UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIERÍA INDUSTRIAL VALENCIA

TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

MODELADO CINEMÁTICO Y DINÁMICO DE UN ROBOT PARALELO CON ARQUITECTURA 3UPS-RPU

APLICACIÓN EN LA SIMULACIÓN DE ROBOTS DE REHABILITACIÓN DE MIEMBRO INFERIOR

AUTOR: DANIEL NEDOSSEIKINE

TUTOR: DR. ÁNGEL VALERA FERNÁNDEZ

COTUTORES: DR. LEOPOLDO ARMESTO ÁNGEL
JOSÉ LUIS PULLOQUINGA ZAPATA

Curso Académico: 2020-21

Agradecimientos

A las personas que me han ayudado a realizar este Trabajo Final de Máster, sin vuestra colaboración no hubiera sido posible.

A mis amigos de toda la vida y a los que he hecho por el camino, por compartir tantos momentos juntos.

Y en especial a mis padres, por su cariño y apoyo incondicional durante todos estos años.
Я вас очень люблю.

Resumen

En los últimos años, los campos de aplicación de la robótica han crecido enormemente. Así, además de las aplicaciones industriales típicas (pick&place, soldadura por arco o por puntos, atención de máquinas), hoy en día los robots se están empleando en la medicina, en servicios de asistencia, operaciones de rescate, etc.

Con el presente Trabajo Fin de Máster se propone trabajar con el modelado y la simulación de robots paralelos. Para ello se partirá del diseño CAD de un robot paralelo de 4 grados de libertad orientado a la rehabilitación de miembros inferiores del cuerpo humano. Se trata de un robot que se está desarrollado en ámbito de un proyecto de investigación del Plan Nacional.

Dicho diseño se importará al simulador de robots CoppeliaSim. Se trata de un simulador que incorpora un entorno de desarrollo integrado y que se basa en una arquitectura de control distribuido. De esta forma, cada objeto/modelo puede ser controlado individualmente a través de un script embebido, un plugin, un nodo ROS o BlueZero, un cliente API remoto o una solución personalizada.

Con este simulador se tendrá que resolver primero el problema cinemático directo e inverso, para poder determinar la relación entre las variables de articulación activas y las variables del espacio cartesiano. Posteriormente se tendrá que resolver también el problema dinámico del robot, obteniéndose la relación entre los términos dinámicos, centrífugos, de Coriolis y gravitacionales, y las fuerzas aplicadas sobre los actuadores del robot (acciones de control).

Gracias a este robot virtual se podrán analizar diferentes controladores no lineales de posición, como son los controladores por dinámica inversa, los controladores pasivos o los controladores adaptativos. También será posible realizar la co-simulación del robot (pudiendo ejecutarse el controlador y el robot virtual en la misma máquina o en máquinas distintas) para validar las diferentes estrategias de control que se quieran utilizar.

Además, este trabajo permitirá realizar también un conjunto de aplicaciones muy interesantes. Por ejemplo, se podrá proporcionar como entradas al robot virtual los valores de las articulaciones del robot paralelo real del laboratorio. Esto permitirá ver en tiempo real en el robot simulado la evolución del robot real sin necesidad de una señal de vídeo, requiriendo por ello un menor ancho de banda en las comunicaciones. También permitirá validar las diferentes misiones que se están desarrollando con el robot paralelo, como por ejemplo el análisis de las singularidades.

Abstract

Over the last years, the robotic application fields have experienced a big growth. Thanks to this, appart from the common industrial applications (pick&place, arc or spot welding, machine tending), nowadays robots are being used in medicine, assistance services, rescue operations, etc.

The goal of this Master's Thesis is the modeling and simulation of parallel robots. The work will be based on a CAD design of a 4 degree-of-freedom parallel robot aimed at the rehabilitation of the lower limbs of the human body. The robot is being developed within the scope of a research project of the Spanish National Plan.

This design will be imported into the CoppeliaSim robot simulator. This simulator includes an integrated development environment and is based on a distributed control architecture. This way, each object/model can be controlled individually via an embedded script, a plugin, a ROS or BlueZero node, a remote API client or a custom solution.

With this simulator, the direct and inverse kinematic problem will have to be solved first, in order to determine the relationship between the active articulation variables and the variables of the cartesian space. Subsequently, the dynamic problem of the robot will also have to be solved, obtaining the relationship between the dynamic, centrifugal and Coriolis and gravitational terms and the forces applied on the robot's actuators (control actions).

Thanks to this virtual robot, different non-linear position controllers can be analyzed, such as inverse dynamics controllers, passive controllers or adaptive controllers. Furthermore, this will allow the co-simulation of the robot (running the controller and the virtual robot in the same machine or different ones) to validate the different control strategies.

In addition, this Master's Thesis will also allow to carry out a set of very interesting applications, for example, the values of the joints of the real parallel laboratory robot can be provided as inputs to the virtual robot. This way, it would be possible to see the evolution of the real robot in real time in the simulated robot without the need of a video signal, thus requiring a lower communication bandwidth. It will also allow to validate the different missions that are being developed with the parallel robot, such as the analysis of singularities.

Índice general

Índice de figuras

Índice de tablas

Parte I

MEMORIA

Capítulo 1

Introducción

La robótica se está integrando cada vez más en la vida cotidiana. Desde los primeros robots enfocados en el ámbito puramente industrial, que trabajaban de forma aislada, se ha evolucionado hasta robots que pueden coexistir en un mismo espacio que las personas e incluso interactuar con ellas. Esto ha abierto las puertas a nuevos campos de aplicación.

Junto a los robots, también han evolucionado los sistemas de computación. El acceso a una potencia de cálculo mayor ha popularizado el uso de herramientas informáticas de simulación, que permiten analizar prototipos incluso antes de proceder a su fabricación. La simulación de diseños resulta muy ventajosa, sobre todo en situaciones donde su puesta en marcha real puede resultar arriesgada.

El presente proyecto se desarrolla en el ámbito de la rehabilitación, un sector donde las características que ofrecen los robots, como la precisión y la autonomía, tienen una especial relevancia en el diseño de tratamientos más rápidos y efectivos.

Hasta el momento, la cantidad de sistemas robotizados para medicina y rehabilitación ha sido reducida, debido la dificultad de integración con el cuerpo humano. Pese a estas complicaciones, este tipo de sistemas se están popularizando gracias a la evolución de la tecnología. Actualmente se pueden encontrar prótesis microeléctricas, capaces de detectar las señales eléctricas generadas por los músculos con el fin de controlar miembros mecánicos, exoesqueletos para potenciar las capacidades del cuerpo humano y robots de rehabilitación que tienen como objetivo asistir en los tratamientos.

En el caso de la rehabilitación, el uso de robots supone gran una ventaja tanto para el paciente, que puede recibir un tratamiento más adecuado, como para el especialista, el cual puede realizar un seguimiento más detallado del proceso de recuperación gracias a los sensores integrados. Sin embargo, al tratarse de sistemas que pueden producir grandes fuerzas en un amplio rango de movimientos, es necesario garantizar que son absolutamente seguros para los usuarios. Aquí es donde la simulación previa toma un papel de gran importancia. La validación de los diseños de forma virtual agiliza su desarrollo permitiendo obtener resultados más seguros y eficientes, todo ello en un tiempo menor y con un coste económico reducido.

Este trabajo se va a realizar en torno a un robot paralelo de 4 grados de libertad, destinado a la rehabilitación de miembros inferiores del cuerpo humano. Este robot forma parte proyecto de investigación IMBiO3R desarrollado bajo el Plan Nacional del Ministerio de Economía, Industria e Innovación (Gobierno de España). En concreto se trabajará con la tercera versión de una familia de prototipos realizados con el mismo fin.

Este último modelo tiene similitudes con su predecesor (versión 2). En primer lugar, mantiene los 4 de grados de libertad, desplazamiento en el plano XZ y rotación alrededor de los ejes Y y Z. Por otro lado, presenta la misma configuración UPS (universal-prismático-esférico) para las patas exteriores y RPU (revolución-prismático-universal) para la pata central.

No obstante, en la nueva versión del robot se han introducido ciertas mejoras con el objetivo principal de añadir rigidez a la estructura. Las articulaciones esféricas, que previamente se habían realizado mediante una esfera metálica contenida en un cojinete de Nylon, se han sustituido por una solución completamente metálica, en la que los dos componentes se han separado por elementos rodantes para disminuir la fricción y las holguras. Las juntas universales también se han sustituido por unas de mayor calidad. Por otro lado, se han empleado actuadores más potentes y lineales con unos motores más silenciosos. Todas estas actualizaciones han dotado un mayor rango de movimiento al sistema y han mejorado su respuesta permitiendo la realización de un control óptimo.

