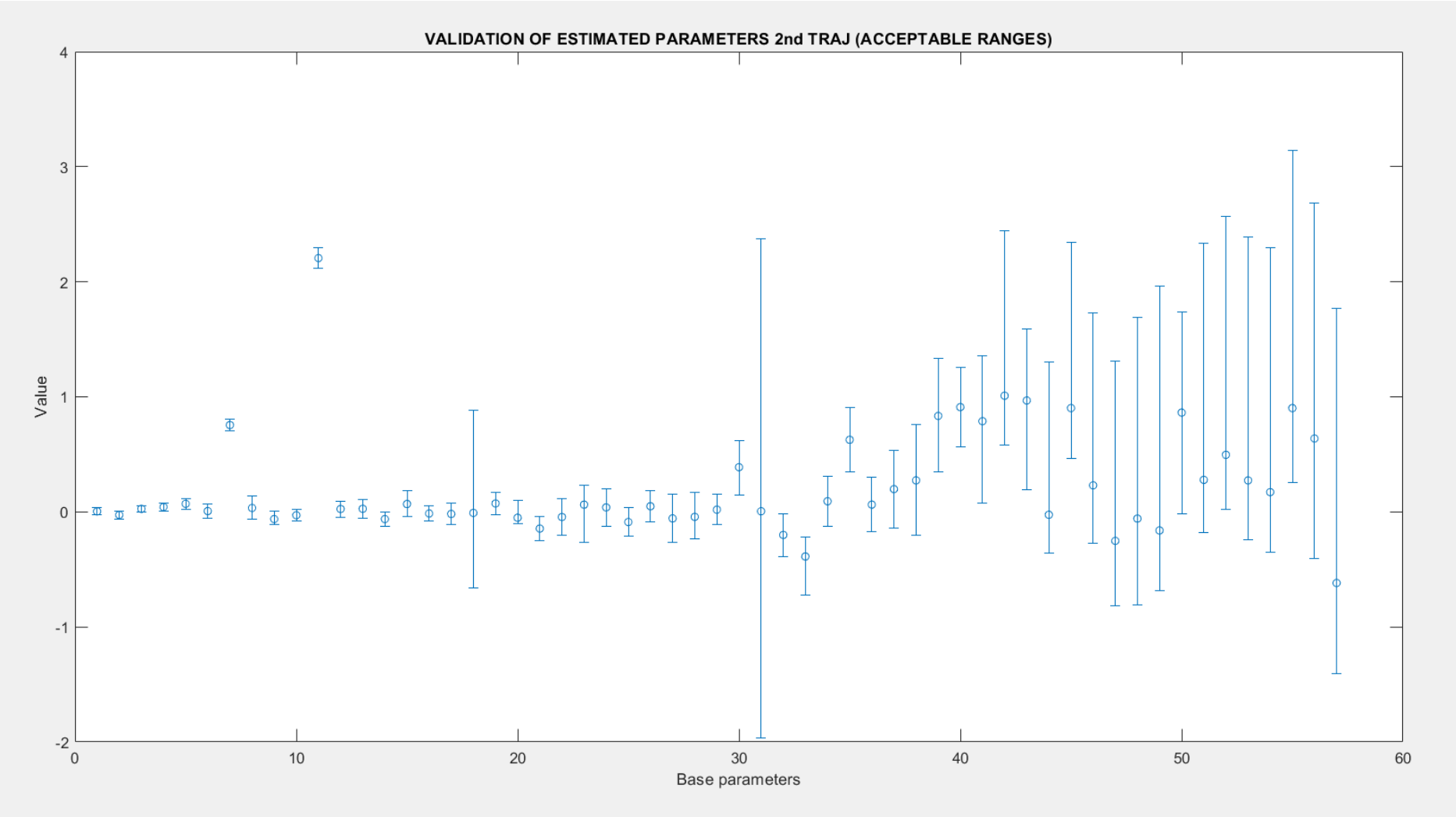


Fig 59: Verificación de la validez de los parámetros base estimados con esta segunda trayectoria (aceptabilidad dentro del rango). Están desordenados respecto a la trayectoria 1

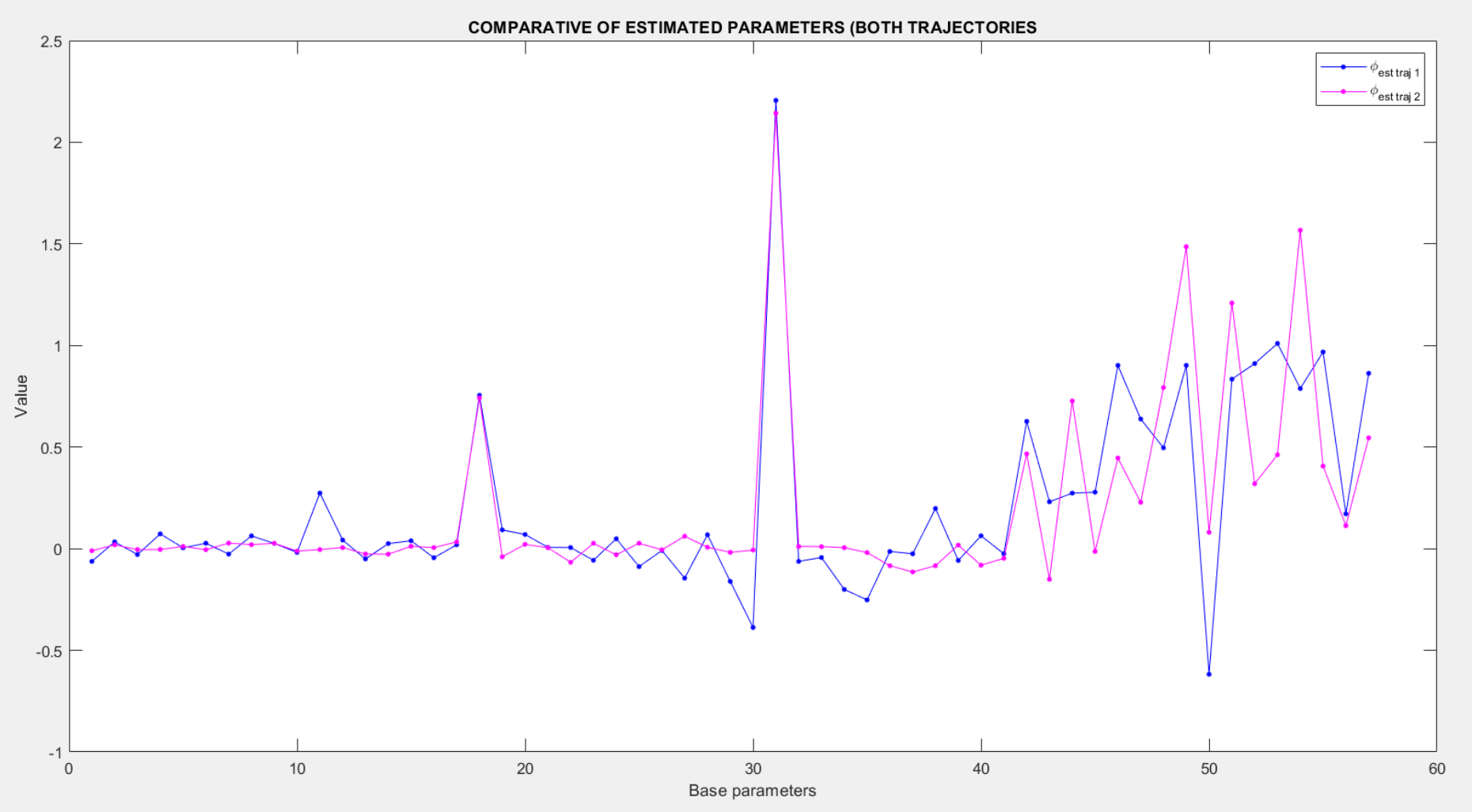


Fig 60: Comparativa entre los valores estimado con la primera y con la segunda trayectoria

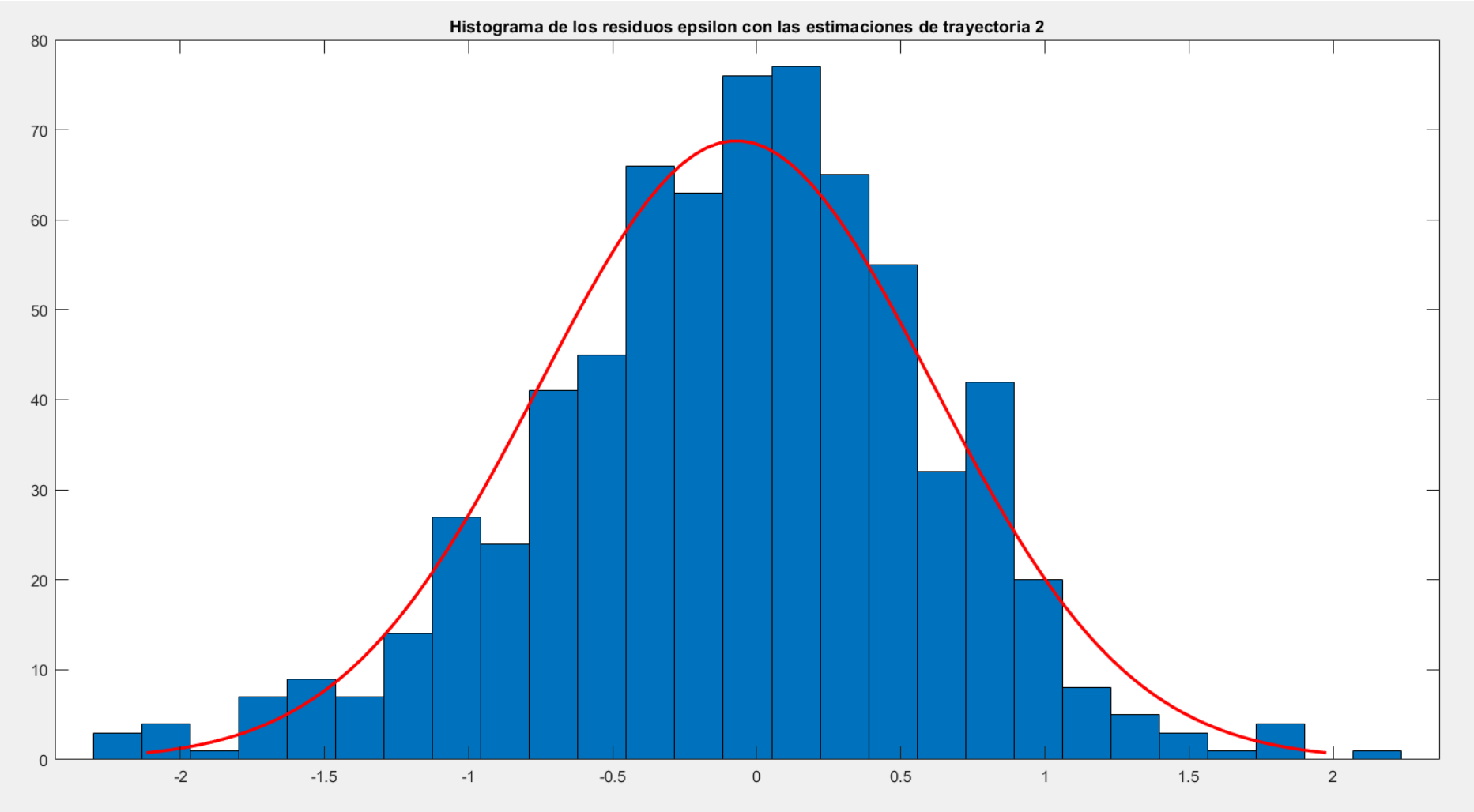


Fig 61: Histograma de los residuos en la segunda trayectoria

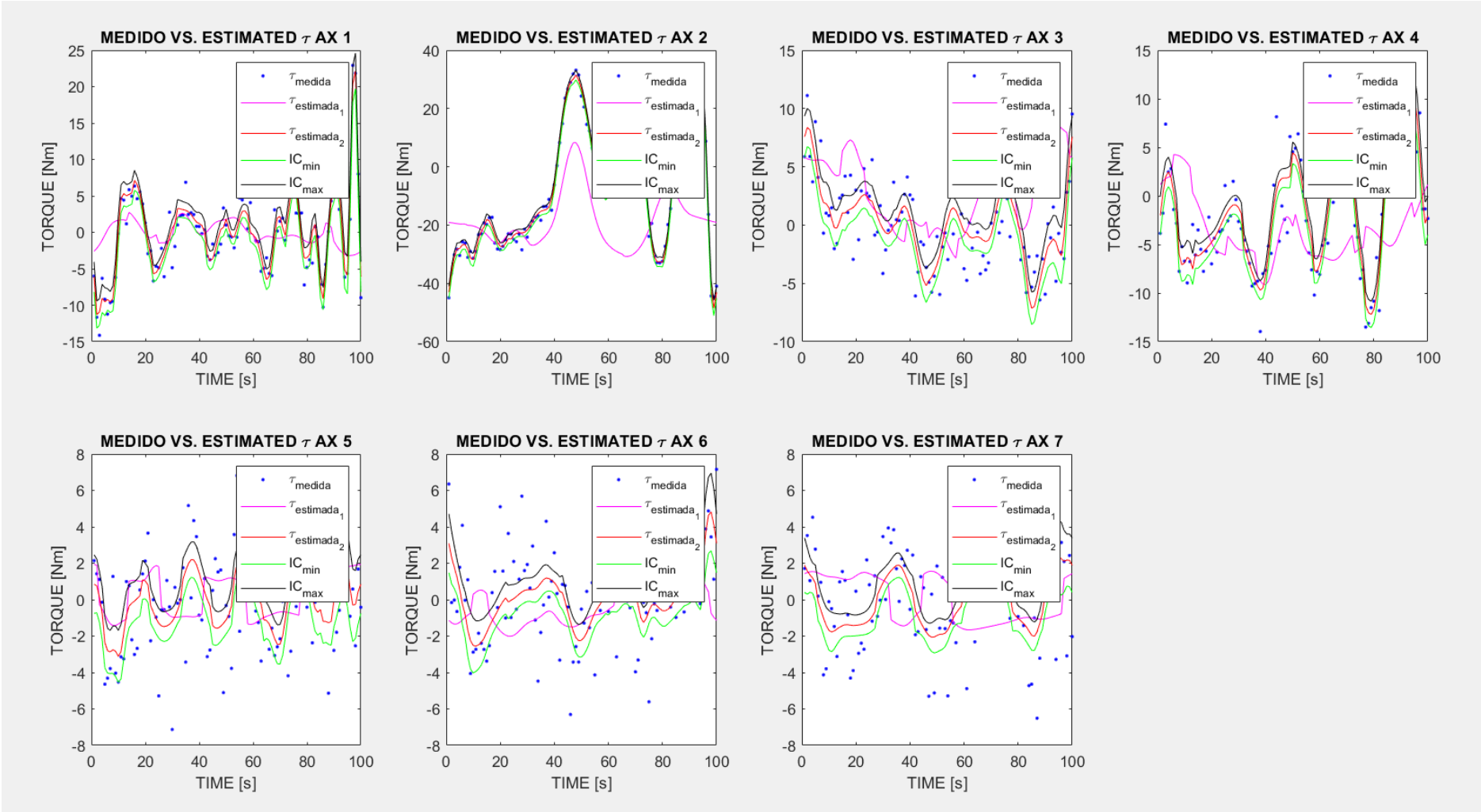


Fig 62: Pares medidios y pares estimados con sus intervalos de confianza al 95% para la estimación de la trayectoria 2. Se muestra la estimación de la trayectoria 1 para ver las diferencias