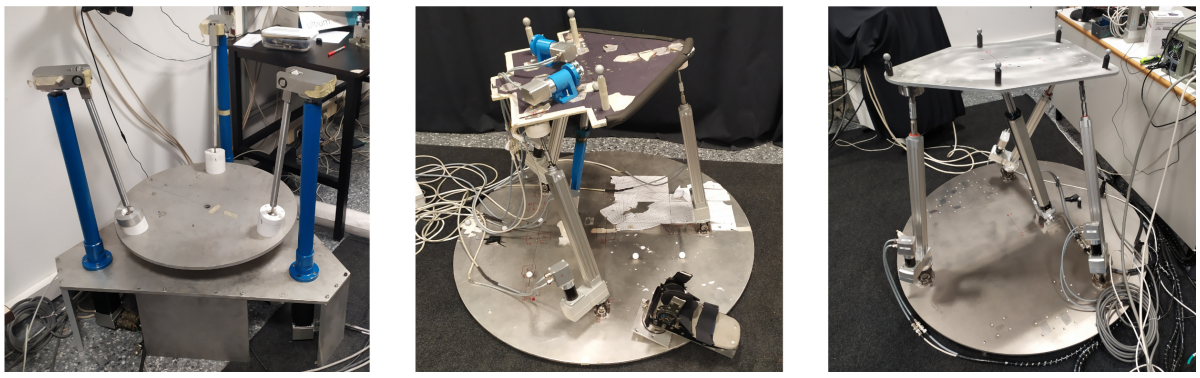


Figura 1.1: Prototipos del robot de rehabilitación: versión 1 (3 GdL), versión 2 (4 GdL) y versión 3 (4 GdL)

1.1 Objetivos

La finalidad de este trabajo es explorar métodos de modelado y simulación de robots paralelos. En concreto se plantea el uso de CoppeliaSim, un simulador universal de robots de acceso gratuito para estudiantes y uso particular. Esta plataforma se usará tanto de forma aislada como en combinación de aplicaciones externas para realizar diversos tipos de ensayos. En base a la misión principal, se propone la siguiente serie de objetivos específicos:

1. Incorporar el diseño CAD del robot al simulador CoppeliaSim. Se partirá de un ensamblaje completo del robot, al que se añadirán las articulaciones necesarias para permitir el movimiento relativo entre los distintos componentes.

2. Desarrollar un modelo cinemático en CoppeliaSim que pueda resolver los problemas cinemáticos directo e inverso. De forma adicional se incorporará un programa para permitir la selección del modo de operación deseado de una forma sencilla.

3. Crear un modelo dinámico que tendrá en cuenta las propiedades físicas de los componentes del robot, como la masa y la inercia. En este caso solo se abordará el problema dinámico directo con el fin de poder realizar un control de la posición de los actuadores del robot.

4. Reproducir los movimientos del robot dentro del entorno de simulación en tiempo real comprobando que no se producen retrasos apreciables. A continuación, se realizará una comparación aproximada del ancho de banda necesario para esta solución frente a una transmisión de vídeo.

5. Realizar una co-simulación junto a Matlab/Simulink empleando el modelo dinámico y validar que se comporta como el robot real. Se analizará el error cometido en el seguimiento de una trayectoria ensayando diversos controladores.

1.2 Estructura del documento

Para completar los objetivos propuestos, se partirá un planteamiento teórico donde se hará una descripción detallada del robot y de las herramientas informáticas utilizadas. En primer lugar, se hará una introducción a los sistemas de rehabilitación robotizados, analizando las necesidades específicas de los tratamientos de rehabilitación de miembro inferior.

A continuación, se hará una clasificación de los diferentes tipos de robots atendiendo a su arquitectura y se justificará la selección del robot paralelo, en base a los requisitos de la aplicación a la que esta destinado. Posteriormente, se abordarán los problemas de control presentes en el campo de la robótica y se realizará un estudio cinemático y dinámico del robot real.

En la ultima sección del planteamiento teórico, se presentará el software CoppeliaSim, realizando un recorrido por sus características y funcionalidades.

Seguidamente, se procederá al desarrollo práctico del proyecto, comenzando con la importación de un diseño CAD a la plataforma de simulación. A partir de este punto, se realizarán las modificaciones necesarias para generar dos modelos universales, uno puramente cinemático y otro dinámico. El capítulo se completará con el desarrollo de las dos aplicaciones propuestas en los objetivos.

Esta primera parte de la memoria finalizará con las conclusiones donde se evaluará el trabajo realizado y se propondrán posibles ampliaciones y aplicaciones.

La segunda parte del documento irá destinada a la elaboración de un presupuesto con un formato específico para proyectos de investigación. Se analizarán los diferentes gastos realizados durante el desarrollo de este trabajo y se presentará una tabla resumen con el importe final.

Finalmente, será posible encontrar dos anexos que aportarán información complementaria sobre este proyecto. En el primero de ellos se recopilarán las propiedades dinámicas del sistema robotizado, mientras que en el segundo se incluirán las diferentes piezas de código que se han desarrollado.

Capítulo 2

Planteamiento teórico

2.1 Sistemas de rehabilitación robotizados

El campo de la ingeniería robótica aplicada a la rehabilitación y ortopedia tiene sus orígenes en los años 60 con el desarrollo de exoesqueletos por parte de grupos de investigación en los Estados Unidos y Yugoslavia [1]. Aunque el primer grupo tenía el objetivo de incrementar las capacidades físicas del ser humano, incluso con fines militares, mientras que el segundo quería desarrollar tecnologías de asistencia para personas con movilidad reducida, ambos se enfrentaron a problemas similares, especialmente en como sería la interacción con el cuerpo humano.

Desde las primeras aplicaciones, la tecnologías robóticas se han hecho más maduras y han comenzando a tomar un papel importante en los campos de la medicina y rehabilitación. Los sistemas robotizados pueden suponer una gran ayuda para los profesionales, sirviendo como instrumentos auxiliares u operando de forma semi-independiente. Esto permite la realización de tratamientos de una forma más precisa y controlada, pudiendo monitorizar la respuesta del paciente mediante diversos sensores.

El uso de robots, tiene como ventaja adicional la operación de forma remota. En ocasiones los especialistas deben atender a pacientes de diferentes clínicas, que en casos extremos pueden estar situadas en diferentes países, por lo que disponer de este tipo de sistemas posibilita una atención más rápida sin la necesidad de desplazamientos. Además, en épocas como la actual, en la que se está experimentando el brote de COVID-19, sería posible llevar a cabo los tratamientos cumpliendo con las recomendaciones de seguridad y minimizando el contacto entre el fisioterapeuta y el paciente situándolos en salas separadas.

2.1.1 Rehabilitación de miembros inferiores

En el miembro inferior se encuentran unas de las articulaciones más importantes del cuerpo humano, el tobillo y la rodilla. Así mismo, las lesiones en estas articulaciones son las que se dan con más frecuencia en deportes y en la vida general [2], por lo que se está investigando activamente métodos para su tratamiento y prevención.

Las principales causas de los traumatismos de tobillo son la carencia de una fuerza, flexibilidad y propiocepción (sentido de la posición muscular) adecuadas, mientras que para la rodilla las lesiones suelen deberse a golpes o movimientos bruscos con un rango excesivo. Si no se realiza un tratamiento adecuado las lesiones pueden convertirse en crónicas disminuyendo el rango de movimiento y causando problemas de equilibrio. Diversos investigadores consideran que es posible aumentar la propiocepción mediante entranamientos de coordinación, mejorando el control postural y la fuerza de los músculos [2].

Teniendo en cuenta lo anterior, se han desarrollado diversos sistemas de rehabilitación, comenzando por dispositivos pasivos basados en bandas elásticas hasta que apareció el primer robot especializado, *the Rutgers Ankle* [3]. Este robot estaba diseñado específicamente para tratar lesiones de tobillo y contaba con 6 grados de libertad. No obstante, para la rehabilitación de tobillo y en general del miembro inferior, no son siempre necesarios todos ellos y es por ello que se han desarrollado sistemas de 4 grados de libertad como el que se aborda este en trabajo.

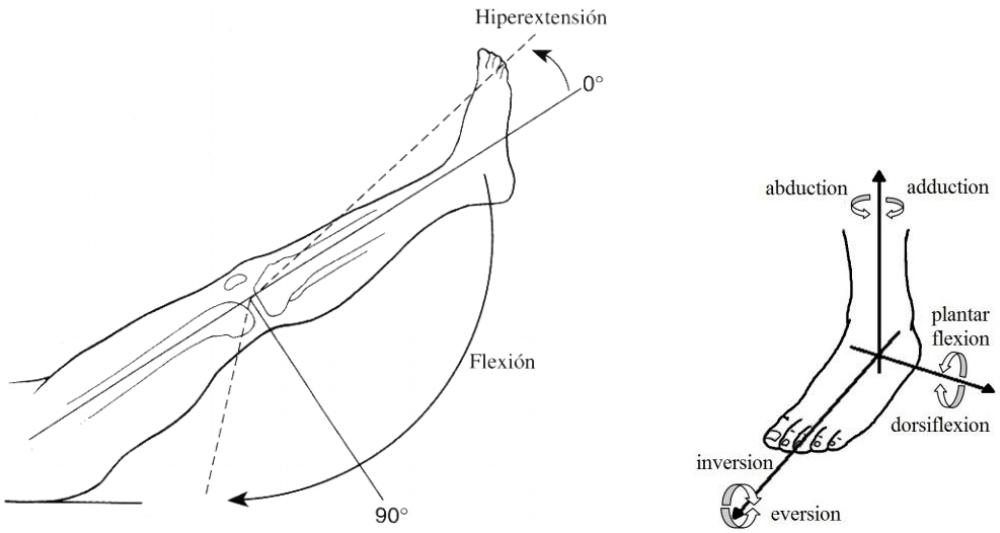


Figura 2.1: Movimientos de la rodilla (izquierda) y del tobillo (derecha)

Además de los grados de libertad, es importante conocer los rangos de movimiento de la articulación y los esfuerzos máximos que puede soportar. En la tabla 2.1 y la tabla 2.2 se recogen los rangos de movimiento para la rodilla [4] y el tobillo [5] respectivamente. Estos valores son orientativos y podrán variar dependiendo del grado de la lesión que se haya producido.