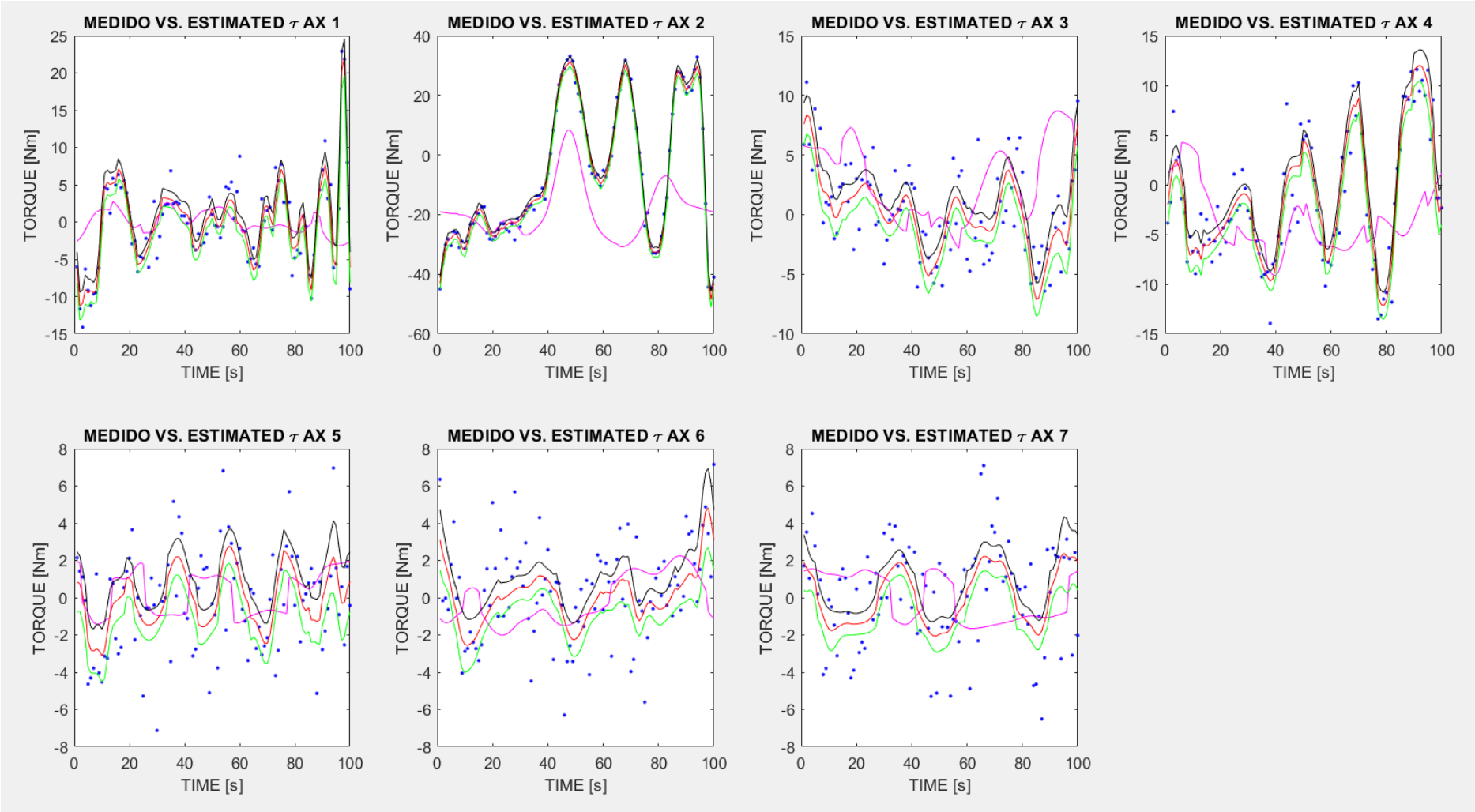


Fig 63: Mismas gráficas que en la figura anterior, oculatando la leyenda para mayor visibilidad

8 CONCLUSIONES

En este último capítulo de contenido analítico sobre el trabajo realizado, se plasman las conclusiones que se extraen de este trabajo.

Para hacerlo más claro y sencillo de seguir, se procede a enumerar las conclusiones que he sacado de todo este trabajo, punto por punto, deteniéndome en aquellos puntos que considero clave.

- Es fundamental la correcta definición de los puntos, bases, sistemas de referencia, sólidos y coordenadas generalizadas del mecanismo.

- El éxito en la obtención de los modelos cinemático y dinámicos, depende en gran medida del análisis previo que se realice, y la acotación llevada a cabo sobre el mecanismo.

- Puede resultar interesante realizar un diseño 3D simplificado del robot con las dimensiones reales, por si hay que inventarse a posteriori para hacer simulaciones, parámetros inerciales del robot. Éstos serán extraíbles de funciones generadas partiendo de funciones existentes de la librería de Dinámica de Sistemas Multicuerpo.

- A la hora de obtener las ecuaciones dinámicas, hay que tener en cuenta qué formulación se va a emplear, con las ventajas e inconvenientes que tiene asociadas, para ver cuál es más adecuada al problema que se presenta.

- Es fundamental realizar una definición de variables concreta y concisa, con pocos caracteres. De lo contrario los costes computacionales pueden ser altísimos y ralentizar mucho los cálculos y simulaciones.

- El modelo de fricción a considerar, conviene que sea una combinación entre modelo de fricción viscoso y de Coulomb, muy empleado en identificación de robots.

- Será interesante realizar una lectura de las ecuaciones buscando parámetros que no deberían estar porque no tienen repercusión, y aquellos que sí deberían estar.

- Debido a la dificultad que entraña la interpretación de las ecuaciones dinámicas, es recomendable fragmentar el problema en sólidos, comenzar con pocos sólidos y realizar simulaciones de comportamiento conocido (péndulos por ejemplo) con los sólidos del robot. Conforme se valide que los resultados son correctos para pequeño número de sólidos, ir aumentando el modelo incluyendo más sólidos, hasta alcanzar el número real de sólidos.

- Generar un código independiente para la lectura de las ecuaciones dinámicas obtenidas previamente, y almacenarlas como un conjunto de ecuaciones simbólicas, a las cuales se les dará uso en otro código que llama a este primero, para calcular las matrices a evaluar (K y tau). Será interesante en la generación de estas matrices, llevar a cabo simplificaciones sobre sus expresiones para aligerar posteriormente tiempos de ensamblado.

- Es conveniente parametrizar una trayectoria cualquiera basada en series de Fourier, de mismo número de armónicos que los que emplean los científicos que publican los artículos de la bibliografía (5 armónicos).

- La obtención de los parámetros base del modelo, puede hacerse con una trayectoria cualquiera.

- Se han de diseñar trayectorias óptimas (suficientemente excitantes para los parámetros a estimar), de forma que los parámetros tengan mucha importancia en el curso de la simulación. Suele ser interesante apurar las velocidades a los límites del robot.

- Cuando se apure la trayectoria a los límites del robot, habrá que considerar un factor de seguridad porque el tiempo de reacción, no es inmediato en los actuadores de los enlaces.

- Será necesario crear unas trayectorias de enlace entre posición de parado inicial e inicio de trayectoria óptima, y final de trayectoria óptima y posición de parado final. Esto puede hacerse haciendo uso de curvas de Bezier.

- Para inventar un caso de experimento ficticio (si no se dispone de robot), basta con inventar unos parámetros que tengan sentido, y distorisionar los valroes de pares reales con una función randomizadora, tomándolos como valores medidos. Con ellos puede llevarse a cabo la estimación.

- En el caso concreto de este trabajo, los experimentos se crean partiendo de trayectorias muy optimizadas y altamente excitantes.

- Los valores de los pares estimados son parejos a los reales, con ese efecto de desviación en la medida debida a errores incontrolables en un experimento real.

- Los valores de los parámetros estimados son aceptables, porque todos ellos están dentro de un rango prefijado como aceptable, en función de la desvicación en la estimación de estos parámetros.

- Los intervalos de confianza en la estimación de los parámetros son aceptables y siguen tendencias esperadas.

- La validación ha resultado satisfactoria, puesto que se obtienen valores similares de parámetros estimados con las dos trayectorias, y los residuos son de valores muy parejos.

- El criterio de optimización de trayectorias no es determinante. Los resultados son muy parejos.

- Es realmente complejo abordar un problema tan abstracto como lo es este, de identificación dinámica de un robot colaborativo.

- Los valores de los pares obtenidos son de un orden de magnitud lógico, y mayores en los ejes que se supone que deberían serlo. Se asemejan bastante a los valores de los pares obtenidos por los científicos en los artículos de la bibliografía. Quizá en este caso son algo más pequeños, probablemente por una invención de los parámetros para simular los experimentos no muy precisa, y por ser trayectorias sin cargas, o robots con configuraciones internas diferentes, que permitan exprimirlos más al máximo.

- Los residuos, o errores en la estimación de los pares, siguen una distribución normal según lo esperado, de valor bastante bueno, y que cuadra con el orden de magnitud previsible. Se parece mucho a los resultados obtenidos por los científicos en los artículos de la bibliografía.

Por último, a título personal, me gustaría dar mis conclusiones acerca de todo el trabajo realizado en este trabajo, y en el máster en general.

Soy consiente de que habré cometido errores a lo largo del proyecto, pero quería llevar a cabo un trabajo ambicioso y sensiblemente más complejo de lo habitual. Este máster tiene infinidad de asignaturas del ámbito mecánico, que pueden abordarse muy bien desde el punto de vista computacional.

En general, todas estas asignaturas llamaron mi atención, pero la que consiguió captar mi interés con más fuerza, fue Identificación de Sistemas Dinámicos. Tras haber trabajado en varias empresas a fecha de publicación de este trabajo, soy consciente de los problemas reales que supone para muchas empresas del ámbito industrial, el no poder predecir comportamientos de sus sistemas, sin otro remedio más que comprarlos y probarlos.

La asignatura de Identificación de Sistemas dinámicos, combinada con Dinámica de Sistemas Multicuerpo, me parece una sofisticada forma de abordar problemas convencionales con precisión, rigor y fundamento práctico y eficaz, para construir modelos, realizar análisis, estudiar situaciones, anticiparse a problemas, y obtener valores importantes difíciles de encontrar por catálogo.

Este trabajo ha sido largo, complejo, abstracto, se podría decir que hasta extenuante en algunos puntos. Pero ha valido la pena. Gracias a mi tutor, y todo lo aprendido en el máster, puedo decir que sé construir modelos dinámicos (con más o menos acierto), puedo analizar situaciones desde un punto de vista analítico, y he aprendido técnicas computacionales para solventar problemas realmente complejos. Confío en que este trabajo sirva a otras personas con idénticas inquietudes, y a pesar de haber cometido algún error, estoy contento.

9 BIBLIOGRAFÍA Y REFERENCIAS

[1] Y. R. Stürz, L. M. Affolter, and R. S. Smith, "Parameter Identification of the KUKA LBR iiwa Robot Including Constraints on Physical Feasibility," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6863–6868, 2017.

[2] C. Gaz, F. Flacco, and A. De Luca, "Identifying the dynamic model used by the KUKA LWR: A reverse engineering approach," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 1386–1392, 2014.

[3] T. Xu *et al.*, "Dynamic Identification of the KUKA LBR iiwa Robot with Retrieval of Physical Parameters Using Global Optimization," *IEEE Access*, vol. 8, no. June, pp. 108018–108031, 2020.

[4] A. Jubien, M. Gautier, and A. Janot, "Dynamic identification of the Kuka LWR robot using motor torques and joint torque sensors data," *IFAC Proc. Vol.*, vol. 19, pp. 8391–8396, 2014.

[5] A. Jubien, M. Gautier, and A. Janot, "Dynamic identification of the Kuka LightWeight robot: Comparison between actual and confidential Kuka's parameters," *IEEE/ASME Int. Conf. Adv. Intell. Mechatronics, AIM*, pp. 483–488, 2014.

[6] J. Swevers, C. Ganseman, D. Bilgin Tükel, J. De Schutter, and H. Van Brüssel, "Optimal robot excitation and identification," *IEEE Trans. Robot. Autom.*, vol. 13, no. 5, pp. 730–740, 1997.

[7] X. Iriarte Goñi, "Identificación de robots manipuladores: reducción de modelos y diseño de experimentos (Ph.D. Thesis)," 2010.

[8] KUKA, "KUKA Sensitive robotics_LBR iiwa," 2017.

[9] J. Ros, "Procedimientos en Dinámica de Sistemas Multicuerpo," 2019.

[10] D. X. I. Goñi, "Apuntes de Identificación de Sistemas Dinámicos," 2020.

10 ANEXOS

Se incluyen únicamente códigos y funciones principales. Funciones modificadas de la librería o subfunciones, se encuentran dentro de la carpeta de proyecto entregada. Para más información, consultar al autor.

TRABAJO_IKER_PELLEJERO_edit.m [Obtenición del modelo dinámico y simulaciones].

```
Do_you_want_to_run_the_script_again

%% Clean environment
close all
clear all
clc

%% Init lib_3D_MEC_MATLAB environment
init_lib_3D_MEC_MATLAB;

%----Begin Edit----
g=newParam('g',9.80665);
GravityVector=Vector3D([0,0,-g]','xyz');

%Integration step
Delta_t=0.001;

%Graphical Output refresh freq (n_frames/n_steps)
Delta_t_refresh=1/24;

% Assembly Init Problem solver parameters
Geom_Eq_init_tol=1.0e-10;
Geom_Eq_init_relax=0.1;

% Assembly Problem solver parameters
Geom_Eq_tol=Delta_t^2*10^-3;
Geom_Eq_relax=0.1;

% Equilibrium Problem solver parameters
Dyn_Eq_tol=1.0e-10;
Dyn_Eq_relax=0.1;

% Perturbed Dynamic State solver parameters
Per_Dyn_State_tol=1.0e-12;
Per_Dyn_State_relax=0.1;

% pop_up_dialogs=false;
optimize_matlabFunction=false;
updateDraws_automatic=true;
% equilibrium_problem_solved=false;
% pop_up_dialogs=false;
Set_characteristic_size(1);
n_traj=0;
%----End Edit----
```

```
purgeMSDFunctions;
purgeDraws;

fig_dir='fig';
mkdir(fig_dir);

% dyn_equations_type='Newton-Euler'
% dyn_equations_type='Virtual_Power';
dyn_equations_type='All';

Set_value(t,0);

%% Symbolic Procedures in Multibody Dynamics
%-----------------------------------------
%% Parametrization
%% Parametrization Kinematics
% Generalized coordinates, velocities and accelerations
%----Begin Edit----

newCoord('theta1',0,0);
newCoord('theta2',0,0);
newCoord('theta3',0,0);
newCoord('theta4',0,0);
newCoord('theta5',0,0);
newCoord('theta6',0,0);
newCoord('theta7',0,0);

%----End Edit----
q, dq, ddq, n_q

% Geometric Parameters
%----Begin Edit----

%Base
newParam('r1_base',85.5/1000);
newParam('r2_base',34.2/1000);
newParam('r_in_base',20/1000);
newParam('height_base',150/1000);
%Sol1
newParam('r_out_sol1',34.2/1000);
newParam('r_in_sol1',20/1000);
newParam('height_sol1',210/1000);
newParam('length_sol1',68/1000);
%Sol2
newParam('r_out_sol2',34.2/1000);
newParam('r_in_sol2',20/1000);
newParam('height_sol2',210/1000);
newParam('length_sol2',68/1000);
%Sol3
newParam('r_out_sol3',34.2/1000);
newParam('r_in_sol3',20/1000);
newParam('height_sol3',210/1000);
newParam('length_sol3',68/1000);
```

```
%Sol4
newParam('r_out_sol4',34.2/1000);
newParam('r_in_sol4',20/1000);
newParam('height_sol4',210/1000);
newParam('length_sol4',68/1000);
%Sol5
newParam('r_out_sol5',34.2/1000);
newParam('r_in_sol5',20/1000);
newParam('height_sol5',190/1000);
newParam('length_sol5',68/1000);
%Sol6
newParam('r_out_sol6',34.2/1000);
newParam('r_in_sol6',20/1000);
newParam('length_sol6',150/1000);
%Sol7
newParam('r_in_sol7',20/1000);
newParam('r1_out_sol7',34.2/1000);
newParam('r2_out_sol7',25/1000);
newParam('height_sol7',30/1000);

%----End Edit----
param, n_param

% Bases, Points and Frames
%----Begin Edit----

%Defining Bases
newBase('xyz','Base1',3,theta1);
newBase('Base1','Base2',2,theta2);
newBase('Base2','Base3',3,theta3);
newBase('Base3','Base4',2,theta4);
newBase('Base4','Base5',3,theta5);
newBase('Base5','Base6',2,theta6);
newBase('Base6','Base7',3,theta7);

%Defining Points
newPoint('O','P1',Vector3D([0,0,height_base]','xyz'));
newPoint('P1','P2',Vector3D([0,0,height_sol1]','Base1'));
newPoint('P2','P3',Vector3D([0,0,height_sol2]','Base2'));
newPoint('P3','P4',Vector3D([0,0,height_sol3]','Base3'));
newPoint('P4','P5',Vector3D([0,0,height_sol4]','Base4'));
newPoint('P5','P6',Vector3D([0,0,height_sol5]','Base5'));
newPoint('P6','P7',Vector3D([0,0,length_sol6/3]','Base6'));
newPoint('P7','PEndEffector',Vector3D([0,0,height_sol7]','Base7'
));

% Inertial Frame/Ground
newFrame('O','xyz','Gr');
newFrame('O','xyz','IF');
```

```
% Defining Frames
newFrame('O','xyz','FBase');
newFrame('P1','Base1','FSol1');
newFrame('P2','Base2','FSol2');
newFrame('P3','Base3','FSol3');
newFrame('P4','Base4','FSol4');
newFrame('P5','Base5','FSol5');
newFrame('P6','Base6','FSol6');
newFrame('P7','Base7','FSol7');


%% Define Solids
% rho=7850;

%
scad2stl('Plane','Plane','x',3,'y',8,'z',0.5,'x2',Get_value(W_in
t_base),'y2',8,'z2',Get_value(H_int_base));
% % newSTLDraw('Gr','Plane.stl',[0,1,1]);
% Ground=newSolid('Ground','Gr','Plane.stl',rho);
% newSolidDraw('Ground',[0,1,1]);
%
%
scad2stl('Base','Base','x',Get_value(Wbase),'y',Get_value(Lbase)
,'z',Get_value(Hbase),'x2',Get_value(W_int_base),'y2',Get_value(
Lbase),'z2',Get_value(H_int_base),'r_out',Get_value(1.02*Rarm),'
height',Get_value(h));
% %newSTLDraw('FArm1','Arm1.stl',[0,1,1]);
% Base=newSolid('Base','FBase','Base.stl',rho);
% newSolidDraw('Base',[0.5,0.4,0.5]);
%
%
scad2stl('Arm','Arm','r_out',Get_value(Rarm),'height',Get_value(
h-
Hbase),'length',Get_value(Larm),'ratio',Get_value(ratio),'R_supp
ort',Get_value(ratio));
% %newSTLDraw('FArm2','Arm2.stl',[1,0,1]);
% Arm=newSolid('Arm','FArm','Arm.stl',rho);
% newSolidDraw('Arm',[1,0,1]);
%
%
scad2stl('Disc','Disc','r_out',Get_value(Rdis),'r_in',Get_value(
1.3*Rarm),'width',Get_value(esp));
% %newSTLDraw('FDisc','Disc.stl',[0.7,0.7,0]);
% Disc=newSolid('Disc','FDisc','Disc.stl',rho);
% newSolidDraw('Disc',[0.7,0.7,0]);

%
scad2stl('Spring','Spring','height',(0.5*Get_value(h)),'Radius',
Get_value(Rsupport),'Wire_radius',0.005,'Coils',5)
% Spring=newSolid('FArm','Spring',rho,'Spring.stl');
% newSolidDraw('Spring',[0.40,0.35,0.2]);
```

```
Base=newSolid_without_STL('Base','FBase');
scad2stl('Base','Base','r_in',Get_value(r_in_base),'r_out1',Get_
value(r1_base),'r_out2',Get_value(r2_base),'height',Get_value(he
ight_base));
newSTLDraw('FBase','Base.stl',[0.7,0.7,0.7]);

Sol1=newSolid_without_STL('Sol1','FSol1');
scad2stl('Sol1','Sol1','r_in',Get_value(r_in_sol1),'r_out',Get_v
alue(r_out_sol1),'height',Get_value(height_sol1),'length',Get_va
lue(length_sol1));
newSTLDraw('FSol1','Sol1.stl',[1,0.5,0]);

Sol2=newSolid_without_STL('Sol2','FSol2');
scad2stl('Sol2','Sol2','r_in',Get_value(r_in_sol2),'r_out',Get_v
alue(r_out_sol2),'height',Get_value(height_sol2),'length',Get_va
lue(length_sol2));
newSTLDraw('FSol2','Sol2.stl',[0.95,0.95,0.95]);

Sol3=newSolid_without_STL('Sol3','FSol3');
scad2stl('Sol3','Sol3','r_in',Get_value(r_in_sol3),'r_out',Get_v
alue(r_out_sol3),'height',Get_value(height_sol3),'length',Get_va
lue(length_sol3));
newSTLDraw('FSol3','Sol3.stl',[1,0.5,0]);

Sol4=newSolid_without_STL('Sol4','FSol4');
scad2stl('Sol4','Sol4','r_in',Get_value(r_in_sol4),'r_out',Get_v
alue(r_out_sol4),'height',Get_value(height_sol4),'length',Get_va
lue(length_sol4));
newSTLDraw('FSol4','Sol4.stl',[0.95,0.95,0.95]);

Sol5=newSolid_without_STL('Sol5','FSol5');
scad2stl('Sol5','Sol5','r_in',Get_value(r_in_sol5),'r_out',Get_v
alue(r_out_sol5),'height',Get_value(height_sol5),'length',Get_va
lue(length_sol5));
newSTLDraw('FSol5','Sol5.stl',[1,0.5,0]);

Sol6=newSolid_without_STL('Sol6','FSol6');
scad2stl('Sol6','Sol6','r_in',Get_value(r_in_sol6),'r_out',Get_v
alue(r_out_sol6),'length',Get_value(length_sol6));
newSTLDraw('FSol6','Sol6.stl',[0.95,0.95,0.95]);

Sol7=newSolid_without_STL('Sol7','FSol7');
scad2stl('Sol7','Sol7','r_in',Get_value(r_in_sol7),'r_out1',Get_
value(r1_out_sol7),'r_out2',Get_value(r2_out_sol7),'height',Get_
value(height_sol7));
newSTLDraw('FSol7','Sol7.stl',[1,0.5,0]);

newFrameDraw('FSol1',0.5);
newFrameDraw('FSol2',0.5);
newFrameDraw('FSol3',0.5);
newFrameDraw('FSol4',0.5);
newFrameDraw('FSol5',0.5);
newFrameDraw('FSol6',0.5);
newFrameDraw('FSol7',0.5);
```

```
updateDraws;

%
scad2stl('Plane','Plane','r_out',Get_value(Rdis),'r_in',Get_valu
e(R2),'width',Get_value(esp));
% %newSTLDraw('FDisc','Disc.stl',[0.7,0.7,0]);
% Plane=newSolid('Fplane','PLane','Plane.stl',rho);
% newSolidDraw('Plane',[0.7,0.7,0]);

%
camlight('headlight');
% shading faceted;
material('dull');
azimut = 150; elevation = 10;
view(azimut,elevation);
camzoom(4);
camorbit(1,8);
% pause
% view(0,0,1)
% pause


% if Quest_YesNoStop('Do you want to visualize the
parametrization?','No')
%     if exist('./draw_dimensions.m','file')
%         %% Draw Dimensions
% %         azimut = 141.6; elevation = 22.4;
% %         view(azimut,elevation);
%         draw_dimensions;
%         load('extended_state');
%         updateDraws;
%         Help(['Graphical Output shows the parametrization.',
newline 'The situation is as-given (no Asembly or Initial
Assembly problem solved)']);
%         delete(dimensions_handle);
%     else
%         Warning('You need to provide dimension drawing
instructions on ''./draw_dimensions.m'' file') ;
%     end
% end

%% Constraint Equations
% % Constraint equations at the coordinate level (geometric)
% %----Begin Edit----
% e_theta_1=Vector3D([1,0,0]','Btheta');
% e_theta_2=Vector3D([0,1,0]','Btheta');
% e_theta_3=Vector3D([0,0,1]','Btheta');
%
% % a) Geometric equation
% newPhi((Get_value(Pos('O','P1'),'Base1')-
Get_value([0,0,height_base]','Base1')),false);
%
% % b) non-holonomic
```

```
%
% %----Begin Edit----
% Vel_P1_Base=Vel('IF','P1','FSol1');
%
% %FORMA 1
%
% e_Base1_1=Vector3D([1,0,0]','Base1');
% e_Base1_2=Vector3D([0,1,0]','Base1');
% e_Base1_3=Vector3D([0,0,1]','Base1');
% newdPhiNH([dot(Vel_P1_Base,e_Base1_1);
%            dot(Vel_P1_Base,e_Base1_2)],true);
% %----End Edit----
%
% Phi, dPhi, dPhiH, dPhiNH
% n_Phi, n_dPhi, n_dPhiH, n_dPhiNH
%
% % Export Assembly Problem
Export_Assembly_Problem;
assembly_problem;

%% Initial Assembly Problem
% % Additional Eqs for initial assembly problem
% % Coordinate level
% %----Begin Edit----
% theta0=newParam('theta0',0);%puedo probar con pi/50
% l0=newParam('l0',(-3.6));
% %----End Edit----
%
% newPhiInit([theta-theta0;
%            l-l0],true);
%
%
% % Velocity Level
% %----Begin Edit----
% vel0=newParam('vel0',2);
%
%
% newdPhiInit([dl-vel0],true);
%
% PhiInit, dPhiInit
% n_PhiInit, n_dPhiInit
%
% % Export Assembly Problem Init
% Export_Assembly_Problem_init;
% % assembly_problem_init;
%
% % %% Assembly Problem init
% % assembly_problem_init;
% % %
% % Help('Graphical Output shows the situation, after solving
Initial Assembly problem');

%% Wrench Definition
```

```
% Constraint Wrenches (Definition using helper function
newConstraint_Wrench)


%
Wrench_Ground_Base=newConstraint_Wrench([1,1,1],[1,1,1],'O','xyz
','Base','Gr');
%
Wrench_Base_Sol1=newConstraint_Wrench([1,1,1],[1,1,0],'P1','Base
1','Sol1','Base');
%
Wrench_Sol1_Sol2=newConstraint_Wrench([1,1,1],[1,0,1],'P2','Base
2','Sol2','Sol1');
%
Wrench_Sol2_Sol3=newConstraint_Wrench([1,1,1],[1,1,0],'P3','Base
3','Sol3','Sol2');
%
Wrench_Sol3_Sol4=newConstraint_Wrench([1,1,1],[1,0,1],'P4','Base
4','Sol4','Sol3');
%
Wrench_Sol4_Sol5=newConstraint_Wrench([1,1,1],[1,1,0],'P5','Base
5','Sol5','Sol4');
%
Wrench_Sol5_Sol6=newConstraint_Wrench([1,1,1],[1,0,1],'P6','Base
6','Sol6','Sol5');
%
Wrench_Sol6_Sol7=newConstraint_Wrench([1,1,1],[1,1,0],'P7','Base
7','Sol7','Sol6');

%----End Edit----

epsilon, n_epsilon
```

```
% Constitutive
%----Begin Edit----
% Viscous Base->Sol1 (rolling)
% theta1_eq=newParam('theta1_eq',0);
k_Base_Sol1 =newParam('k_Base_Sol1',300);
c_Base_Sol1=newParam('c_Base_Sol1',0.5);
cc1=newParam('cc1',1);
sign_dt1=newParam('sign_dt1',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Base_Sol1=Vector3D([0,0,0]','Base1');
MV_Base_Sol1_P1=Vector3D([0,0,(-c_Base_Sol1*dtheta1-
cc1*sign_dt1)]','Base1'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Base_Sol1_P1=Vector3D([0,0,(-c_Base_Sol1*dtheta1-
cc1*sign(dtheta1))]','Base1'); %OPCIÓN 2 (COMENTAR/DESCOMENTAR)


WrenchV_Base_Sol1=Vector6D(FV_Base_Sol1,MV_Base_Sol1_P1,'P1','So
l1'); %ACTION
WrenchV_Sol1_Base=Vector6D(-FV_Base_Sol1,-
MV_Base_Sol1_P1,'P1','Base'); %REACTION


% Viscous Sol1->Sol2 (rolling)
k_Sol1_Sol2 =newParam('k_Sol1_Sol2',300);
c_Sol1_Sol2=newParam('c_Sol1_Sol2',0.05);
cc2=newParam('cc2',1);
sign_dt2=newParam('sign_dt2',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Sol1_Sol2=Vector3D([0,0,0]','Base2');
MV_Sol1_Sol2_P2=Vector3D([0,(-c_Sol1_Sol2*dtheta2-
cc2*sign_dt2),0]','Base2'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol1_Sol2_P2=Vector3D([0,(-c_Sol1_Sol2*dtheta2-
cc2*sign(dtheta2)),0]','Base2'); %OPCIÓN 2
(COMENTAR/DESCOMENTAR)


WrenchV_Sol1_Sol2=Vector6D(FV_Sol1_Sol2,MV_Sol1_Sol2_P2,'P2','So
l2'); %ACTION
WrenchV_Sol2_Sol1=Vector6D(-FV_Sol1_Sol2,-
MV_Sol1_Sol2_P2,'P2','Sol1'); %REACTION
```

```
% Viscous Sol2->Sol3 (rolling)
k_Sol2_Sol3 =newParam('k_Sol2_Sol3',300);
c_Sol2_Sol3=newParam('c_Sol2_Sol3',0.5);
cc3=newParam('cc3',1);
sign_dt3=newParam('sign_dt3',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Sol2_Sol3=Vector3D([0,0,0]','Base3');
MV_Sol2_Sol3_P3=Vector3D([0,0,(-c_Sol2_Sol3*dtheta3-
cc3*sign_dt3)]','Base3'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol2_Sol3_P3=Vector3D([0,0,(-c_Sol2_Sol3*dtheta3-
cc3*sign(dtheta3))]','Base3'); %OPCIÓN 2 (COMENTAR/DESCOMENTAR)

WrenchV_Sol2_Sol3=Vector6D(FV_Sol2_Sol3,MV_Sol2_Sol3_P3,'P3','So
l3'); %ACTION
WrenchV_Sol3_Sol2=Vector6D(-FV_Sol2_Sol3,-
MV_Sol2_Sol3_P3,'P3','Sol2'); %REACTION


% Viscous Sol3->Sol4 (rolling)
k_Sol3_Sol4 =newParam('k_Sol3_Sol4',300);
c_Sol3_Sol4=newParam('c_Sol3_Sol4',0.05);
cc4=newParam('cc4',1);
sign_dt4=newParam('sign_dt4',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Sol3_Sol4=Vector3D([0,0,0]','Base4');
MV_Sol3_Sol4_P4=Vector3D([0,(-c_Sol3_Sol4*dtheta4-
cc4*sign_dt4),0]','Base4'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol3_Sol4_P4=Vector3D([0,(-c_Sol3_Sol4*dtheta4-
cc4*sign(dtheta4)),0]','Base4'); %OPCIÓN 2
(COMENTAR/DESCOMENTAR)

WrenchV_Sol3_Sol4=Vector6D(FV_Sol3_Sol4,MV_Sol3_Sol4_P4,'P4','So
l4'); %ACTION
WrenchV_Sol4_Sol3=Vector6D(-FV_Sol3_Sol4,-
MV_Sol3_Sol4_P4,'P4','Sol3'); %REACTION
```

```
% Viscous Sol4->Sol5 (rolling)
k_Sol4_Sol5 =newParam('k_Sol4_Sol5',300);
c_Sol4_Sol5=newParam('c_Sol4_Sol5',0.5);
cc5=newParam('cc5',1);
sign_dt5=newParam('sign_dt5',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Sol4_Sol5=Vector3D([0,0,0]','Base5');
MV_Sol4_Sol5_P5=Vector3D([0,0,(-c_Sol4_Sol5*dtheta5-
cc5*sign_dt5)]','Base5'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol4_Sol5_P5=Vector3D([0,0,(-c_Sol4_Sol5*dtheta5-
cc5*sign(dtheta5))]','Base5'); %OPCIÓN 2 (COMENTAR/DESCOMENTAR)

WrenchV_Sol4_Sol5=Vector6D(FV_Sol4_Sol5,MV_Sol4_Sol5_P5,'P5','So
l5'); %ACTION
WrenchV_Sol5_Sol4=Vector6D(-FV_Sol4_Sol5,-
MV_Sol4_Sol5_P5,'P5','Sol4'); %REACTION

% Viscous Sol5->Sol6 (rolling)
k_Sol5_Sol6 =newParam('k_Sol5_Sol6',300);
c_Sol5_Sol6=newParam('c_Sol5_Sol6',0.05);
cc6=newParam('cc6',1);
sign_dt6=newParam('sign_dt6',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Sol5_Sol6=Vector3D([0,0,0]','Base6');
MV_Sol5_Sol6_P6=Vector3D([0,(-c_Sol5_Sol6*dtheta6-
cc6*sign_dt6),0]','Base6'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol5_Sol6_P6=Vector3D([0,(-c_Sol5_Sol6*dtheta6-
cc6*sign(dtheta6)),0]','Base6'); %OPCIÓN 2
(COMENTAR/DESCOMENTAR)

WrenchV_Sol5_Sol6=Vector6D(FV_Sol5_Sol6,MV_Sol5_Sol6_P6,'P6','So
l6'); %ACTION
WrenchV_Sol6_Sol5=Vector6D(-FV_Sol5_Sol6,-
MV_Sol5_Sol6_P6,'P6','Sol5'); %REACTION

% Viscous Sol6->Sol7 (rolling)
k_Sol6_Sol7 =newParam('k_Sol6_Sol7',300);
c_Sol6_Sol7=newParam('c_Sol6_Sol7',0.5);
cc7=newParam('cc7',1);
sign_dt7=newParam('sign_dt7',0); %OPCIÓN 1
(COMENTAR/DESCOMENTAR)

FV_Sol6_Sol7=Vector3D([0,0,0]','Base7');
MV_Sol6_Sol7_P7=Vector3D([0,0,(-c_Sol6_Sol7*dtheta7-
cc7*sign_dt7)]','Base7'); %OPCIÓN 1 (COMENTAR/DESCOMENTAR)
% MV_Sol6_Sol7_P7=Vector3D([0,0,(-c_Sol6_Sol7*dtheta7-
cc7*sign(dtheta7))]','Base7'); %OPCIÓN 2 (COMENTAR/DESCOMENTAR)

WrenchV_Sol6_Sol7=Vector6D(FV_Sol6_Sol7,MV_Sol6_Sol7_P7,'P7','So
l7'); %ACTION
WrenchV_Sol7_Sol6=Vector6D(-FV_Sol6_Sol7,-
MV_Sol6_Sol7_P7,'P7','Sol6'); %REACTION
```

```
% Motors

% Motor Base->Sol1
% syms 'M1' 'real'
newParam('M1',0);
FMot_Base_Sol1=Vector3D([0,0,0]','Base1');
MMot_Base_Sol1_P1=Vector3D([0,0,M1]','Base1');%opcion1

WrenchMot_Base_Sol1=Vector6D(FMot_Base_Sol1,MMot_Base_Sol1_P1,'P
1','Sol1'); %ACTION
WrenchMot_Sol1_Base=Vector6D(-FMot_Base_Sol1,-
MMot_Base_Sol1_P1,'P1','Base'); %REACTION


% Motor Sol1->Sol2
% syms 'M2' 'real'
newParam('M2',0);
FMot_Sol1_Sol2=Vector3D([0,0,0]','Base2');
MMot_Sol1_Sol2_P2=Vector3D([0,M2,0]','Base2');%opcion1

WrenchMot_Sol1_Sol2=Vector6D(FMot_Sol1_Sol2,MMot_Sol1_Sol2_P2,'P
2','Sol2'); %ACTION
WrenchMot_Sol2_Sol1=Vector6D(-FMot_Sol1_Sol2,-
MMot_Sol1_Sol2_P2,'P2','Sol1'); %REACTION


% Motor Sol2->Sol3
% syms 'M3' 'real'
newParam('M3',0);
FMot_Sol2_Sol3=Vector3D([0,0,0]','Base3');
MMot_Sol2_Sol3_P3=Vector3D([0,0,M3]','Base3');%opcion1

WrenchMot_Sol2_Sol3=Vector6D(FMot_Sol2_Sol3,MMot_Sol2_Sol3_P3,'P
3','Sol3'); %ACTION
WrenchMot_Sol3_Sol2=Vector6D(-FMot_Sol2_Sol3,-
MMot_Sol2_Sol3_P3,'P3','Sol2'); %REACTION


% Motor Sol3->Sol4
% syms 'M4' 'real'
newParam('M4',0);
FMot_Sol3_Sol4=Vector3D([0,0,0]','Base4');
MMot_Sol3_Sol4_P4=Vector3D([0,M4,0]','Base4');%opcion1

WrenchMot_Sol3_Sol4=Vector6D(FMot_Sol3_Sol4,MMot_Sol3_Sol4_P4,'P
4','Sol4'); %ACTION
WrenchMot_Sol4_Sol3=Vector6D(-FMot_Sol3_Sol4,-
MMot_Sol3_Sol4_P4,'P4','Sol3'); %REACTION
```

```
% Motor Sol4->Sol5
% syms 'M5' 'real'
newParam('M5',0);
FMot_Sol4_Sol5=Vector3D([0,0,0]','Base5');
MMot_Sol4_Sol5_P5=Vector3D([0,0,M5]','Base5');%opcion1

WrenchMot_Sol4_Sol5=Vector6D(FMot_Sol4_Sol5,MMot_Sol4_Sol5_P5,'P
5','Sol5'); %ACTION
WrenchMot_Sol5_Sol4=Vector6D(-FMot_Sol4_Sol5,-
MMot_Sol4_Sol5_P5,'P5','Sol4'); %REACTION


% Motor Sol5->Sol6
% syms 'M6' 'real'
newParam('M6',0);
FMot_Sol5_Sol6=Vector3D([0,0,0]','Base6');
MMot_Sol5_Sol6_P6=Vector3D([0,M6,0]','Base6');%opcion1

WrenchMot_Sol5_Sol6=Vector6D(FMot_Sol5_Sol6,MMot_Sol5_Sol6_P6,'P
6','Sol6'); %ACTION
WrenchMot_Sol6_Sol5=Vector6D(-FMot_Sol5_Sol6,-
MMot_Sol5_Sol6_P6,'P6','Sol5'); %REACTION


% Motor Sol6->Sol7
% syms 'M7' 'real'
newParam('M7',0);
FMot_Sol6_Sol7=Vector3D([0,0,0]','Base7');
MMot_Sol6_Sol7_P7=Vector3D([0,0,M7]','Base7');%opcion1

WrenchMot_Sol6_Sol7=Vector6D(FMot_Sol6_Sol7,MMot_Sol6_Sol7_P7,'P
7','Sol7'); %ACTION
WrenchMot_Sol7_Sol6=Vector6D(-FMot_Sol6_Sol7,-
MMot_Sol6_Sol7_P7,'P7','Sol6'); %REACTION


% Inertia
WrenchIG_Base=newInertia_and_Gravity_Wrench('Base');
WrenchIG_Sol1=newInertia_and_Gravity_Wrench('Sol1');
WrenchIG_Sol2=newInertia_and_Gravity_Wrench('Sol2');
WrenchIG_Sol3=newInertia_and_Gravity_Wrench('Sol3');
WrenchIG_Sol4=newInertia_and_Gravity_Wrench('Sol4');
WrenchIG_Sol5=newInertia_and_Gravity_Wrench('Sol5');
WrenchIG_Sol6=newInertia_and_Gravity_Wrench('Sol6');
WrenchIG_Sol7=newInertia_and_Gravity_Wrench('Sol7');

% torques=[M1;M2;M3;M4;M5;M6;M7];
%% Draw Forces and Moments
% %----Begin Edit----
% FC_Ground_Disc=Wrench_Ground_Disc.FW; %Probar a calcular la
otra
% FCV_Ground_Disc=WrenchV_GrDi.FW;
%
```

```
% newVectorDraw('J','FC_Ground_Disc',[-1.5,-
1.5,0.05*1],[1,0,0]);
% newVectorDraw('J','FCV_Ground_Disc',[2,2,6],[0,0,1]);
%
% Vel_O1_Base=Vel('Gr','O1','FBase');
%
% newVectorDraw('O1','Vel_O1_Base',[2.5,2.5,5],[1,1,0]);
%----End Edit----

%Show Kinematic + Dynamic parameters
param, n_param

%% Newton-Euler Dynamic Equations
% using Screws (Vector6D)
%----Begin Edit----
% Base
Sum_Wrenches_Base_NE=
WrenchIG_Base+WrenchV_Sol1_Base+WrenchMot_Sol1_Base;
%+Wrench_Ground_Base-Wrench_Base_Sol1;

% Sol1
Sum_Wrenches_Sol1_NE=
WrenchIG_Sol1+WrenchV_Base_Sol1+WrenchV_Sol2_Sol1+WrenchMot_Base
_Sol1+WrenchMot_Sol2_Sol1; %+Wrench_Base_Sol1-Wrench_Sol1_Sol2;

% Sol2
Sum_Wrenches_Sol2_NE=
WrenchIG_Sol2+WrenchV_Sol1_Sol2+WrenchV_Sol3_Sol2+WrenchMot_Sol1
_Sol2+WrenchMot_Sol3_Sol2; %+Wrench_Sol1_Sol2-Wrench_Sol2_Sol3;

% Sol3
Sum_Wrenches_Sol3_NE=
WrenchIG_Sol3+WrenchV_Sol2_Sol3+WrenchV_Sol4_Sol3+WrenchMot_Sol2
_Sol3+WrenchMot_Sol4_Sol3; %+Wrench_Sol2_Sol3-Wrench_Sol3_Sol4;

% Sol4
Sum_Wrenches_Sol4_NE=
WrenchIG_Sol4+WrenchV_Sol3_Sol4+WrenchV_Sol5_Sol4+WrenchMot_Sol3
_Sol4+WrenchMot_Sol5_Sol4; %+Wrench_Sol3_Sol4-Wrench_Sol4_Sol5;

% Sol5
Sum_Wrenches_Sol5_NE=
WrenchIG_Sol5+WrenchV_Sol4_Sol5+WrenchV_Sol6_Sol5+WrenchMot_Sol4
_Sol5+WrenchMot_Sol6_Sol5; %+Wrench_Sol4_Sol5-Wrench_Sol5_Sol6;

% Sol6
Sum_Wrenches_Sol6_NE=
WrenchIG_Sol6+WrenchV_Sol5_Sol6+WrenchV_Sol7_Sol6+WrenchMot_Sol5
_Sol6+WrenchMot_Sol7_Sol6; %+Wrench_Sol5_Sol6-Wrench_Sol6_Sol7;

% Sol7
Sum_Wrenches_Sol7_NE=  WrenchIG_Sol7+WrenchV_Sol6_Sol7-
WrenchMot_Sol6_Sol7; %+Wrench_Sol6_Sol7;
```

```
%----End Edit----

%% Newton-Euler
if strcmp(dyn_equations_type,'All') ||
strcmp(dyn_equations_type,'Newton-Euler')
    %----Begin Edit----
    Dyn_eq_NE=[Sum_Wrenches_Base_NE;
        Sum_Wrenches_Sol1_NE;
        Sum_Wrenches_Sol2_NE;
        Sum_Wrenches_Sol3_NE;
        Sum_Wrenches_Sol4_NE;
        Sum_Wrenches_Sol5_NE;
        Sum_Wrenches_Sol6_NE;
        Sum_Wrenches_Sol7_NE]
        %----End Edit----
%     %So mass matrix is positive definite
%     Dyn_eq_NE=-Dyn_eq_NE; % Dyn_eq_NE=simplify(Dyn_eq_NE);
%     FC_vepsilon=-jacobian(Dyn_eq_NE,epsilon);
FC_vepsilon=simplify(FC_vepsilon);
end

save Dyn_eq_NE Dyn_eq_NE

%% Virtual Power or Lagrange
if strcmp(dyn_equations_type,'All') ||
strcmp(dyn_equations_type,'Virtual_Power') ||
strcmp(dyn_equations_type,'Lagrange')
    %----Begin Edit----
    Dyn_eq_VP=sym(zeros(n_q,1));

    Twist_Base = Twist('Base');
    Twist_Sol1 = Twist('Sol1');
    Twist_Sol2 = Twist('Sol2');
    Twist_Sol3 = Twist('Sol3');
    Twist_Sol4 = Twist('Sol4');
    Twist_Sol5 = Twist('Sol5');
    Twist_Sol6 = Twist('Sol6');
    Twist_Sol7 = Twist('Sol7');
```

```
    for i=1:n_q
        Dyn_eq_VP(i,1)=dot( Sum_Wrenches_Base_NE , diff(
Twist_Base , dq(i) ) )+...
            dot( Sum_Wrenches_Sol1_NE , diff( Twist_Sol1 , dq(i)
) )+...
            dot( Sum_Wrenches_Sol2_NE , diff( Twist_Sol2 , dq(i)
) )+...
            dot( Sum_Wrenches_Sol3_NE , diff( Twist_Sol3 , dq(i)
) )+...
            dot( Sum_Wrenches_Sol4_NE , diff( Twist_Sol4 , dq(i)
) )+...
            dot( Sum_Wrenches_Sol5_NE , diff( Twist_Sol5 , dq(i)
) )+...
            dot( Sum_Wrenches_Sol6_NE , diff( Twist_Sol6 , dq(i)
) )+...
            dot( Sum_Wrenches_Sol7_NE , diff( Twist_Sol7 , dq(i)
) );
    end

    save Dyn_eq_VP Dyn_eq_VP

    %----End Edit----

%     %So mass matrix is positive definite
%     Dyn_eq_VP=-Dyn_eq_VP;
%     FC_qepsilon=-jacobian(Dyn_eq_VP,epsilon);
FC_qepsilon=simplify(FC_qepsilon);

%     %% Define Lagrange multipliers
%     newLambdas();
%     lambda, n_lambda
%
%
Dyn_eq_VP_open=subs(Dyn_eq_VP,epsilon,sym(zeros(size(epsilon))))
; %Dyn_eq_VP_open=simplify(Dyn_eq_VP_open);
%     Dyn_eq_L=Dyn_eq_VP_open+dPhi_dq'*lambda;
end

%% Inertial properties obtaining
rho=7850;

fv_Base = stlRead('Base.stl');
%
[mass_Base,first_moment_of_mass_Base,inertia_tensor_Base]=inert_
prop(fv_Base,rho);
[mass_s_Base,first_mass_moment_s_Base,inertia_tensor_s_Base,mass
_n_Base,first_mass_moment_n_Base,inertia_tensor_n_Base,param_s_B
ase,param_n_Base]=round_inert_prop_MODIFICADO(fv_Base,rho,'Base'
);
```

```
    for i=1:length(param_s_Base)
        newParam(char(param_s_Base(i)),param_n_Base(i));
    end


fv_Sol1 = stlRead('Sol1.stl');
%
[mass_Sol1,first_moment_of_mass_Sol1,inertia_tensor_Sol1]=inert_
prop(fv_Sol1,rho);
[mass_s_1,first_mass_moment_s_1,inertia_tensor_s_1,mass_n_1,firs
t_mass_moment_n_1,inertia_tensor_n_1,param_s_1,param_n_1]=round_
inert_prop_MODIFICADO(fv_Sol1,rho,'Sol1');
    for i=1:length(param_s_1)
        newParam(char(param_s_1(i)),param_n_1(i));
    end


fv_Sol2 = stlRead('Sol2.stl');
%
[mass_Sol2,first_moment_of_mass_Sol2,inertia_tensor_Sol2]=inert_
prop(fv_Sol2,rho);
[mass_s_2,first_mass_moment_s_2,inertia_tensor_s_2,mass_n_2,firs
t_mass_moment_n_2,inertia_tensor_n_2,param_s_2,param_n_2]=round_
inert_prop_MODIFICADO(fv_Sol2,rho,'Sol2');
    for i=1:length(param_s_2)
        newParam(char(param_s_2(i)),param_n_2(i));
    end


fv_Sol3 = stlRead('Sol3.stl');
%
[mass_Sol3,first_moment_of_mass_Sol3,inertia_tensor_Sol3]=inert_
prop(fv_Sol3,rho);
[mass_s_3,first_mass_moment_s_3,inertia_tensor_s_3,mass_n_3,firs
t_mass_moment_n_3,inertia_tensor_n_3,param_s_3,param_n_3]=round_
inert_prop_MODIFICADO(fv_Sol3,rho,'Sol3');
    for i=1:length(param_s_3)
        newParam(char(param_s_3(i)),param_n_3(i));
    end


fv_Sol4 = stlRead('Sol4.stl');
%
[mass_Sol4,first_moment_of_mass_Sol4,inertia_tensor_Sol4]=inert_
prop(fv_Sol4,rho);
[mass_s_4,first_mass_moment_s_4,inertia_tensor_s_4,mass_n_4,firs
t_mass_moment_n_4,inertia_tensor_n_4,param_s_4,param_n_4]=round_
inert_prop_MODIFICADO(fv_Sol4,rho,'Sol4');
    for i=1:length(param_s_4)
        newParam(char(param_s_4(i)),param_n_4(i));
    end
```

```
fv_Sol5 = stlRead('Sol5.stl');
%
[mass_Sol5,first_moment_of_mass_Sol5,inertia_tensor_Sol5]=inert_
prop(fv_Sol5,rho);
[mass_s_5,first_mass_moment_s_5,inertia_tensor_s_5,mass_n_5,firs
t_mass_moment_n_5,inertia_tensor_n_5,param_s_5,param_n_5]=round_
inert_prop_MODIFICADO(fv_Sol5,rho,'Sol5');
    for i=1:length(param_s_5)
        newParam(char(param_s_5(i)),param_n_5(i));
    end


fv_Sol6 = stlRead('Sol6.stl');
%
[mass_Sol6,first_moment_of_mass_Sol6,inertia_tensor_Sol6]=inert_
prop(fv_Sol6,rho);
[mass_s_6,first_mass_moment_s_6,inertia_tensor_s_6,mass_n_6,firs
t_mass_moment_n_6,inertia_tensor_n_6,param_s_6,param_n_6]=round_
inert_prop_MODIFICADO(fv_Sol6,rho,'Sol6');
    for i=1:length(param_s_6)
        newParam(char(param_s_6(i)),param_n_6(i));
    end


fv_Sol7 = stlRead('Sol7.stl');
%
[mass_Sol7,first_moment_of_mass_Sol7,inertia_tensor_Sol7]=inert_
prop(fv_Sol7,rho);
[mass_s_7,first_mass_moment_s_7,inertia_tensor_s,mass_n_7,first_
mass_moment_n_7,inertia_tensor_n_7,param_s_7,param_n_7]=round_in
ert_prop_MODIFICADO(fv_Sol7,rho,'Sol7');
    for i=1:length(param_s_7)
        newParam(char(param_s_7(i)),param_n_7(i));
    end

param_s_exp=[param_s_1;param_s_2;param_s_3;param_s_4;param_s_5;p
aram_s_6;param_s_7;c_Base_Sol1;c_Sol1_Sol2;c_Sol2_Sol3;c_Sol3_So
l4;c_Sol4_Sol5;c_Sol5_Sol6;c_Sol6_Sol7;cc1;cc2;cc3;cc4;cc5;cc6;c
c7];
param_n_exp=[param_n_1;param_n_2;param_n_3;param_n_4;param_n_5;p
aram_n_6;param_n_7;0.5;0.5;0.5;0.5;0.5;0.5;0.5;0.8;0.8;0.8;0.8;0
.8;0.8;0.8];

save param_s_exp param_s_exp
save param_n_exp param_n_exp

%     M1=eval(subs(M1,0));
%     M2=eval(subs(M2,0));
%     M3=eval(subs(M3,0));
%     M4=eval(subs(M4,0));
%     M5=eval(subs(M5,0));
%     M6=eval(subs(M6,0));
%     M7=eval(subs(M7,0));
%
```

```
%% Export Direct Dynamics Problem
 Export_Direct_Dynamics_Problem;


%  %% Dynamic Equilibrium Equations
% %----Begin Edit----
% Extra_Dyn_Eq_eq=[theta;
%     dtheta];
% %----End Edit----
%
% %%  Equilibrium problem "a la" Newton-Euler
%     Dyn_Eq_eq_NE=[Dyn_eq_NE;
%         ddPhi;
%         dPhi;
%         Phi;
%         Extra_Dyn_Eq_eq];
%
% %%  Equilibrium problem "a la" Virtual Power or Lagrange
%      Dyn_Eq_eq_VP=[Dyn_eq_L;
%         ddPhi;
%         dPhi;
%             Phi;
%             Extra_Dyn_Eq_eq];
%
% %% Export Dynamic Equilibrium Problem
% Export_Equilibrium_Problem;
%
% %% Export Perturbed Dynamic Equilibrium Problem
% Export_Perturbed_Dynamic_State_Problem;
%
% theta_per=newParam('theta_per',0);
%
% % newParam('dpsi0',1);
% Extra_Perturbed_Dyn_State_eq=[theta-theta_per;
%     dtheta];
% %                                 ddl];
% %                                 dtheta];
%
% Perturbed_Dyn_State_eq_NE=[Dyn_eq_NE;
%     ddPhi;
%     dPhi;
%     Phi;
%     Extra_Perturbed_Dyn_State_eq];
%
% Perturbed_Dyn_State_eq_VP=[Dyn_eq_L;
%     ddPhi;
%     dPhi;
%     Phi;
%     Extra_Perturbed_Dyn_State_eq];
%
% %% Export Perturbed Dynamic Equilibrium Problem
% Export_Perturbed_Dynamic_State_Problem;
%
% %% Export Eigenvalue Problem
```

```
% Export_Eigenvalue_Problem;
%
%% Numerical Procedures in Multibody Dynamics
%-------------------------------------------
dyn_equations_type='Virtual_Power';

% %% Assembly Problem init
% assembly_problem_init;
%
% %% Equilibrium Problem
% assembly_problem;
%
% direct_dynamics_problem;
% dynamic_equilibrium_problem;
% updateDraws
% eigenvalue_problem;
% showEigenvalues(eigenvalues);
%
% %% Perturbed Dynamic State Problem
% Set_value(theta_per,Get_value(theta)+10*pi/180)
% perturbed_dynamic_state_problem

%% Kinematic Simulation  (from current state)
% T_span=5;
% t_end=Get_value(t)+T_span;
% t_value_ant=Get_value(t);
% fid = fopen('kinematics_extended_state_series.txt','w');
% printf_extended_state(fid);
%
% updateDraws_automatic=false;
%
% dynamic_simulation_Euler;
%
% updateDraws_automatic=true;
%
% fclose(fid);

%% Dynamic Simulation
% Setup time step and integration span
% Each time you run this section you get T_span additional time
of
% simulation
T_span=6
t_end=Get_value(t)+T_span;
t_value_ant=Get_value(t);

Geom_Eq_tol=Delta_t^2*10^-3;
Geom_Eq_relax=1;

updateDraws_automatic=false;

fid = fopen('dynamics_extended_state_series.txt','w');
printf_extended_state(fid);
```

```
dynamic_simulation_Euler_MODIFICADO;

fclose(fid);

updateDraws_automatic=true;


load('dynamics_extended_state_series.txt');

% figure;
% vector_tiempo=dynamics_extended_state_series(:,1);
% vector_theta=dynamics_extended_state_series(:,2);
% plot(vector_tiempo,vector_theta);
% xlabel('time [s]');
% ylabel('theta [rad]');
% title('Evolución del ángulo theta con el tiempo')

load('Dyn_eq_VP.mat');
eqs = fopen('Dyn_eq_VP.txt','wt+');
% fprintf(fileID,'%6s %12s\n','x','exp(x)');
% fprintf(fileID,'%6.2f %12.8f\n',A);
for i=1:length(Dyn_eq_VP)
fprintf(eqs,'%1s\n',strcat('eq',num2str(i)));
fprintf(eqs,char(Dyn_eq_VP(i,1)));
fprintf(eqs,'\n\n\n');
end
fclose(eqs);
```