Tipo de movimiento	Rango
Flexión	130^{o}-135^{o}
Hiperextensión	9^{o}-11^{o}

Tabla 2.1: Rangos de movimiento de la rodilla

Tipo de movimiento	Rango
Flexión Dorsal	20º-30º
Flexión Plantar	37º-46º
Eversión	15º-26º
Inversión	22º-36º

Tabla 2.2: Rangos de movimiento del tobillo

El sistema rehabilitación debe poder alcanzar estos rangos y reproducir los movimientos naturales del paciente siguiendo trayectorias preestablecidas por el especialista. Esto se correspondería a un ejercicio de tipo pasivo, en el que el paciente no interviene y simplemente deja que el robot realice todos los movimientos.

Sin embargo, ha quedado demostrado que una rehabilitación activa puede acelerar el proceso de curación [6], por este motivo es interesante poder incorporar ejercicios de tipo activo en el tratamiento. En esta categoría se pueden encontrar los ejercicios activo-resisitivos y los ejercicios activo-asistenciales.

En los ejercicios activo-resistivos se pretende que el paciente venza una determinada fuerza definida por el especialista con el objetivo de fortalecer la articulación. Por su parte, en los ejercicios activo-asistenciales se asume que el paciente aún no es capaz de realizar el ejercicio de forma completamente independiente, lo que ocurre sobretodo al comienzo del proceso de rehabilitación. En este caso se debe ayudar al paciente a realizar los movimientos con una fuerza controlada, por lo que las referencias de posición se calculan mediante la Ecuación 2.1.

$$\vec{q}_{Nuevaref} = \vec{q}_{ref} + K_{st} \cdot \vec{T}_{xy} \tag{2.1}$$

Siendo $\vec{q}_{ref}$ la posición de referencia, K_{st} la constante de rigidez, $\vec{T}_{xy}$ el par medido en la articulación y $\vec{q}_{Nuevaref}$ la nueva posición de referencia dados K_{st} y $\vec{T}_{xy}$.

2.2 Robot paralelo de 4 grados de libertad

Como se ha introducido en la subsección 2.1.1 es necesario un sistema de al menos 4 grados de libertad para poder asistir en el proceso rehabilitación. Existen robots en diversas configuraciones que resultan adecuadas y que pueden ser clasificados en dos grupos principales:

- Robots de cadena cinemática abierta: en este tipo de configuración los eslabones que conforman el robot parten de una base y se conectan en serie unos con otros. En este grupo se encuentran por ejemplo los brazos robóticos utilizados en la industria del automóvil.

- Robots de cadena cinemática cerrada: en este tipo de configuración los elementos del robot parten y terminan en una misma base, creando uno o más lazos cerrados. También se puede denominar como configuración en paralelo. Un ejemplo clásico es la plataforma de Stewart [7].

También es posible encontrar robots donde se combinan cadenas cinemáticas abiertas y cerradas formando configuraciones híbridas.

Los robots paralelos presentan una estructura rígida con una alta relación carga/peso, lo que les otorga una gran precisión. No obstante, el área de trabajo de este tipo de robots suele ser

Figura 2.2: Robot serie (izquierda), robot paralelo (derecha)

menor que en los robots serie. Para las aplicaciones de rehabilitación es importante disponer de un sistema que pueda soportar los esfuerzos realizados por el paciente y que a su vez, sea capaz de realizar movimientos de forma precisa, por este motivo el robot paralelo es el candidato idóneo. En efecto, este tipo de configuración ha sido la elegida para el desarrollo del robot de rehabilitación estudiado en este trabajo, en concreto su última versión en el momento de redacción del documento.

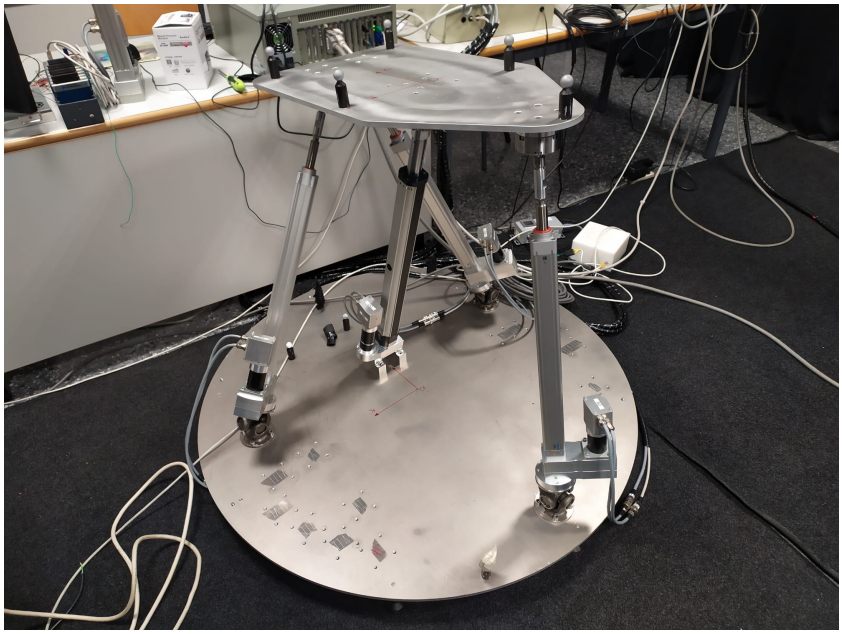


Figura 2.3: Robot de rehabilitación

2.2.1 Problema cinemático directo e inverso

El estudio cinemático de un sistema mecánico permite obtener las posiciones, velocidades y aceleraciones de sus componentes en base al movimiento que experimentan sin tener en cuenta las fuerzas que producen dicho movimiento [8].

En robótica se plantean dos tipos de problema dado un sistema de coordenadas de referencia y un punto de interés, que suele coincidir con el elemento terminal del robot:

- El problema cinemático directo consiste en determinar la posición y orientación del efector final en base al valor de las variables articulares activas (actuadores) del robot.

- El problema cinemático inverso consiste en determinar la configuración que debe adoptar el sistema, en concreto, los valores de los pares cinemáticos activos, para que el efector final alcance la posición y orientación deseadas.

2.2.2 Problema dinámico directo e inverso

El modelo dinámico de un sistema permite conocer las fuerzas que experimenta el sistema, teniendo en cuenta las propiedades inerciales del mismo, como las masas y el primer y segundo de inercia [8]. De forma análoga a los problemas cinemáticos, se pueden plantear dos tipos de problema dinámico:

- El problema dinámico directo consiste en localizar el elemento terminal en base a las fuerzas o pares que se aplican en las articulaciones activas del robot.

- El problema dinámico inverso consiste en determinar la fuerza o par que se debe aplicar a las articulaciones activas del robot para que el elemento terminal alcance la posición y orientación deseadas.

2.2.3 Modelo cinemático

El robot paralelo de rehabilitación, presenta una configuración 3UPS+RPU, permitiendo dos desplazamientos y dos rotaciones. Su modelo cinemático fue realizado originalmente por el Dr. Vicente Mata y ha sido adaptado por el doctorando José L. Pulloquinga. Para el modelado se ha aplicado la notación de Denavit-Hartenberg (DH) bajo la convención de Paul.

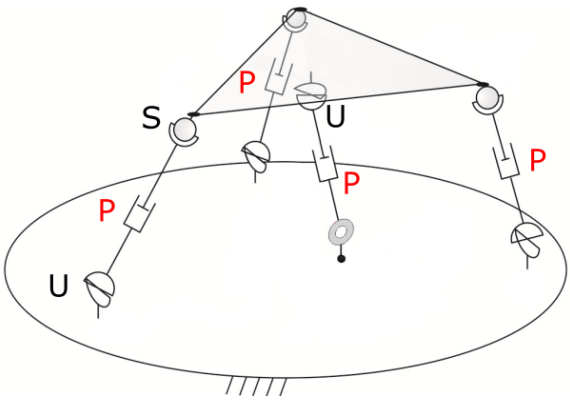


Figura 2.4: Robot 3UPS-RPU

El modelo dispone de dos sistemas de referencia, uno fijo $\{O_F - X_F Y_F Z_F\}$ correspondiente a la base, y otro móvil $\{O_M - X_M Y_M Z_M\}$ correspondiente a la plataforma superior. A partir de del sistema de referencia fijo se obtiene la posición de los vértices de las patas. Con el objetivo de optimizar el diseño y reducir las singularidades, la última versión del robot permite configurar de forma independiente la conexión de cada una de las tres patas UPS con las dos plataformas y además desplaza la pata RPU fuera del centro geométrico de la plataforma fija.