Dynamic_Equations.m [Escritura simbólica y generación de ecuaciones dinámicas]

```
clear all
close all
%  symbolic definition of the coordinates and their derivatives
syms t1 t2 t3 t4 t5 t6 t7 real
syms dt1 dt2 dt3 dt4 dt5 dt6 dt7 real
syms ddt1 ddt2 ddt3 ddt4 ddt5 ddt6 ddt7 real

%  symbolic definition of the inercial parameters
syms m_Base mx_Base my_Base mz_Base Ixx_Base Ixy_Base Ixz_Base
Iyy_Base Iyz_Base Izz_Base real
syms m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 real
syms m2 mx2 my2 mz2 Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 real
syms m3 mx3 my3 mz3 Ixx3 Ixy3 Ixz3 Iyy3 Iyz3 Izz3 real
syms m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4 Izz4 real
syms m5 mx5 my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 real
syms m6 mx6 my6 mz6 Ixx6 Ixy6 Ixz6 Iyy6 Iyz6 Izz6 real
syms m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7 Iyz7 Izz7 real

%  symbolic definition of the friction parameters
syms c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6 cc7 s1 s2 s3
s4 s5 s6 s7 real

%  symbolic definition of the geometric parameters
syms rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3 ri3 h3 l3
ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri7 r1o7 r2o7 h7 real

%  symbolic definition of the input torques
syms M1 M2 M3 M4 M5 M6 M7 real

%  symbolic definition of the gravity constant
syms g real

eq1=…

eq2=…

eq3=…

eq4=…

eq5=…

eq6=…

eq7=…

% Kinematic equations (accelerations)
    % No existen en este caso

% Vector of dynamic equations
Eq_Dyn=-[eq1;eq2;eq3;eq4;eq5;eq6;eq7];
% Eq_Dyn=simplify(Eq_Dyn);

save Eq_Dyn Eq_Dyn
```