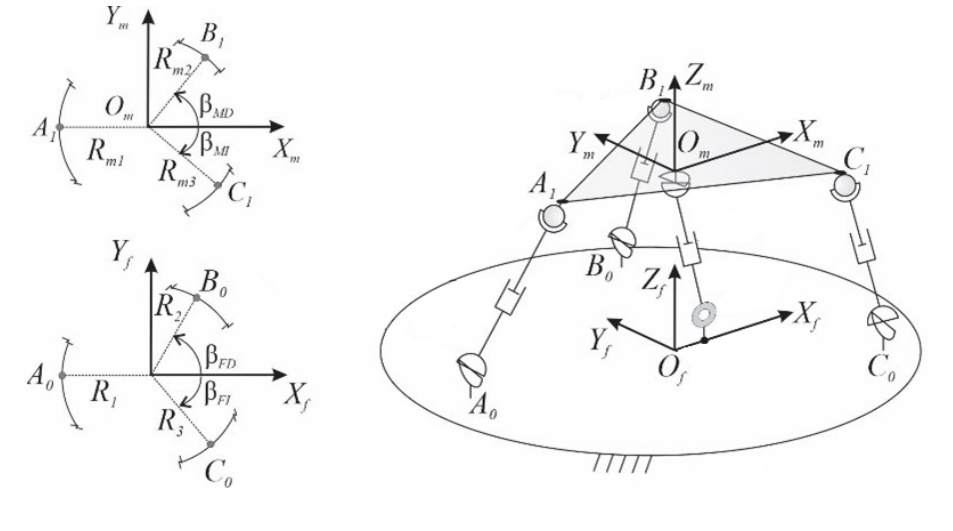


Figura 2.5: Posición de los vértices de las patas (izquierda), sistemas de referencia (derecha)

Los pares universales se modelizan mediante dos pares de rotación perpendiculares entre sí, mientras que el par esférico se modeliza mediante tres pares de rotación también perpendiculares entre sí. En cada pata se definen los sistemas de referencia necesarios, tal como se muestra en la figura 2.6.

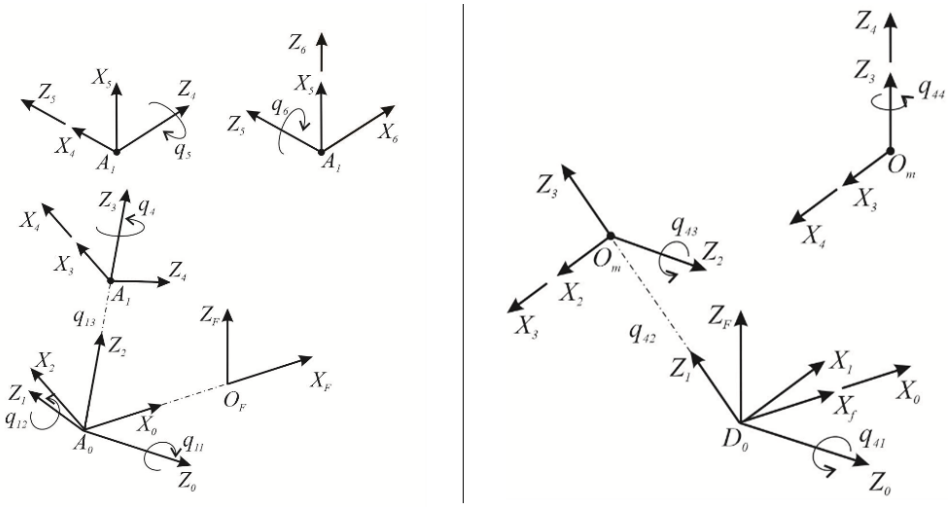


Figura 2.6: Sistemas de referencia y pares en las patas UPS (izquierda) y RPU (derecha)

Seguidamente se obtienen los parámetros de Denavit-Hartenberg, α, a, d y θ, para cada sistema de referencia local establecido en cada articulación que une dos barras. Dichos valores definen la relación entre el sistema de referencia de la barra $\{O_i - X_iY_iZ_i\}$ y el sistema de referencia de la barra a la que se conecta $\{O_{i-1} - X_{i-1}Y_{i-1}Z_{i-1}\}$. En primer lugar, se obtiene α_i, que representa el ángulo girado entre el eje z_{i-1} y z_i, siguiendo la regla de la mano derecha respecto al eje X_i. A continuación, se calcula la distancia mínima, a_i, entre los ejes Z_{i-1} y Z_i, que en este caso será siempre cero. Por último, d_i y θ_i se corresponden a las traslaciones y rotaciones en cada par. Atendiendo a este procedimiento se tabulan los parámetros obtenidos para las barras de cada tipo de pata.

Barra i-ésima	$\boldsymbol{\alpha_i}$	$\boldsymbol{a_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	$-\pi/2$	0	0	q_1
2	$\pi/2$	0	0	q_2
3	0	0	q_3	0
4	$\pi/2$	0	0	q_4
5	$\pi/2$	0	0	q_5
6	$\pi/2$	0	0	q_6

Tabla 2.3: Parámetros DH de las patas UPS

Barra i-ésima	$\boldsymbol{\alpha_i}$	$\boldsymbol{a_i}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	$-\pi/2$	0	0	q_1
2	$-\pi/2$	0	q_2	π
4	$\pi/2$	0	0	q_3
4	0	0	0	q_4

Tabla 2.4: Parámetros DH de la pata RPU

Los parámetros α, a, a y θ se recogen en la matriz de Denavit-Hartenberg ($H_{i_1,i}$), también conocida como matriz de transformación homogénea:

$$H_{i_1,i} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\cos\alpha_i & r_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\cos\alpha_i & r_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Por otro lado se define la matriz de rotación de Euler de la plataforma móvil (${}^fR_m$), que viene expresada en ángulos de Tait-Bryan. Esta notación asigna un ángulo de inclinación cero al cuerpo en horizontal, a diferencia de los ángulos de Euler que le asignarían $\frac{\pi}{2}$.

$${}^fR_m = R_x(\phi)R_y(\theta)R_z(\psi)$$

$${}^fR_m = \begin{bmatrix} \cos\theta\cos\psi & -\cos\theta\sin\psi & \sin\theta \\ \sin\psi & \cos\psi & 0 \\ -\sin\theta\cos\psi & \sin\theta\sin\psi & \cos\theta \end{bmatrix} \tag{2.3}$$

Tras determinar las matrices de transformación homogénea y la matriz de rotación de Euler, en base a la posición y orientación de la plataforma superior $\{x_m, y_m, z_m, \phi, \theta, \psi\}$ se calculan las coordenadas q_{i1}, q_{i2} y q_{i3} de las patas UPS ($i = 1, 2, 3$) y q_{i1} y q_{i2} de la pata RPU ($i = 4$). Los pares activos se corresponden con la coordenada q_{i3} de las patas exteriores y q_{i2} de la pata central. Para determinar las coordenadas de interés se resuelve el sistema definido por los lazos cinemáticos representados en la figura siguiente.

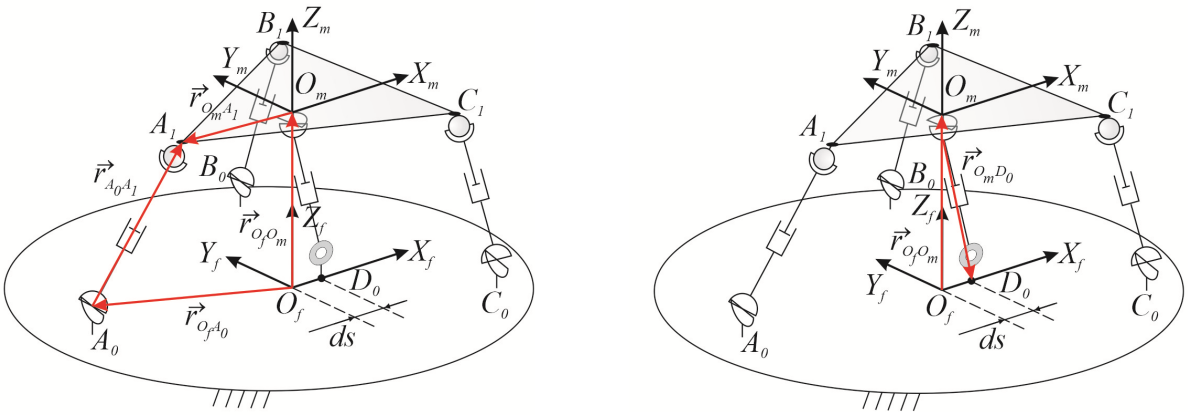


Figura 2.7: Lazos cinemáticos en las patas UPS (izquierda) y RPU (derecha)

De esta forma las coordenadas del punto A_1 vendrán dadas por,

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} + {}^{f}R_m \begin{bmatrix} -R_m \\ 0 \\ 0 \end{bmatrix} = \left[H_{FP1}H_{01}(q_1)H_{12}(q_2)H_{23}(q_3)\right] \tag{2.4}$$

para el punto B_1,

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} + {}^{f}R_m \begin{bmatrix} R_m\cos(\beta) \\ R_m\,\mathrm{sen}(\beta) \\ 0 \end{bmatrix} = \left[H_{FP2}H_{01}(q_1)H_{12}(q_2)H_{23}(q_3)\right] \tag{2.5}$$

para el punto C_1,

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} + {}^{f}R_m \begin{bmatrix} R_m\cos(\beta) \\ R_m\,\mathrm{sen}(\beta) \\ 0 \end{bmatrix} = \left[H_{FP3}H_{01}(q_1)H_{12}(q_2)H_{23}(q_3)\right] \tag{2.6}$$

y para la pata central 4, se empleará la expresión

$$\begin{bmatrix} x_m \\ y_m \\ z_m \end{bmatrix} = \left[H_{FP4}H_{01}(q_1)H_{12}(q_2)\right] \tag{2.7}$$

siendo H_{FPi} la matriz de transformación desde el origen fijo hasta el origen local de cada pata.