get_symbolic_K_symplify.m [Obtención de matrices para evaluar en identificación]

```
Dynamic_Equations
%  symbolic definition of the coordinates and their derivatives
syms t1 t2 t3 t4 t5 t6 t7 real
syms dt1 dt2 dt3 dt4 dt5 dt6 dt7 real
syms ddt1 ddt2 ddt3 ddt4 ddt5 ddt6 ddt7 real
    q=[t1 t2 t3 t4 t5 t6 t7]';
    dq=[dt1 dt2 dt3 dt4 dt5 dt6 dt7]';
    ddq=[ddt1 ddt2 ddt3 ddt4 ddt5 ddt6 ddt7]';

%  symbolic definition of the sign of the generalized velocities
syms s1 s2 s3 s4 s5 s6 s7 real
s_dq=[s1 s2 s3 s4 s5 s6 s7]';

%  symbolic definition of the input torques
syms M1 M2 M3 M4 M5 M6 M7 real
    torques=[M1 M2 M3 M4 M5 M6 M7]';

%  symbolic definition of the geometric parameters
syms rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3 ri3 h3 l3
ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri7 r1o7 r2o7 h7 real
%  symbolic definition of the gravity constant
syms g real
    param_geom=[rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3
ri3 h3 l3 ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri7 r1o7 r2o7
h7 g]';


%  symbolic definition of the inercial parameters
syms m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 m2 mx2 my2 mz2
Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3 Ixy3 Ixz3 Iyy3
Iyz3 Izz3 m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4 Izz4 m5 mx5
my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 m6 mx6 my6 mz6 Ixx6 Ixy6
Ixz6 Iyy6 Iyz6 Izz6 m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7 Iyz7 Izz7
c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6 cc7 real
    param=[ m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 m2 mx2
my2 mz2 Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3 Ixy3
Ixz3 Iyy3 Iyz3 Izz3 m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4 Izz4
m5 mx5 my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 m6 mx6 my6 mz6 Ixx6
Ixy6 Ixz6 Iyy6 Iyz6 Izz6 m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7 Iyz7
Izz7 c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6 cc7]';

load Eq_Dyn.mat
```

```
for i=1:length(param)
    tic
    param_i=[param(1:i-1,1);param(i+1:end)];
    fprintf(['Subsitute all params but ',num2str(i),' by 0\n']);
    Eq_i=subs(Eq_Dyn,param_i,zeros(size(param_i)));
    fprintf('Simplifying Eq_Dyn\n');
    Eq_i=simplify(Eq_i);
    fprintf('Calculating Ki\n');
    Ki=jacobian(Eq_i,param(i,1));
    fprintf('Simplifying Ki\n');
    Ki=simplify(Ki);
    K(:,i)=Ki;
    save K K
    toc
end

tau= - subs(Eq_Dyn,param,zeros(size(param)));
save tau tau;

tic,fprintf('Generating evalK\n');
matlabFunction(K,'File','evalK','vars',{q,dq,ddq,s_dq,param_geom
})
toc

tic,fprintf('Generating evaltau\n');
matlabFunction(tau,'File','evaltau','vars',{q,torques})
toc
```

traj_and_phi_b_obtaining.m [Parametrización de trayectoria aleatoria y obtención de expresiones de parámetros base]

```
% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% number of points
n=100;

% fundamental frequency
f=0.1;

% crit='cond';
crit='dopt';

if crit=='cond'
    load best_cond.mat;
    best_value=best_cond;
elseif crit=='dopt'
    load best_dopt.mat;
    best_value=best_dopt;
end

% cont=0;
%
% while cont<10
%     cont=cont+1;
%     rng('default');
    param_traj=2*pi*rand((2*n_h+1)*n_dof,1);

%     options=optimoptions('Display','iter','maxiter',10000);

    W=get_W_with_param_traj(param_traj,n,f,crit);
% end

save W W
% load('W.mat');
% MatrizW = fopen('W.txt','wt+');
% % fprintf(fileID,'%6s %12s\n','x','exp(x)');
% % fprintf(fileID,'%6.2f %12.8f\n',A);
% for i=1:length(W)
%     tic
%     disp(['Escribiendo fila ',num2str(i)])
%     fprintf(MatrizW,'%1f\n',strcat('Fila',num2str(i)));
%     fprintf(MatrizW,char(W(i,:)));
%     fprintf(MatrizW,'\n');
%     toc
% end
% fclose(MatrizW);

[phi_b,Wb]=QR_reduction(W);
phi_b
phi_b_definitivo=vpa(phi_b,4);
```