Cabe destacar que debido a la geometría del robot, no es posible realizar desplazamientos en Y_f ni giros alrededor de X_f. De esta forma tanto y_m como ϕ son iguales a cero.

2.2.4 Modelo dinámico

Al igual que en el modelado cinemático, la formulación dinámica ha sido realizada por el Dr. Vicente Mata y adaptada a la última versión del robot por el doctorando José L. Pulloquinga. En este caso se va a realizar un análisis de las fuerzas y momentos que afectan al robot con el objetivo de poder determinar las fuerzas o acciones de control que deben aplicar los actuadores.

Se va a trabajar con una simplificación del robot compuesta por sólidos rígidos básicos. Los componentes que intervendrán en los cálculos serán la plataforma superior y las 4 patas, estando estas últimas divididas en un cilindro y en un émbolo. De estos elementos se han obtenido las propiedades dinámicas referidas a un sistema de referencia local mediante la aplicación SolidWorks. Estos datos quedan recogidos en el Apéndice A.

El modelado se comienza por las patas, en concreto por las patas exteriores que tienen una configuración UPS. Seguidamente se realizará el mismo procedimiento para la pata interior, con configuración RPU. En la figura siguiente se muestra la representación de los dos tipos de pata como dos sólidos rígidos y los sistemas de referencia empleados.

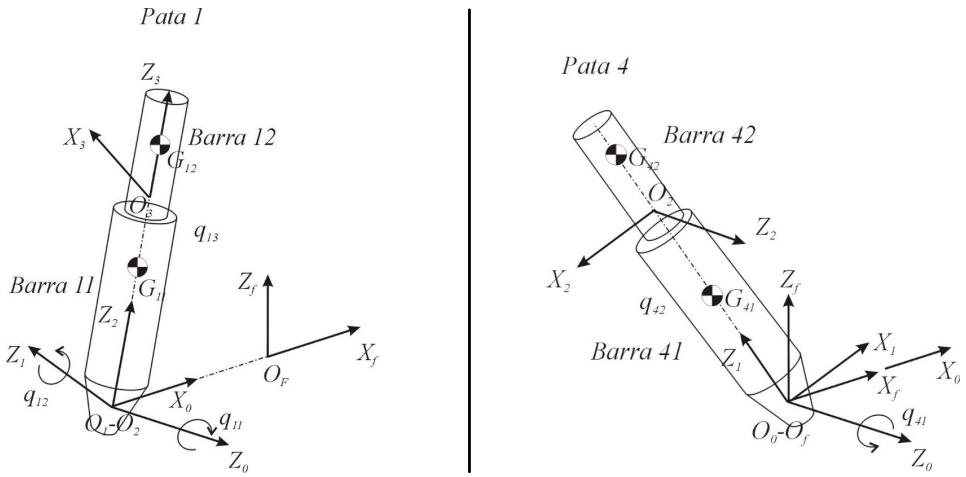


Figura 2.8: Simplificación de las patas del robot: patas 1,2 y 3 (izquierda), pata 4 (izquierda)

Atendiendo a las restricciones cinemáticas de las patas, se obtienen las velocidades y aceleraciones de cada barra en su centro de masas, que es el punto en el que se supondrán aplicadas las fuerzas. Los parámetros angulares, son iguales medidos tanto desde el centro de masas como del sistema local establecido en una articulación asociada al solido correspondiente.

$$\vec{\omega}_{11} = \vec{\omega}_{12} = \dot{q}_{11} \cdot \vec{u}_{Z_0} + \dot{q}_{12} \cdot \vec{u}_{Z_1} \tag{2.8}$$

$$\vec{\alpha}_{11} = \vec{\alpha}_{12} = \ddot{q}_{11} \cdot \vec{u}_{Z_0} + \ddot{q}_{12} \cdot \vec{u}_{Z_1} + \dot{q}_{12} \cdot \left(\tilde{\omega}_1^f \cdot R_1 \cdot \vec{u}_{Z_1} \right) \tag{2.9}$$

Siendo:

$$\vec{u}_{Z_0} = \begin{bmatrix} 0 & -1 & 0 \end{bmatrix}^T$$

$$\vec{u}_{Z_1} = \begin{bmatrix} -\sin q_{11} & 0 & \cos q_{11} \end{bmatrix}^T$$

Las velocidades y aceleraciones lineales expresadas respecto al centro de gravedad no coinciden con las que se pueden calcular para el sistema de referencia local establecido en las articulaciones, por lo tanto se debe considerar la distancia vectorial desde la articulación y el centro de gravedad del solido rígido.

$$\vec{v}_{G_{11}} = \vec{\omega}_{11} \times \vec{r}_{O_2 G_{11}} \tag{2.10}$$

$$\vec{\alpha}_{G_{11}} = \vec{\omega}_{11} \times (\vec{\omega}_{11} \times \vec{r}_{O_2 G_{11}}) + \vec{\alpha}_{11} \times \vec{r}_{O_2 G_{11}} \tag{2.11}$$

Siendo:

$$^2\vec{r}_{O_2 G_{11}} = \begin{bmatrix} x_{G_{11}} & y_{G_{11}} & z_{G_{11}} \end{bmatrix}^T$$

En el caso de la barra superior, además será necesario añadir la componente correspondiente al desplazamiento relativo entre las dos barras.

$$\vec{v}_{G_{12}} = \vec{v}_{O_2} + \vec{\omega}_{11} \times \vec{r}_{O_2 G_{12}} + \dot{q}_{13} \cdot \vec{u}_{Z_1} \tag{2.12}$$

$$\vec{\alpha}_{G_{12}} = \vec{v}_{0_2} + \vec{\omega}_{11} \times (\vec{\omega}_{11} \times \vec{r}_{O_2 G_{11}}) + \vec{\alpha}_{11} \times \vec{r}_{O_2 G_{12}} + \ddot{q}_{13} \cdot \vec{u}_{Z_1} + 2 \cdot (\vec{\omega}_{12} \times (\dot{q}_{13} \times \vec{u}_{Z_2})) \tag{2.13}$$

Siendo:

$$\vec{r}_{O_2 G_{12}} = \vec{r}_{O_2 O_3} + {}^f R_3 \cdot {}^3\vec{r}_{O_3 G_{12}}$$

$$\vec{r}_{O_2 O_3} = {}^f R_2 \cdot \begin{bmatrix} 0 \\ 0 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos q_{11} \cdot \sin q_{11} \cdot q_{13} \\ -\cos q_{12} \cdot q_{13} \\ \sin q_{11} \cdot \sin q_{11} \cdot q_{13} \end{bmatrix}$$

$$^3\vec{r}_{O_3 G_{12}} = \begin{bmatrix} x_{G_{12}} & y_{G_{12}} & z_{G_{12}} \end{bmatrix}^T$$

Para la pata central se procede de forma análoga, se parte de las velocidades y aceleraciones locales y después se calculan estos parámetros en el centro de masas.

$$\vec{\omega}_{41} = \vec{\omega}_{42} = \dot{q}_{41} \cdot \vec{u}_{Z_0} \tag{2.14}$$

$$\vec{\alpha}_{41} = \vec{\alpha}_{42} = \ddot{q}_{41} \cdot \vec{u}_{Z_0} \tag{2.15}$$

$$\vec{v}_{G_{41}} = \vec{\omega}_{41} \times \vec{r}_{O_1 G_{41}} \tag{2.16}$$

$$\vec{\alpha}_{G_{41}} = \vec{\omega}_{41} \times (\vec{\omega}_{11} \times \vec{r}_{O_2 G_{41}}) + \vec{\alpha}_{41} \tag{2.17}$$

$$\vec{v}_{G_{42}} = \vec{\omega}_{41} \times \vec{r}_{O_2 G_{42}} + \dot{q}_{42} \cdot \vec{u}_{Z_1} \tag{2.18}$$

$$\vec{\alpha}_{G_{42}} = \vec{\omega}_{41} \times (\vec{\omega}_{41} \times \vec{r}_{O_2 G_{41}}) + \vec{\alpha}_{41} \times \vec{r}_{O_1 G_{42}} + \ddot{q}_{42} \cdot \vec{u}_{Z_1} + 2 \cdot (\vec{\omega}_{41} \times (\dot{q}_{42} \cdot \vec{u}_{Z_1})) \tag{2.19}$$