QR_reduction.m [Reducción QR]

```
function [phi_b_num,Wb,v]=QR_reduction(W)

tic
disp('Obteniendo matrices Q, R y E')
[Q,R,E]=qr(W);
toc

tic
disp('Calculando el rango de W')
r=rank(W);
disp(['El rango de W es ',num2str(r)]);
toc

%El tamaño de la matriz W es nxp
n=size(W,1);
p=size(W,2);

if r<p
    for i=r+1:p
    R(i,i)=0;
    end

    tic
    disp('Obteniendo matrices R1 y R2')
    R1=R(1:r,1:r);
    R2=R(1:r,r+1:end);
    toc

    tic
    disp('Calculando beta')
    beta=inv(R1)*R2;
    toc

    tic
    disp('Obteniendo matrices W1 y W2')
    W1=W*E(:,1:r);
    w2=W*E(:,r+1:end);
    toc

    tic
    disp('Obteniendo matrices Q1 y Q2')
    Q1=Q(:,1:r);
    Q2=Q(:,r+1:end);

    %  symbolic definition of the inercial parameters
%     syms m_Base mx_Base my_Base mz_Base Ixx_Base Ixy_Base
Ixz_Base Iyy_Base Iyz_Base Izz_Base real
%     syms m_Sol1 mx_Sol1 my_Sol1 mz_Sol1 Ixx_Sol1 Ixy_Sol1
Ixz_Sol1 Iyy_Sol1 Iyz_Sol1 Izz_Sol1 real
%     syms m_Sol2 mx_Sol2 my_Sol2 mz_Sol2 Ixx_Sol2 Ixy_Sol2
Ixz_Sol2 Iyy_Sol2 Iyz_Sol2 Izz_Sol2 real
```

```
%     syms m_Sol3 mx_Sol3 my_Sol3 mz_Sol3 Ixx_Sol3 Ixy_Sol3
Ixz_Sol3 Iyy_Sol3 Iyz_Sol3 Izz_Sol3 real
%     syms m_Sol4 mx_Sol4 my_Sol4 mz_Sol4 Ixx_Sol4 Ixy_Sol4
Ixz_Sol4 Iyy_Sol4 Iyz_Sol4 Izz_Sol4 real
%     syms m_Sol5 mx_Sol5 my_Sol5 mz_Sol5 Ixx_Sol5 Ixy_Sol5
Ixz_Sol5 Iyy_Sol5 Iyz_Sol5 Izz_Sol5 real
%     syms m_Sol6 mx_Sol6 my_Sol6 mz_Sol6 Ixx_Sol6 Ixy_Sol6
Ixz_Sol6 Iyy_Sol6 Iyz_Sol6 Izz_Sol6 real
%     syms m_Sol7 mx_Sol7 my_Sol7 mz_Sol7 Ixx_Sol7 Ixy_Sol7
Ixz_Sol7 Iyy_Sol7 Iyz_Sol7 Izz_Sol7 real
syms m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 m2 mx2 my2 mz2
Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3 Ixy3 Ixz3 Iyy3
Iyz3 Izz3 m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4 Izz4 m5 mx5
my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 m6 mx6 my6 mz6 Ixx6 Ixy6
Ixz6 Iyy6 Iyz6 Izz6 m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7 Iyz7 Izz7
real


    %  symbolic definition of the friction parameters
%     syms c_Base_Sol1 c_Sol1_Sol2 c_Sol2_Sol3 c_Sol3_Sol4
c_Sol4_Sol5 c_Sol5_Sol6 c_Sol6_Sol7 real
syms c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6 cc7 real

    %  symbolic definition of the geometric parameters
%     syms r1_base r2_base r_in_base height_base r_out_sol1
r_in_sol1 height_sol1 length_sol1 r_out_sol2 r_in_sol2
height_sol2 length_sol2 r_out_sol3 r_in_sol3 height_sol3
length_sol3 r_out_sol4 r_in_sol4 height_sol4 length_sol4
r_out_sol5 r_in_sol5 height_sol5 length_sol5 r_out_sol6
r_in_sol6 length_sol6 r_in_sol7 r1_out_sol7 r2_out_sol7
height_sol7 real
syms rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3 ri3 h3 l3
ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri7 r1o7 r2o7 h7 g real



    % Inertial parameters vector (only the ones that appear in
the dynamic equations)
    phi_inertial=[m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1
m2 mx2 my2 mz2 Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3
Ixy3 Ixz3 Iyy3 Iyz3 Izz3 m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4
Izz4 m5 mx5 my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 m6 mx6 my6 mz6
Ixx6 Ixy6 Ixz6 Iyy6 Iyz6 Izz6 m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7
Iyz7 Izz7]';

    % Frictional parameters
    phi_frictional=[c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6
cc7]';

    % Vector of parameters to be estimated
    phi=[phi_inertial;phi_frictional];
```

```
    tic
    disp('Obteniendo phi1 y phi2')
    E_prima_phi=E'*phi;
    phi1=E_prima_phi(1:r,:);
    phi2=E_prima_phi(r+1:end,:);
    toc

    tic
    disp('Calculando los parámetros base(phi_b) del modelo
reducido')
    phi_b=phi1+beta*phi2;
    beta_num=beta;
    for i=1:size(beta_num,1)
        for j=1:size(beta_num,2)
            if abs(beta_num(i,j)) < 1e-4
                beta_num(i,j)=0;
            end
        end
    end
    phi_b_num=phi1+beta_num*phi2;
    toc

    Wb=W1;

    r=rank(W);
% WE=W*E;
% Wb=WE(:,1:r);
    Etphi=E'*[1:size(W,2)]';
    v=Etphi(1:r);

    save v v

else
    disp('El rango de W es igual que su número de columnas. No
es necesaria reducción de parámetros')
end
```

QR_reduction_exp.m [Reducción QR para los experimentos]

```
function [phi,phi_b_num,Wb,E,r,beta,v]=QR_reduction_exp(W)

tic
disp('Obteniendo matrices Q, R y E')
[Q,R,E]=qr(W);
toc

tic
disp('Calculando el rango de W')
r=rank(W);
disp(['El rango de W es ',num2str(r)]);
toc

%El tamaño de la matriz W es nxp
n=size(W,1);
p=size(W,2);

if r<p
    for i=r+1:p
    R(i,i)=0;
    end

    tic
    disp('Obteniendo matrices R1 y R2')
    R1=R(1:r,1:r);
    R2=R(1:r,r+1:end);
    toc

    tic
    disp('Calculando beta')
    beta=inv(R1)*R2;
    toc

    tic
    disp('Obteniendo matrices W1 y W2')
    W1=W*E(:,1:r);
    w2=W*E(:,r+1:end);
    toc

    tic
    disp('Obteniendo matrices Q1 y Q2')
    Q1=Q(:,1:r);
    Q2=Q(:,r+1:end);

    %  symbolic definition of the inercial parameters
%     syms m_Base mx_Base my_Base mz_Base Ixx_Base Ixy_Base
Ixz_Base Iyy_Base Iyz_Base Izz_Base real
%     syms m_Sol1 mx_Sol1 my_Sol1 mz_Sol1 Ixx_Sol1 Ixy_Sol1
Ixz_Sol1 Iyy_Sol1 Iyz_Sol1 Izz_Sol1 real
%     syms m_Sol2 mx_Sol2 my_Sol2 mz_Sol2 Ixx_Sol2 Ixy_Sol2
Ixz_Sol2 Iyy_Sol2 Iyz_Sol2 Izz_Sol2 real
```

```
%     syms m_Sol3 mx_Sol3 my_Sol3 mz_Sol3 Ixx_Sol3 Ixy_Sol3
Ixz_Sol3 Iyy_Sol3 Iyz_Sol3 Izz_Sol3 real
%     syms m_Sol4 mx_Sol4 my_Sol4 mz_Sol4 Ixx_Sol4 Ixy_Sol4
Ixz_Sol4 Iyy_Sol4 Iyz_Sol4 Izz_Sol4 real
%     syms m_Sol5 mx_Sol5 my_Sol5 mz_Sol5 Ixx_Sol5 Ixy_Sol5
Ixz_Sol5 Iyy_Sol5 Iyz_Sol5 Izz_Sol5 real
%     syms m_Sol6 mx_Sol6 my_Sol6 mz_Sol6 Ixx_Sol6 Ixy_Sol6
Ixz_Sol6 Iyy_Sol6 Iyz_Sol6 Izz_Sol6 real
%     syms m_Sol7 mx_Sol7 my_Sol7 mz_Sol7 Ixx_Sol7 Ixy_Sol7
Ixz_Sol7 Iyy_Sol7 Iyz_Sol7 Izz_Sol7 real
syms m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1 m2 mx2 my2 mz2
Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3 Ixy3 Ixz3 Iyy3
Iyz3 Izz3 m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4 Izz4 m5 mx5
my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 m6 mx6 my6 mz6 Ixx6 Ixy6
Ixz6 Iyy6 Iyz6 Izz6 m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7 Iyz7 Izz7
real


    %  symbolic definition of the friction parameters
%     syms c_Base_Sol1 c_Sol1_Sol2 c_Sol2_Sol3 c_Sol3_Sol4
c_Sol4_Sol5 c_Sol5_Sol6 c_Sol6_Sol7 real
syms c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6 cc7 real

    %  symbolic definition of the geometric parameters
%     syms r1_base r2_base r_in_base height_base r_out_sol1
r_in_sol1 height_sol1 length_sol1 r_out_sol2 r_in_sol2
height_sol2 length_sol2 r_out_sol3 r_in_sol3 height_sol3
length_sol3 r_out_sol4 r_in_sol4 height_sol4 length_sol4
r_out_sol5 r_in_sol5 height_sol5 length_sol5 r_out_sol6
r_in_sol6 length_sol6 r_in_sol7 r1_out_sol7 r2_out_sol7
height_sol7 real
syms rb1 rb2 rib hb ro1 ri1 h1 l1 ro2 ri2 h2 l2 ro3 ri3 h3 l3
ro4 ri4 h4 l4 ro5 ri5 h5 l5 ro6 ri6 l6 ri7 r1o7 r2o7 h7 g real



    % Inertial parameters vector (only the ones that appear in
the dynamic equations)
    phi_inertial=[m1 mx1 my1 mz1 Ixx1 Ixy1 Ixz1 Iyy1 Iyz1 Izz1
m2 mx2 my2 mz2 Ixx2 Ixy2 Ixz2 Iyy2 Iyz2 Izz2 m3 mx3 my3 mz3 Ixx3
Ixy3 Ixz3 Iyy3 Iyz3 Izz3 m4 mx4 my4 mz4 Ixx4 Ixy4 Ixz4 Iyy4 Iyz4
Izz4 m5 mx5 my5 mz5 Ixx5 Ixy5 Ixz5 Iyy5 Iyz5 Izz5 m6 mx6 my6 mz6
Ixx6 Ixy6 Ixz6 Iyy6 Iyz6 Izz6 m7 mx7 my7 mz7 Ixx7 Ixy7 Ixz7 Iyy7
Iyz7 Izz7]';

    % Frictional parameters
    phi_frictional=[c1 c2 c3 c4 c5 c6 c7 cc1 cc2 cc3 cc4 cc5 cc6
cc7]';

    % Vector of parameters to be estimated
    phi=[phi_inertial;phi_frictional];
```

```
    tic
    disp('Obteniendo phi1 y phi2')
    E_prima_phi=E'*phi;
    phi1=E_prima_phi(1:r,:);
    phi2=E_prima_phi(r+1:end,:);
    toc

    tic
    disp('Calculando los parámetros base(phi_b) del modelo
reducido')
    phi_b=phi1+beta*phi2;
    beta_num=beta;
    for i=1:size(beta_num,1)
        for j=1:size(beta_num,2)
            if abs(beta_num(i,j)) < 1e-4
                beta_num(i,j)=0;
            end
        end
    end
    phi_b_num=phi1+beta_num*phi2;
    toc

    Wb=W1;

    r=rank(W);
% WE=W*E;
% Wb=WE(:,1:r);
    Etphi=E'*[1:size(W,2)]';
    v=Etphi(1:r);

    save v v

else
    disp('El rango de W es igual que su número de columnas. No
es necesaria reducción de parámetros')
end
```

main_symbolic_trajectory.m [Parametrización de trayectorias simbólicas]

```
clear all

% the parameters a_ji and b_ji correspond to the ith dof and jth
harmonic

% time
syms t real

% fundamental frequency in Hz
syms f real


% number of degrees of freedom
% number of harmonics
[n_dof,n_h]=get_dof_and_harm;

% create symbolic variables
for i=1:n_dof
    s=['syms a0',num2str(i),' real'];
    eval(s), clear s;
    for j=1:n_h
        s=['syms a',num2str(j),num2str(i),'
b',num2str(j),num2str(i),' real'];
        eval(s),clear s;
    end
end

% create parameters vector
param_traj=[];
for i=1:n_dof
    s=['param_traj=[param_traj;a0',num2str(i),'];'];
    eval(s),clear s;
    for j=1:n_h

s=['param_traj=[param_traj;a',num2str(j),num2str(i),';b',num2str
(j),num2str(i),'];'];
        eval(s),clear s;
    end
end

% create expression of qi
for i=1:n_dof
    s=['q',num2str(i),'=a0',num2str(i),';'];
    eval(s),clear s;
    for j=1:n_h

s=['q',num2str(i),'=q',num2str(i),'+a',num2str(j),num2str(i),'*c
os(2*pi*f*j*t)+b',num2str(j),num2str(i),'*sin(2*pi*f*j*t);'];
        s,
        eval(s),clear s;
    end
end
```

```
% gather the qi expressions in a single vector
q=[];
for i=1:n_dof
    s=['q=[q;q',num2str(i),'];'];
    eval(s),clear s;
end

dq=diff(q,t);
ddq=diff(dq,t);

matlabFunction(q,dq,ddq,'File','evalTrajectory','vars',{t,f,para
m_traj});

% q
% param_traj

qMAT=jacobian(q,param_traj); %Se obtiene todo lo que multiplica
a los parámetros de trayectoria (dentro de las coordenadas de
trayectoria). Matriz A en la docu de MATLAB. Matriz R en los
apuntes
dqMAT=jacobian(dq,param_traj); %Se obtiene todo lo que
multiplica a los parámetros de trayectoria(dentro de las
velocidades de trayectoria). Matriz A en la docu de MATLAB.
Matriz R en los apuntes
ddqMAT=jacobian(ddq,param_traj);

matlabFunction(qMAT,'File','evalqMAT','vars',{t,f});
matlabFunction(dqMAT,'File','evaldqMAT','vars',{t,f});
matlabFunction(ddqMAT,'File','evalddqMAT','vars',{t,f});
```