Siendo:

$$\vec{r}_{O_2 G_{41}} = \begin{bmatrix} x_{G_{41}} & y_{G_{41}} & z_{G_{41}} \end{bmatrix}^T$$

$$\vec{r}_{O_1 G_{42}} = \vec{r}_{O_1 O_2} + {}^f R_2 \cdot {}^2\vec{r}_{O_2 G_{42}} = \vec{r}_{O_1 O_2} + {}^f R_2 \cdot \begin{bmatrix} x_{G_{42}} & y_{G_{42}} & z_{G_{42}}^T \end{bmatrix}$$

A continuación, se obtienen las velocidades y aceleraciones para la plataforma móvil a partir de las derivadas temporales de los ángulos de Euler.

$$\vec{\omega}_m = \begin{bmatrix} -\sin \Psi \cdot \dot{\Theta} \\ \cos \Psi \cdot \dot{\Theta} \\ \dot{\Psi} \end{bmatrix} \tag{2.20}$$

$$\vec{\alpha}_m = \begin{bmatrix} -\sin \Psi \cdot \ddot{\Theta} - \cos \Psi \cdot \dot{\Psi} \cdot \dot{\Theta} \\ \cos \Psi \cdot \ddot{\Theta} - \cos \Psi \cdot \dot{\Psi} \cdot \dot{\Theta} \\ \ddot{\Psi} \end{bmatrix} \tag{2.21}$$

$$\vec{v}_{G_m} = \vec{v}_{O_m} + \vec{\omega}_m \times \vec{r}_{O_m G_m} \tag{2.22}$$

$$\vec{a}_{G_m} = \vec{a}_{O_m} + \vec{\omega}_m \times \left(\vec{\omega}_m \times \vec{r}_{O_m G_m)}\right) + \vec{\alpha}_m \times \vec{r}_{O_m Gm} \tag{2.23}$$

Siendo:

$$\vec{v}_{O_m} = \begin{bmatrix} \dot{x}_m & 0 & \dot{z}_m \end{bmatrix}$$

$$\vec{a}_{O_m} = \begin{bmatrix} \ddot{x}_m & 0 & \ddot{z}_m \end{bmatrix}$$

$$\vec{r}_{O_m G_m} = {}^f R_m \cdot \vec{r}_{O_m Gm} = {}^f R_m \cdot \begin{bmatrix} X_{G_m} & Y_{G_m} & Z_{G_m} \end{bmatrix}^T$$

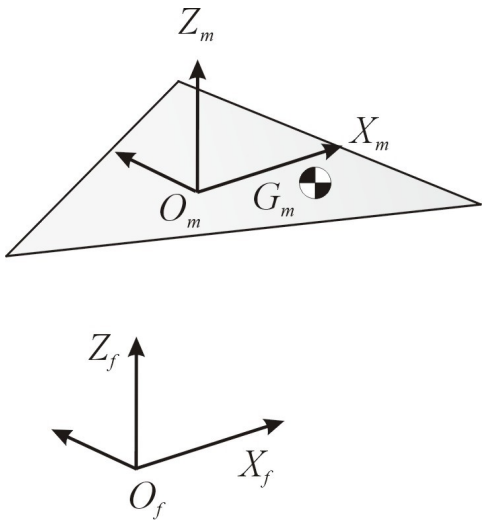


Figura 2.9: Simplificación de la plataforma móvil del robot

Con las velocidades y aceleraciones del centro de gravedad de cada solido rígido es posible obtener las acciones inerciales y los pesos aplicados sobre dicho punto.

$$\vec{F}_{in_i} = -m_i \cdot \vec{\alpha}_{G_i} \tag{2.24}$$

$$\vec{T}_{in_i} = -\left(I_{G_i} \cdot \vec{\alpha}_{ij} + \vec{\omega}_i \cdot (I_{G_i} \cdot \vec{\omega}_i)\right) \tag{2.25}$$

$$\vec{P}_i = \begin{bmatrix} 0 & 0 & -m_i & g \end{bmatrix}^T \tag{2.26}$$

Siendo:

$${}^k I_{G_i} = \begin{bmatrix} Ixx_i & -Ixy_i & -Ixz_i \\ & -Iyy_i & -Iyz_i \\ \ddots & & Izz_i \end{bmatrix}$$

$$I_{G_i} = {}^f R_k \cdot {}^k I_{G_i} \cdot ({}^f R_k)^T$$

Con ${}^k I_{G_i}$ siendo la matriz de inercias de cada elemento respecto al sistema de referencia local.

Para obtener las fuerzas de inercia generalizadas se debe tener en cuenta que se está trabajando con el siguiente conjunto de 15 coordenadas generalizadas dependientes.

$$\vec{q} = \left[\underbrace{q_{11}, q_{12}, q_{21} q_{22}, q_{31}, q_{32}, q_{41}, x_m, z_m, \Theta, \Psi}_{Secundarias}, \underbrace{q_{13}, q_{23}, q_{33}, q_{42}}_{Independientes} \right]^T \tag{2.27}$$

Las fuerzas de inercia generalizadas se obtienen derivando parcialmente las velocidades (lineales y angulares) y multiplicando el resultado por las fuerzas y momentos aplicados.

$$\vec{Q}_{in} = \vec{F}_{in_{11}} \cdot \frac{\partial \vec{v}_{G_{11}}}{\partial \dot{\vec{q}}} + \vec{T}_{in_{11}} \cdot \frac{\partial \vec{\omega}_{11}}{\partial \dot{\vec{q}}} + \vec{F}_{in_{12}} \cdot \frac{\partial \vec{v}_{G_{12}}}{\partial \dot{\vec{q}}} + \vec{T}_{in_{12}} \cdot \frac{\partial \vec{\omega}_{12}}{\partial \dot{\vec{q}}} + \cdots + \vec{F}_{in_m} \cdot \frac{\partial \vec{v}_{G_m}}{\partial \dot{\vec{q}}} + \vec{T}_{in_m} \cdot \frac{\partial \vec{\omega}_m}{\partial \dot{\vec{q}}} \quad (2.28)$$

Para obtener la componente gravitacional se derivan parcialmente las velocidades y se multiplican por los pesos.

$$\vec{Q}_{grav} = \vec{P}_{in_{11}} \cdot \frac{\partial \vec{v}_{G_{11}}}{\partial \dot{\vec{q}}} + \vec{P}_{in_{12}} \cdot \frac{\partial \vec{v}_{G_{12}}}{\partial \dot{\vec{q}}} + \cdots + \vec{P}_{in_m} \cdot \frac{\partial \vec{v}_{G_m}}{\partial \dot{\vec{q}}} \quad (2.29)$$

Las fuerzas activas de los actuadores se calculan de forma similar teniendo en cuenta que su dirección de actuación coincide con las coordenadas generalizadas independientes.

$$\vec{Q}_{ex} = \vec{F}_1 \cdot \frac{\partial \vec{v}_{G_{12}}}{\partial \dot{\vec{q}}} + \vec{F}_2 \cdot \frac{\partial \vec{v}_{G_{22}}}{\partial \dot{\vec{q}}} + \vec{F}_3 \cdot \frac{\partial \vec{v}_{G_{32}}}{\partial \dot{\vec{q}}} + \vec{F}_4 \cdot \frac{\partial \vec{v}_{G_{42}}}{\partial \dot{\vec{q}}} \quad (2.30)$$

Siendo:

$$\vec{F}_i = {}^f R_r \cdot \begin{bmatrix} 0 & 0 & F_i \end{bmatrix}^T$$

En último lugar, obtienen las acciones ejercidas por el entorno (por el paciente) a la plataforma.

$$\vec{Q}_{ent} = \vec{F}_m \cdot \frac{\partial \vec{v}_D}{\partial \dot{\vec{q}}} + \vec{T}_m \cdot \frac{\partial \vec{\omega}_m}{\partial \dot{\vec{q}}} \quad (2.31)$$

Siendo:

$$\vec{F}_m = \begin{bmatrix} F_{m_x} & F_{m_y} & F_{m_z} \end{bmatrix}^T \qquad \vec{T}_m = \begin{bmatrix} T_{m_x} & T_{m_y} & T_{m_z} \end{bmatrix}^T$$
$$\vec{v}_d = \vec{v}_{O_m} + \vec{\omega}_m \times \vec{r}_{O_m D}$$
$$\vec{r}_{O_m D} = {}^f R_m \cdot \begin{bmatrix} x_D & y_D & z_D \end{bmatrix}^T$$

La dinámica del robot paralelo queda definida mediante la siguiente expresión.

$$\vec{Q}_{in} + \vec{Q}_{grav} + \vec{Q}_{ex} + \vec{Q}_{ext} - [\Phi_q]^T \cdot \vec{\lambda} = 0 \quad (2.32)$$

En la ecuación anterior, $[\Phi_q]$ se corresponde al jacobiano de las restricciones (obtenido a partir de las Ecuaciones 2.4, 2.5, 2.6 y 2.7) mientras que $\vec{\lambda}$ vector de multiplicadores de Lagrange. Estos términos son necesarios para poder cerrar las cadenas cinemáticas del robot paralelo. Si se desea obtener únicamente las fuerzas en las coordenadas generalizadas activas del robot, la información que aportan los multiplicadores de Lagrange no es relevante. Estos pueden ser eliminados de la ecuación de movimiento empleando un complemento ortogonal que verifique:

$$(R^*)^T \cdot [\Phi_q]^T = 0 \quad (2.33)$$

Así, el complemento ortogonal definido mediante el principio de variable separable [9] queda expresado de la siguiente manera.

$$R^* = \begin{bmatrix} -[\phi_p^s]_{M\times M}^{-1} \cdot [\phi_q^i]_{M\times F} \\ 1_{F\times F} \end{bmatrix}_{N\times F} \tag{2.34}$$

Con:

ϕ_q^s: Matriz jacobiana de las juntas pasivas.
ϕ_q^i: Matriz jacobiana de las juntas activas.

Se añade el complemento ortogonal a la ecuación de movimiento y se descompone el término Q_{in} en factores dependientes de la aceleración $\hat{M}_{N\times N} \cdot \vec{\ddot{q}}_{N\times 1}$ y de la velocidad $\vec{Q}_{cyc_{N\times N}} \cdot \vec{\dot{q}}_{N\times 1}$.

$$\begin{aligned}(R^*)_{F\times N}^T \cdot (\hat{M}_{M\times N} \cdot \vec{\ddot{q}}_{N\times 1} + Q_{cyc_{N\times N}} \cdot \vec{\dot{q}}_{N\times 1} + \vec{Q}_{grav_{N\times 1}} + \vec{Q}_{ent_{N\times 1}}) = \\ \underbrace{(R^*)_{F\times N}^T \cdot (Q_{ext_{N\times F}} \cdot \vec{F}_{act_{F\times 1}})}_{\text{fuerzas generalizadas}}\end{aligned} \tag{2.35}$$

Como el resultado de $(R^*)_{F\times N}^T \cdot Q_{ext_{N\times F}}$ es una matriz identidad, a partir de la expresión anterior se pueden obtener las 4 fuerzas generalizadas correspondientes a las acciones de control.

$$\vec{F}_{act} = (R^*)_{F\times N}^T \cdot (\hat{M}_{N\times N} \cdot \vec{\ddot{q}}_{N\times 1} + \vec{Q}_{cyc_{N\times N}} \cdot \vec{\dot{q}}_{N\times 1} + \vec{Q}_{grav_{N\times 1}} + \vec{Q}_{ent_{N\times 1}}) \tag{2.36}$$

Con:

$\vec{F}_{act}$: Fuerzas activas generalizadas (Acciones de control).
F: Número de actuadores del robot (cuatro en este caso).
N: Número de articulaciones activas y pasivas (variables generalizadas).
$\hat{M}$: Matriz de masas del robot.
$\vec{q}$: Posiciones generalizadas.
$\vec{\dot{q}}$: Velocidades generalizadas.
$\vec{\ddot{q}}$: Aceleraciones generalizadas.
$\vec{Q}_{cyc}$: Matriz de términos centrífugos y de Coriolis.
$\vec{Q}_{grav}$: Términos gravitacionales del robot.
$\vec{Q}_{ent}$: Términos de fuerzas y pares del entorno sobre el robot.
R^*: Complemento ortogonal.

Por último, reagrupando términos es posible obtener la expresión en forma compacta.

$$M\left(\vec{q}, \vec{\Phi}\right) \cdot \vec{\ddot{q}} + \vec{C}\left(\vec{q}, \vec{\dot{q}}, \vec{\Phi}\right) \cdot \vec{\dot{q}} + \vec{G}\left(\vec{q}, \vec{\Phi}\right) = \vec{\tau} \tag{2.37}$$

Con:

$(R^*)^T \cdot \hat{M} = M$: Matriz de masas del robot.
$(R^*)^T \cdot \vec{Q}_{cyc} = \vec{C}$: Términos centrífugos y de Corioils.
$(R^*)^T \cdot (\vec{Q}_{grav} + \vec{Q}_{ent}) = \vec{G}$: Términos gravitacionales.
$\vec{F}_{act} = \vec{\tau}$: Fuerzas generalizadas.

2.3 Introducción a CoppeliaSim

CoppeliaSim, anteriormente conocido como V-REP (Virtual Robot Experimentation Platform), es un entorno de simulación de robots originalmente desarrollado por Toshiba y que actualmente es mantenido por Coppelia Robotics AG.

Se trata de un software muy completo, que ofrece multitud de herramientas para el modelado, la simulación y el control de cualquier tipo de robot. A continuación, se pretende hacer una introducción a las funcionalidades del programa (en su versión 4.1.0), que servirá de justificación para los pasos seguidos en el desarrollo práctico de este trabajo.

2.3.1 Interacción con la aplicación

Existen dos formas de interactuar con CoppeliaSim, mediante el uso de su interfaz gráfica o mediante su API (Application Programming Interface) con la posibilidad de emplear distintos lenguajes de programación.

La interfaz gráfica permite interactuar con el programa mediante menús y cuadros de diálogo, ofreciendo una representación visual del entorno de simulación y de las distintas herramientas. Resulta conveniente especialmente para la creación y organización de modelos y sus componentes. En la figura 2.10 se muestra la ventana de la aplicación.

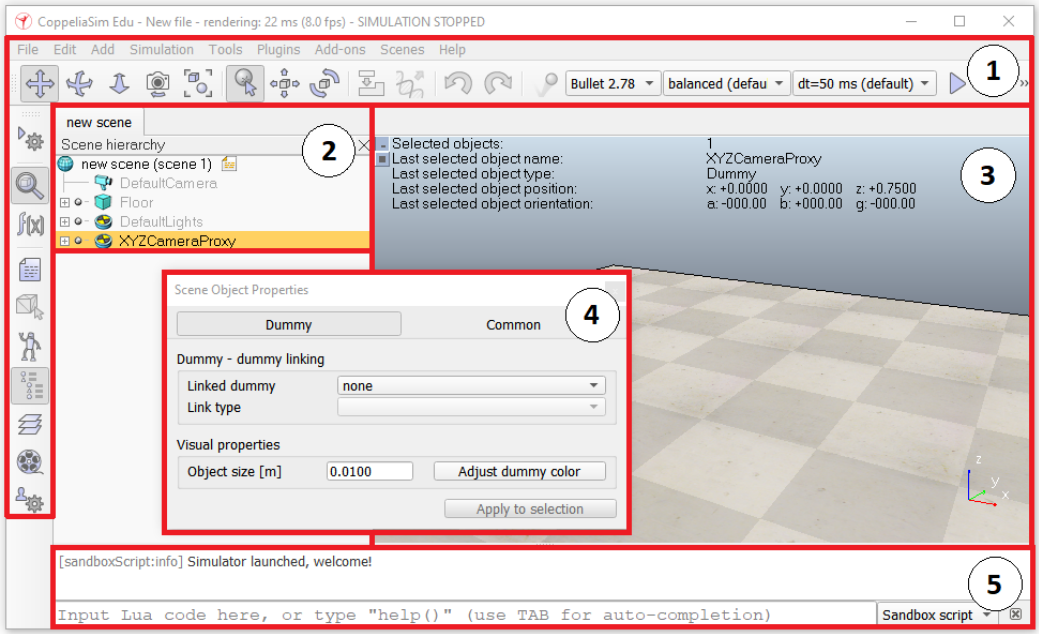


Figura 2.10: Menús (1), Jerarquía (2), Entorno de simulación (3), Cuadro de diálogo (4), Terminal (5)

Por su parte, la API permite interactuar por completo con todas las funcionalidades del programa mediante código. Este método tiene la ventaja de poder recoger las configuraciones y acciones deseadas en un fichero, que se puede ejecutar instantáneamente en cualquier momento, incluso durante la simulación. Esto ofrece una mayor flexibilidad y facilita la automatización de diferentes tareas.

La lista de funciones que incorpora la API es muy extensa, no obstante, la forma de trabajar con ella suele ser siempre la siempre la misma. En primer lugar, se debe crear un manejador (*handle*) asociado al elemento con el cual se desea interactuar:

```
handle = sim.getObjectHandle('Element_name')
```

A continuación, se puede acceder a cualquiera de los métodos asociados al tipo de elemento que se ha seleccionado:

```
sim.setObjectInt32Parameter(objectHandle, parameterID, parameter)
```

Dentro de la aplicación, es posible acceder a la API regular desde el terminal mostrado en la figura 2.10 o mediante un *script*, empleándose en ambos casos LUA, un lenguaje de programación interpretado.

Para añadir un *script* se debe hacer click derecho sobre el elemento al cual se quiere asociar el script y selecionar ***Add***, ***Asociated child script***. Seguidamente, es posible elegir dos tipos de *script*: ***Non threaded*** y ***Threaded***. La principal diferencia entre ellos es que el primer caso incluye funciones que se llaman automáticamente en cada paso de simulación, mientras que el segundo constituye un hilo independiente, que debe ser controlado por el usuario.

En ambos casos, al añadir un nuevo *script* se genera una plantilla que incorpora diversas funciones características de cada tipo de *script*. Por ejemplo, para un *script* de tipo *non-threaded*, se tienen funciones de inicialización, de sensado, de actuación y de finalización.