main_optimize_trajectory.m [Optimización de trayectorias]

```
% main_optimize_trajectory
clear all

% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% number of points
n=100;

% fundamental frequency
f=0.1;

% crit='cond';
crit='dopt';

if crit=='cond'
    load best_cond.mat;
    best_value=best_cond;
elseif crit=='dopt'
    load best_dopt.mat;
    best_value=best_dopt;
end

% time instants
times=0:(1/f)/(n-1):(1/f);

% security factor
sf= 0.95;

qmax=   sf*[170;120;170;120;170;120;175]*pi/180;
qmin= - qmax;
dqmax=  sf*[2267;2267;2267;2000;2167;3600;3600]*2*pi/60/160;

for i=1:length(times)
    Aqi=evalqMAT(times(i),f);
    Adqi=evaldqMAT(times(i),f);
    A(4*n_dof*(i-1)+1:4*n_dof*i,:)=[Aqi;-Aqi;Adqi;-Adqi];
    b(4*n_dof*(i-1)+1:4*n_dof*i,1)=[qmax;-qmin;dqmax;dqmax];
end

% size(A)
% size(b)
% pause
```

```
cont=0;
while true %cont<10
    cont=cont+1;
    rng('shuffle');
    param_traj_0=2*pi*rand((2*n_h+1)*n_dof,1);

%
options=optimoptions(@fmincon,'Display','iter','MaxFunctionEvalu
ations',1000*((2*n_h+1)*n_dof));

options=optimoptions(@fmincon,'Display','iter','MaxFunctionEvalu
ations',Inf,'maxiter',1000);

    fun=@(param_traj)objective_function(param_traj,n,f,crit);
%
[param_traj_opt,value]=fminunc(@(param_traj)objective_function(p
aram_traj,n,f,crit),param_traj_0,options);
%     [param_traj_opt,value]=fminunc(fun,param_traj_0,options);

[param_traj_opt,value]=fmincon(fun,param_traj_0,A,b,[],[],[],[],
[],options);
%     x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)

        value_str=num2str(value);
        cont_str=num2str(cont);

        if crit=='cond'
                s=['param_cond_',value_str,'.mat'];
        elseif crit=='dopt'
                s=['param_dopt_',value_str,'.mat'];
        else
                s=['param_crit_',value_str,'.mat'];
        end

        command_str=['save ',s,' param_traj_opt;'];
        eval(command_str);

        if value<best_value
        best_value=value;
        if crit=='cond'
            best_value_cond=best_value;
            param_traj_opt_cond=param_traj_opt;
            save param_traj_opt_cond param_traj_opt_cond;
            save best_value_cond best_value_ cond;
        elseif crit=='dopt'
            best_value_dopt=best_value;
            param_traj_opt_dopt=param_traj_opt;
            save param_traj_opt_dopt param_traj_opt_dopt;
            save best_value_dopt best_value_dopt;
        end
    end

end
param_traj_opt
```

```
if crit=='cond'
    q_opt=A*param_traj_opt_cond;
elseif crit=='dopt'
    q_opt=A*param_traj_opt_dopt;
end

for i=1:7,Q(:,i)=q_opt(i:7*4:end);end
for i=1:7,DQ(:,i)=q_opt(i+7*2:7*4:end);end

figure
    for i=1:7
        ax(i)=subplot(2,4,i);
        plot(times,Q(:,i)*180/pi)
        hold on
        plot([0,1/f],[-qmax(i),-qmax(i)]*180/pi,'r')
        plot([0,1/f],[qmax(i),qmax(i)]*180/pi,'r')
    end
linkaxes([ax(1),ax(2),ax(3),ax(4),ax(5),ax(6),ax(7)],'y')
sgtitle('Positions in degrees')

figure
for i=1:7
    ax(i)=subplot(2,4,i);
    plot(times,DQ(:,i)*60*160/2/pi)
    hold on
    plot([0,1/f],[-dqmax(i),-dqmax(i)]*60*160/2/pi,'r')
    plot([0,1/f],[dqmax(i),dqmax(i)]*60*160/2/pi,'r')
end
linkaxes([ax(1),ax(2),ax(3),ax(4),ax(5),ax(6),ax(7)],'y')
sgtitle('Velocities in rpm')
```

get_W_with_param_traj.m [Ensamblar la matriz de observación en base a unos parámetros de trayectoria óptimos]

```
function W=get_W_with_param_traj(param_traj,n,f,crit)

% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% get geometric parameters
param_geom=get_param_geom;

% time instants
times=0:(1/f)/(n-1):(1/f);

tic
disp('Comienza el ensamblado de la matriz W para la trayectoria
que hemos obtenido por parametrización')
for i=1:length(times)
    tic
    disp(['Ensamblando fila ',num2str(i),' de la matriz W']);
    % get q, dq, and ddq for each time instant
    [q,dq,ddq]=evalTrajectory(times(i),f,param_traj);
    s_dq=sign(dq);

    % evaluate the observation matrix for each time instant
    Ki=evalK(q,dq,ddq,s_dq,param_geom);

    % assemble the whole observation matrix
    W(n_dof*(i-1)+1:n_dof*i,:)=Ki;
    toc
end
toc
```

get_W_with_param_traj.m [Ejecuta de forma consecutiva Reducción, Parametrización y Optimización]

```
clear all
close all
clc

% Parametrization of the trajectories we'll use for the
optimization
main_symbolic_trajectory

% Reduction of the original model. Obtaining the "v" columns we
need to
% achieve the base parameters
traj_and_phi_b_obtaining

% Optimization of the trajectories. We'll use them to identify
the
% parameters
% main_optimize_trajectory
main_optimize_trajectory_TEMPORAL
```

main_symbolic_bezier.m [Parametriza las trayectorias de enlace con curvas de BEZIER]

```
%function main_symbolic_bezier
clear all,close all,

% variable tiempo
syms t real

% variables de la pose a la que se quiere llegar (pose inicial
de los armÃ³nicos)
syms q dq ddq real

% tiempo que dura la trayectoria de bezier
syms Tb real

% parÃ¡metros de la trayectoria de bezier. El resto son iguales
a cero. (b0=b1=b2=0)
syms b3 b4 b5 real

% pose a la que llegar
pose=[q;dq;ddq];

% parÃ¡metros de la trayectoria de bezier (orden 5 (6
parÃ¡metros))
b=[0;0;0;b3;b4;b5];

% cÃ¡lculo de los parÃ¡metros de la derivada de la trayectoria
for i=1:length(b)-1
    bp(i)=(b(i+1)-b(i))*(length(b)-1)/Tb;
end

% cÃ¡lculo de los parÃ¡metros de la segunda derivada de la
trayectoria
for i=1:length(bp)-1
    bpp(i)=(bp(i+1)-bp(i))*(length(bp)-1)/Tb;
end

% expresiones explÃ-citas de la trayectoria y sus dos primeras
derivadas en funciÃ³n de los parÃ¡metros b3, b4 y b5

%     bt = b(1)*(1-t)^5   + b(2)*5*(1-t)^4*t   + b(3)*10*(1-
t)^3*t^2   + b(4)*10*(1-t)^2*t^3  + b(5)*5*(1-t)*t^4 + b(6)*t^5;
%     bpt= bp(1)*(1-t)^4  + bp(2)*4*(1-t)^3*t  + bp(3)*6*(1-
t)^2*t^2   + bp(4)*4*(1-t)^1*t^3  + bp(5)*t^4;
%     bppt=bpp(1)*(1-t)^3 + bpp(2)*3*(1-t)^2*t + bpp(3)*3*(1-
t)^1*t^2  + bpp(4)*t^3;
```

```
    % bt
    bt=0;
    for i=1:length(b)
        n=length(b)-1;
        bt=bt+(b(i)*(factorial(n)/(factorial(i-1)*factorial(n-
(i-1))))*(1-t)^(n-(i-1))*t^(i-1));
    end

    % bpt
    bpt=0;
    for i=1:length(b)-1
        n=length(b)-2;
        bpt=bpt+(bp(i)*(factorial(n)/(factorial(i-
1)*factorial(n-(i-1))))*(1-t)^(n-(i-1))*t^(i-1));
    end

    % bppt
    bppt=0;
    for i=1:length(b)-2
        n=length(b)-3;
        bppt=bppt+(bpp(i)*(factorial(n)/(factorial(i-
1)*factorial(n-(i-1))))*(1-t)^(n-(i-1))*t^(i-1));
    end

% vector de los parÃ¡metros incÃ³gnita del problema
param_bezier=[b3;b4;b5];

% vector con las tres expresiones simbÃ³licas de posiciÃ³n,
velocidad y aceleraciÃ³n
b_bp_bpp=[bt;bpt;bppt];

% evaluaciÃ³n de las tres expresiones al final de la trayectoria
(t=1)
b_bp_bpp=subs(b_bp_bpp,t,1);

% como el problema es lineal en los parÃ¡metros b3, b4 y b5,
calculo el jacobiano para resolver linealmente
B=jacobian(b_bp_bpp,param_bezier);

% resoluciÃ³n del problema lineal: pose=B*param_sol
param_sol=B\pose;
param_sol

% creo una funciÃ³n que en funciÃ³n del tiempo de trayectoria
(Tb) y la pose a la que hay que llegar (pose) me devuelva los
parÃ¡metros de trayectoria b3, b4 y b5.
matlabFunction(param_sol,'File','eval_param_sol','vars',{Tb,pose
})
% creo las funciones que me dan la trayectoria y sus dos
primeras derivadas para cualquier instante de tiempo.
matlabFunction(bt  ,'File','evalbt','vars',{t,Tb,param_bezier})
matlabFunction(bpt ,'File','evalbpt','vars',{t,Tb,param_bezier})
matlabFunction(bppt,'File','evalbppt','vars',{t,Tb,param_bezier}
)
```

main_numeric_bezier.m [Calcula y representa las trayectorias de enlace (curvas de BEZIER)]

```
clear all
close all

% cargo los parÃ¡metros de la trayectoria ArmÃ³nica a la que
queremos acoplar una trayectoria de Bezier
% load ./optimum_aldakin/param_cond_1_16.8178.mat
% load ./optimum_aldakin/param_cond_2_17.1024.mat
% load ./optimum_aldakin/param_cond_3_20.8017.mat
% load ./optimum_aldakin/param_dopt_4_639.6953.mat
% load ./optimum_aldakin/param_dopt_5_639.9958.mat
% load ./optimum_aldakin/param_dopt_6_639.9697.mat
load ./optimum_aldakin/param_dopt_1_744.4401.mat

% cambio de nombre a la variable para evidenciar que son los
parÃ¡metros de la tray. armÃ³nica
param_harmonic=param_traj_opt;

% tiempo de la trayectoria armÃ³nica (la h viene de "harmonics"
en inglÃ©s)
Th=10;

% tiempo que dura la trayectoria de Bezier
Tb=5;

% nÃºmero de armÃ³nicos de las trayectorias armÃ³nicas
n_h=5;

% nÃºmero de gdl de las trayectorias armÃ³nicas
n_dof=7;

% nÃºmero de repeticiones de la trayectoria armÃ³nica que se
repetirÃ¡n tras la de Bezier
nloops=1;

% instantes de tiempo de la trayectoria de Bezier
t=0:0.01:Tb-0.01;

% instantes de tiempo de la trayectoria de armÃ³nicos
time_h=0:0.01:Th-0.01;

% las operaciones se repiten para cada una de las 7 trayectorias
(7 gdl)
% las funciones trabajan sobre cada una individualmente
```

```
for i=1:n_dof
    % meto en param_t los 11=(2*nh+1) parÃ¡metros de la
trayectoria i-Ã©sima.
    param_t=param_harmonic((2*n_h+1)*(i-1)+1:(2*n_h+1)*i,1);
    % evaluo la trayectoria para t=0. Es decir, obtengo la pose
a la que hay que llegar
    [q,dq,ddq]=evalTrajectory(0,1/Th,param_t);
    % meto la pose en el vector pose
    pose=[q;dq;ddq];
    % calculo los parÃ¡metros de Bezier que llegarÃ¡n a la pose
si la trayectoria de Bezier dura Tb
    % y parte de posiciÃ³n 0, velocidad 0 y aceleraciÃ³n 0
    param_b=eval_param_sol(Tb,pose);
    % para cada instante de tiempo de la trayectoria de bezier,
calculo el valor de Q, DQ y DDQ
    for j=1:length(t)
        Q(i,j)=evalbt(t(j)/Tb,Tb,param_b);
        DQ(i,j)=evalbpt(t(j)/Tb,Tb,param_b);
        DDQ(i,j)=evalbppt(t(j)/Tb,Tb,param_b);
    end

    % para cada instante de tiempo de las nloops repeticiones de
la trayectoria de armÃ³nicos,
    % calculo Q, DQ y DDQ
    for k=1:nloops
        for j=1:length(time_h)
            [Q(i,j+length(t)+(k-1)*length(time_h)),...
             DQ(i,j+length(t)+(k-1)*length(time_h)),...
             DDQ(i,j+length(t)+(k-
1)*length(time_h))]=evalTrajectory(time_h(j),1/Th,param_t);
        end
    end
end

% creo un vector de tiempos que una los tiempos de la
trayectoria de Bezier y los tiempos de
% las nloops repeticiones de la trayectoria de armÃ³nicos
time=0:0.01:Th*nloops+Tb-0.01;

% represento las trayectorias y sus derivadas y pinto una barra
vertical en los puntos
% en los que se empalman las trayectorias.
figure,
subplot(3,1,1),
plot(time,Q)
hold on
for i=1:nloops
    plot([Tb+(i-1)*Th,Tb+(i-1)*Th],[-pi,pi],'k')
end
```

```
axis([0,Th*nloops+Tb,-2*pi,2*pi])
title('Trayectorias (en posición) de Bezier [0,Tb] y armónica
[Tb,(Tb+Th)]')
xlabel('Tiempo [s]')
ylabel('Posición de cada GDL [rad]')
legend('AX1','AX2','AX3','AX4','AX5','AX6','AX7')

subplot(3,1,2)
plot(time,DQ)
hold on
for i=1:nloops
    plot([Tb+(i-1)*Th,Tb+(i-1)*Th],[-pi,pi],'k')
end
axis([0,Th*nloops+Tb,-2*pi,2*pi])
title('Trayectorias (en velocidad) de Bezier [0,Tb] y armónica
[Tb,(Tb+Th)]')
xlabel('Tiempo [s]')
ylabel('Velocidad de cada GDL [rad/s]')
legend('AX1','AX2','AX3','AX4','AX5','AX6','AX7')

subplot(3,1,3)
plot(time,DDQ)
hold on
for i=1:nloops
    plot([Tb+(i-1)*Th,Tb+(i-1)*Th],[-pi,pi],'k')
end
axis([0,Th*nloops+Tb,-2*pi,2*pi])
title('Trayectorias (en aceleración) de Bezier [0,Tb] y armónica
[Tb,(Tb+Th)]')
xlabel('Tiempo [s]')
ylabel('Aceleración de cada GDL [rad/(s^2)]')
legend('AX1','AX2','AX3','AX4','AX5','AX6','AX7')
```

experiments.m [Simula la realización de los experimentos, representa y guarda resultados]

```
%% LIMPIAR ESPACIO DE TRABAJO

clear all
close all
clc

%% TRAYECTORIA
% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% number of points
n=100;

% fundamental frequency
f=0.1;

% crit='cond';
crit='dopt';

load param_cond_3_51.8841.mat

%% ENSAMBLAR MATRIZ DE OBSERVACIÓN
W=get_W_with_param_traj(param_traj_opt,n,f,crit);

save W W

%% OBTENER MODELO REDUCIDO
[phi,phi_b,Wb,E,r,beta]=QR_reduction_exp(W);
phi_b;
phi_b_definitivo=vpa(phi_b,4)

%% OBTENER VALORES DE LOS PARÁMETROS SIMILARES A LOS DEL ROBOT
REAL
load param_s_exp.mat
load param_n_exp.mat

syms Iyx_Sol1 Iyx_Sol2 Iyx_Sol3 Iyx_Sol4 Iyx_Sol5 Iyx_Sol6
Iyx_Sol7 Izx_Sol1 Izx_Sol2 Izx_Sol3 Izx_Sol4 Izx_Sol5 Izx_Sol6
Izx_Sol7 Izy_Sol1 Izy_Sol2 Izy_Sol3 Izy_Sol4 Izy_Sol5 Izy_Sol6
Izy_Sol7 real

sym_duplicated_var=[Iyx_Sol1 Iyx_Sol2 Iyx_Sol3 Iyx_Sol4 Iyx_Sol5
Iyx_Sol6 Iyx_Sol7 Izx_Sol1 Izx_Sol2 Izx_Sol3 Izx_Sol4 Izx_Sol5
Izx_Sol6 Izx_Sol7 Izy_Sol1 Izy_Sol2 Izy_Sol3 Izy_Sol4 Izy_Sol5
Izy_Sol6 Izy_Sol7]

new_param_s_exp=param_s_exp;
new_param_n_exp=param_n_exp;
```

```
cont=0;
for i=1:length(param_s_exp)
    for j=1:length(sym_duplicated_var)
        if param_s_exp(i)==sym_duplicated_var(j)
            cont=cont+1;
            index_duplicated(cont)=i;
        end
    end
end

new_param_s_exp(index_duplicated)=[];
new_param_n_exp(index_duplicated)=[];

param_order=[1,2,3,4,5,8,9,6,10,7,11,12,13,14,15,18,19,16,20,17,
21,22,23,24,25,28,29,26,30,27,31,32,33,34,35,38,39,36,40,37,41,4
2,43,44,45,48,49,46,50,47,51,52,53,54,55,58,59,56,60,57,61,62,63
,64,65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84]
;

for i=1:length(phi)
    phi_real(i,1)=new_param_n_exp(param_order(i));
end

save phi_real phi_real

%% CALCULAR LOS PARES ASOCIADOS A ESA TRAYECTORIA CON LOS
PARÁMETROS TEÓRICAMENTE REALES

tau_real=W*phi_real;

%% OBTENER LA DESVIACIÓN TÍPICA DE LOS PARES MEDIDOS (5-10% DE
AMP MAX

for i=1:7
    max_torque=max(tau_real)
    min_torque=min(tau_real)
end

if abs(max_torque)>abs(min_torque)
    sigma=0.05*abs(max_torque);
else
    sigma=0.05*abs(min_torque);
end

%% OBTENER VALORES DE PARES MEDIDOS (RANDOMIZANDO LOS REALES)

tau_m=tau_real+sigma*randn(size(tau_real));

%% COMPARAR PARES REALES CON LOS MEDIDOS (COMO SI HUBIÉRAMOS
HECHO EXPERIMENTOS)
```

```
figure
for i=1:7
    cont=0;
    for j=i:7:length(tau_real)
        cont=cont+1;
        tau_ax_real(cont)=tau_real(j);
        tau_ax_m(cont)=tau_m(j);
    end
    ax_real(:,i)=tau_ax_real';
    ax_m(:,i)=tau_ax_m';
end

for i=1:7
    ax(i)=subplot(2,4,i);
    plot(ax_real(:,i))
    hold on
    plot(ax_m(:,i))
    xlabel('TIME [s]')
    ylabel('TORQUE [Nm]')
    legend('\tau_{real}','\tau_{medida}')
    title(['REAL VS. MEASURED \tau AX ',num2str(i)])
end

%% CALCULAR LA MATRIZ DE OBSERVACIÓN DEL MODELO REDUCIDO

load v.mat

Wbase=W(:,v);

%% REALIZAR ESTIMACIÓN DE LOS PARÁMETROS

phi_base_est=Wbase\tau_m;
% phi_base_est=inv(Wbase'*Wbase)*Wbase'*tau_m;

E_prima_phi_real=E'*phi_real;
phi_1_real=E_prima_phi_real(1:r,:);
phi_2_real=E_prima_phi_real(r+1:end,:);

phi_base_real = phi_1_real + beta * phi_2_real;

save phi_base_est phi_base_est

%% CALCULAR MATRIZ DE VARIANZAS Y COVARIANZAS EN ESTIMACIÓN DE
PARÁMETROS

VAR_phi_base_est = sigma^2 *inv(Wbase'*Wbase);
sigma_phi_base_est= sqrt(diag(VAR_phi_base_est));
```

```
%% REPRESENTAR PARÁMETROS ESTIMADOS DENTRO DE RANGOS ACEPTABLES
(VER VALIDEZ)

x=1:length(phi_base_est);
for i=1:length(phi_base_est)
    err_neg(i)=phi_base_real(i) - 3*sigma_phi_base_est(i);
    err_pos(i)=phi_base_real(i) + 3*sigma_phi_base_est(i);
end

figure
errorbar(x,phi_base_est,err_neg,err_pos,'o','markersize',5)
xlabel('Base parameters')
ylabel('Value')
title('VALIDATION OF ESTIMATED PARAMETERS (ACCEPTABLE RANGES)')

%% ESTIMAR LOS PARES

tau_est=Wbase*phi_base_est;
save tau_est tau_est

%% CÁLCULOS ASOCIADOS A LOS RESIDUOS

% RESIDUO
epsilon=tau_real-tau_est;

% MEDIA
media_epsilon=mean(epsilon);
sigma_epsilon_est=std(epsilon);

% VARIANZA
m=size(tau_real,1);
var_epsilon_est=sigma_epsilon_est^2;

% HISTOGRAMA EPSILON
figure
histfit(epsilon)
title('Histograma de los residuos epsilon')

%% CALCULAR IC AL 95% PARA LOS PARES

z_975=norminv(0.975,0,1); %norminv me da el valor que da 0.5
prob por debajo para una N de mu, sigma^2

var_tau_est=zeros(m,1); % Asi no aumentamos el tamaño por bucle
(pesado), vamos rellenando esta.

for i=1:m
    var_tau_est(i,1)=Wbase(i,:)*VAR_phi_base_est*(Wbase(i,:))';
end

for i=1:m
    IC_min_tau(i,1)=tau_est(i,1)-z_975*sqrt(var_tau_est(i,1));
    IC_max_tau(i,1)=tau_est(i,1)+z_975*sqrt(var_tau_est(i,1));
end
```

```
%% GRAFICAR PARES MEDIDOS, ESTIMADOS Y SUS INTERVALOS DE
CONFIANZA

figure
for i=1:7
    cont=0;
    for j=i:7:length(tau_real)
        cont=cont+1;
        tau_ax_est(cont)=tau_est(j);
        IC_ax_min(cont)=IC_min_tau(j);
        IC_ax_max(cont)=IC_max_tau(j);
    end
    ax_est(:,i)=tau_ax_est';
    ax_IC_min(:,i)=IC_ax_min';
    ax_IC_max(:,i)=IC_ax_max';

end

for i=1:7
    ax(i)=subplot(2,4,i);
    plot(ax_m(:,i),'b.')
    hold on
    plot(ax_est(:,i),'m')
    plot(ax_IC_min(:,i),'g')
    plot(ax_IC_max(:,i),'k')
    xlabel('TIME [s]')
    ylabel('TORQUE [Nm]')

legend('\tau_{medida}','\tau_{estimada}','IC_{min}','IC_{max}')
    title(['MEDIDO VS. ESTIMATED \tau AX ',num2str(i)])
end
```

validation.m [Realiza la validación, representa y guarda resultados]

```
%% LIMPIAR ESPACIO DE TRABAJO

clear all
close all
clc

%% TRAYECTORIA
% number of dofs and harmonics
[n_dof,n_h]=get_dof_and_harm;

% number of points
n=100;

% fundamental frequency
f=0.1;

% crit='cond';
crit='dopt';

load param_dopt_12_742.0422.mat
% load param_dopt_10_744.4401.mat

%% ENSAMBLAR MATRIZ DE OBSERVACIÓN
W=get_W_with_param_traj(param_traj_opt,n,f,crit);

save W W

%% OBTENER MODELO REDUCIDO
[phi,phi_b,Wb,E,r,beta]=QR_reduction_exp(W);

%% OBTENER VALORES DE LOS PARÁMETROS SIMILARES A LOS DEL ROBOT
REAL
load phi_real.mat

%% CALCULAR LOS PARES ASOCIADOS A ESA TRAYECTORIA CON LOS
PARÁMETROS TEÓRICAMENTE REALES
tau_real=W*phi_real;

%% OBTENER LA DESVIACIÓN TÍPICA DE LOS PARES MEDIDOS (5-10% DE
AMP MAX
for i=1:7
    max_torque=max(tau_real)
    min_torque=min(tau_real)
end

if abs(max_torque)>abs(min_torque)
    sigma=0.05*abs(max_torque);
else
    sigma=0.05*abs(min_torque);
end
```

```
%% OBTENER VALORES DE PARES MEDIDOS (RANDOMIZANDO LOS REALES)
tau_m=tau_real+sigma*randn(size(tau_real));

%% OBTENER PARÁMETROS ESTIMADOS EN EXPERIMENTOS
load phi_base_est.mat

%% COMPARAR PARES REALES CON LOS MEDIDOS (COMO SI HUBIÉRAMOS
HECHO EXPERIMENTOS)
figure
for i=1:7
    cont=0;
    for j=i:7:length(tau_real)
        cont=cont+1;
        tau_ax_real(cont)=tau_real(j);
        tau_ax_m(cont)=tau_m(j);
    end
    ax_real(:,i)=tau_ax_real';
    ax_m(:,i)=tau_ax_m';
end

for i=1:7
    ax(i)=subplot(2,4,i);
    plot(ax_real(:,i))
    hold on
    plot(ax_m(:,i))
    xlabel('TIME [s]')
    ylabel('TORQUE [Nm]')
    legend('\tau_{real}','\tau_{medida}')
    title(['REAL VS. MEASURED \tau AX ',num2str(i)])
end

%% CALCULAR LA MATRIZ DE OBSERVACIÓN DEL MODELO REDUCIDO
load v.mat

Wbase=W(:,v);

%% REALIZAR ESTIMACIÓN DE LOS PARÁMETROS

phi_base_est2=Wbase\tau_m;
load tau_est.mat
tau_est_traj1=tau_est;

E_prima_phi_real=E'*phi_real;
phi_1_real=E_prima_phi_real(1:r,:);
phi_2_real=E_prima_phi_real(r+1:end,:);

phi_base_real = phi_1_real + beta * phi_2_real;

%% CALCULAR MATRIZ DE VARIANZAS Y COVARIANZAS EN ESTIMACIÓN DE
PARÁMETROS

VAR_phi_base_est = sigma^2 *inv(Wbase'*Wbase);
sigma_phi_base_est= sqrt(diag(VAR_phi_base_est));
```

```
%% REPRESENTAR PARÁMETROS ESTIMADOS DENTRO DE RANGOS ACEPTABLES
(VER VALIDEZ)

x=1:length(phi_base_est2);
for i=1:length(phi_base_est2)
    err_neg(i)=phi_base_real(i) - 3*sigma_phi_base_est(i);
    err_pos(i)=phi_base_real(i) + 3*sigma_phi_base_est(i);
end

figure
errorbar(x,phi_base_est,err_neg,err_pos,'o','markersize',5)
xlabel('Base parameters')
ylabel('Value')
title('VALIDATION OF ESTIMATED PARAMETERS 2nd TRAJ (ACCEPTABLE
RANGES)')

%% COMPARAR PARÁMETROS ESTIMADOS EN EXPERIMENTOS Y EN VALIDACIÓN

ind=[9 8 10 19 1 13 2 23 3 17 38 4 20 12 24 22 29 7 34 5 6 31 27
26 25 18 21 15 49 33 11 14 28 32 47 16 44 37 48 36 44 35 46 53
51 55 56 52 45 57 39 40 42 41 43 54 50]

for i=1:length(phi_base_est)
    phi_base_est1(i)=phi_base_est(ind(i))
end


figure
plot(phi_base_est1,'b.-','markersize',10)
hold on
plot(phi_base_est2,'m.-','markersize',10)
xlabel('Base parameters')
ylabel('Value')
legend('\phi_{est traj 1}','\phi_{est traj 2}')
title('COMPARATIVE OF ESTIMATED PARAMETERS (BOTH TRAJECTORIES')

%% ESTIMAR LOS PARES

tau_est_traj2=Wbase*phi_base_est2;

%% CÁLCULOS ASOCIADOS A LOS RESIDUOS

% RESIDUO
epsilon2=tau_real-tau_est_traj2;


% MEDIA
sigma_epsilon_est2=std(epsilon2);

% VARIANZA
m=size(tau_real,1);
var_epsilon_est2=sigma_epsilon_est2^2;
```

```
% HISTOGRAMA EPSILON2

figure
histfit(epsilon2)
title('Histograma de los residuos epsilon con las estimaciones
de trayectoria 2')

%% CALCULAR IC AL 95% PARA LOS PARES

z_975=norminv(0.975,0,1); %norminv me da el valor que da 0.5
prob por debajo para una N de mu, sigma^2

var_tau_est=zeros(m,1); % Asi no aumentamos el tamaño por bucle
(pesado), vamos rellenando esta.

for i=1:m
    var_tau_est(i,1)=Wbase(i,:)*VAR_phi_base_est*(Wbase(i,:))';
end

for i=1:m
    IC_min_tau(i,1)=tau_est_traj2(i,1)-
z_975*sqrt(var_tau_est(i,1));

IC_max_tau(i,1)=tau_est_traj2(i,1)+z_975*sqrt(var_tau_est(i,1));
end
```

```
%% GRAFICAR PARES MEDIDOS, ESTIMADOS Y SUS INTERVALOS DE
CONFIANZA

figure
for i=1:7
    cont=0;
    for j=i:7:length(tau_real)
        cont=cont+1;
        tau_ax_est1(cont)=tau_est_traj1(j);
        tau_ax_est2(cont)=tau_est_traj2(j);
        IC_ax_min(cont)=IC_min_tau(j);
        IC_ax_max(cont)=IC_max_tau(j);
    end
    ax_est1(:,i)=tau_ax_est1';
    ax_est2(:,i)=tau_ax_est2';
    ax_IC_min(:,i)=IC_ax_min';
    ax_IC_max(:,i)=IC_ax_max';

end

for i=1:7
    ax(i)=subplot(2,4,i);
    plot(ax_m(:,i),'b.')
    hold on
    plot(ax_est1(:,i),'m')
    plot(ax_est2(:,i),'r')
    plot(ax_IC_min(:,i),'g')
    plot(ax_IC_max(:,i),'k')
    xlabel('TIME [s]')
    ylabel('TORQUE [Nm]')

legend('\tau_{medida}','\tau_{estimada_1}','\tau_{estimada_2}','
IC_{min}','IC_{max}')
    title(['MEDIDO VS. ESTIMATED \tau AX ',num2str(i)])
end
```