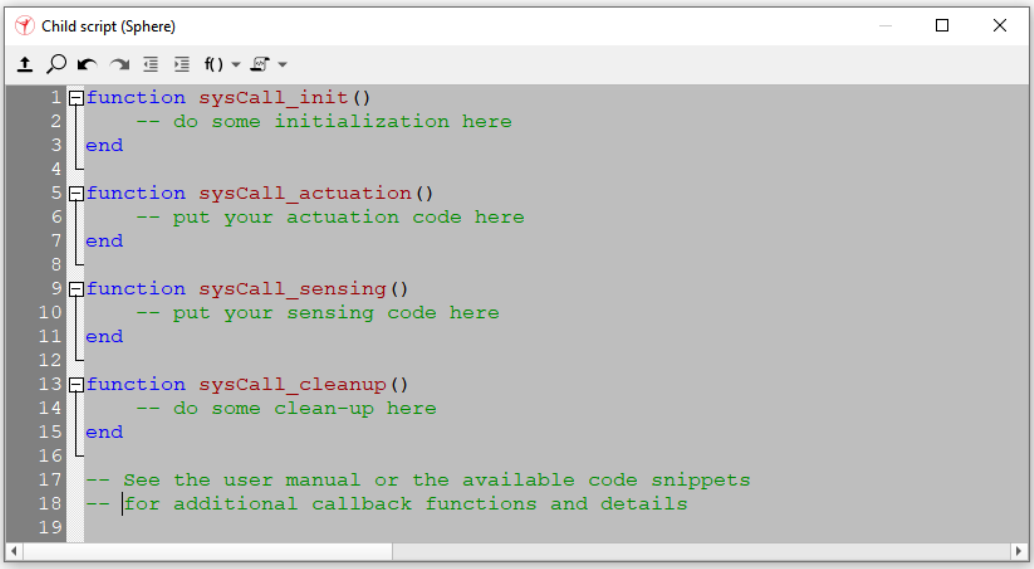


Figura 2.11: Ejemplo de non-threaded script

2.3.2 API remota clásica

CoppeliaSim se puede enlazar con programas externos mediante su API remota, que soporta distintos lenguajes de programación, C/C++, Python, Matlab, etc. De esta forma es posible establecer un control remoto, recopilar datos o realizar co-simulaciones. En el caso de la API remota clásica, la comunicación se puede realizar por medio de *sockets* TCP/IP, actuando CoppeliaSim como servidor y el programa externo como cliente.

Las funciones de la API remota requieren dos parámetros adicionales, el modo de operación y el identificador del cliente. A su vez, estas devuelven un código de error que permite verificar si se ha completado la solicitud correctamente. El resto de funcionalidades de la API remota son análogas a la API regular.

El modo de operación permite al usuario especificar la forma en la que se realizará la ejecución de la acción asociada a la petición para controlar el progreso de la simulación. La API remota de CoppeliaSim incluye los siguientes modos de operación:

- **Llamadas a funciones con espera:** este tipo de solicitudes pausan la ejecución del código del cliente hasta que se obtenga una respuesta del servidor.

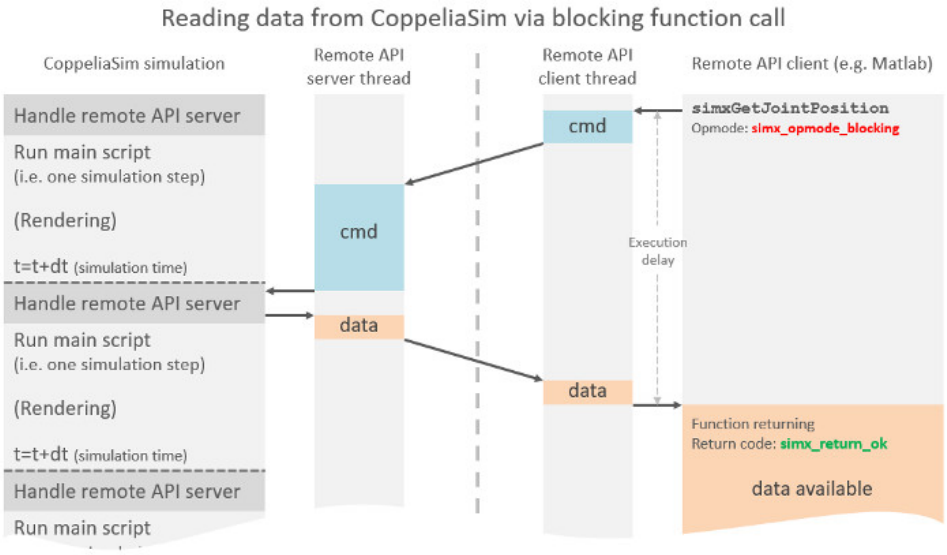


Figura 2.12: Llamada a función con espera

- **Llamadas a funciones sin espera:** este tipo de solicitudes no pausan la ejecución del código del cliente tras ser enviadas. No se tiene garantía de cuando se va a recibir una respuesta del servidor. Existen situaciones en las que se desea enviar datos que deben ser procesados en un mismo paso de simulación por el servidor, para ello se puede pausar la comunicación cargar las solicitudes necesarias y reanudar la comunicación. Cabe destacar que pausar la comunicación no implica que ejecución de la simulación en el servidor se detenga.

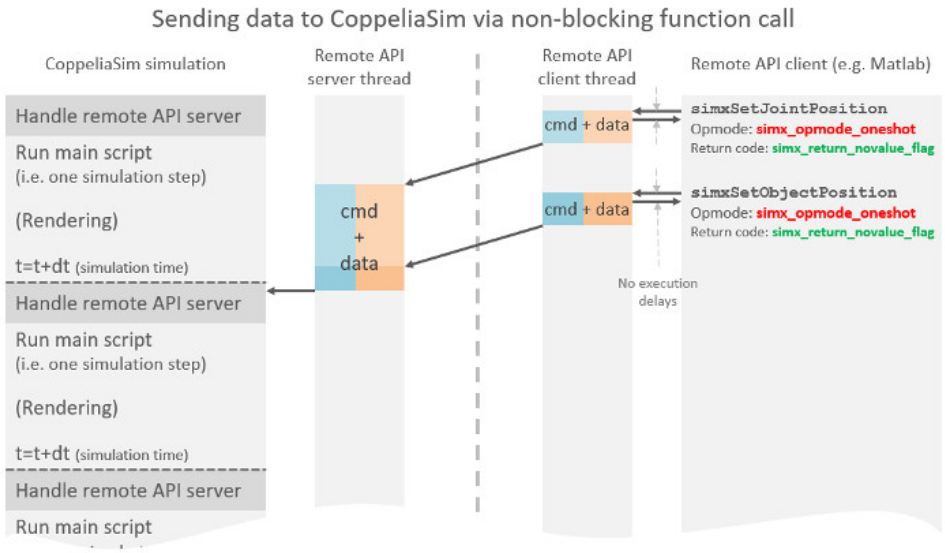


Figura 2.13: Llamada a función sin espera

- **Transmisión continua de datos:** si el cliente va a realizar solicitudes del mismo tipo, el servidor se puede anticipar y precargar dichos datos en una cola de tipo FIFO (*First In First Out*). El cliente podrá consumir los datos de la cola siempre que estén disponibles.

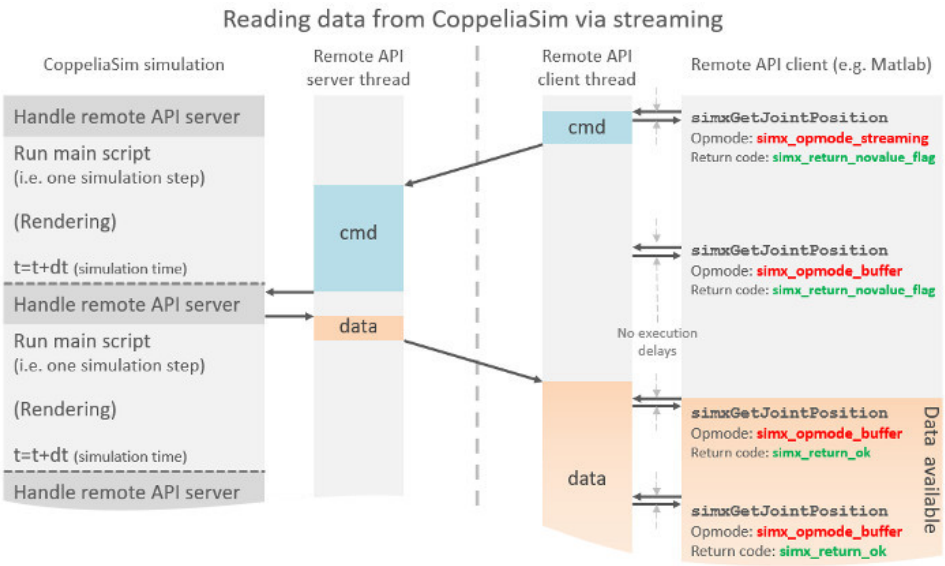


Figura 2.14: Transmisión continua de datos

- **Operación síncrona**: este modo se debe utilizar si se desea que el cliente controle el avance de la simulación dentro de CoppeliaSim. En este caso, el cliente además de las solicitudes propias, deberá realizar llamadas a la función *simxSynchronousTrigger* para que se ejecute el siguiente paso de simulación.

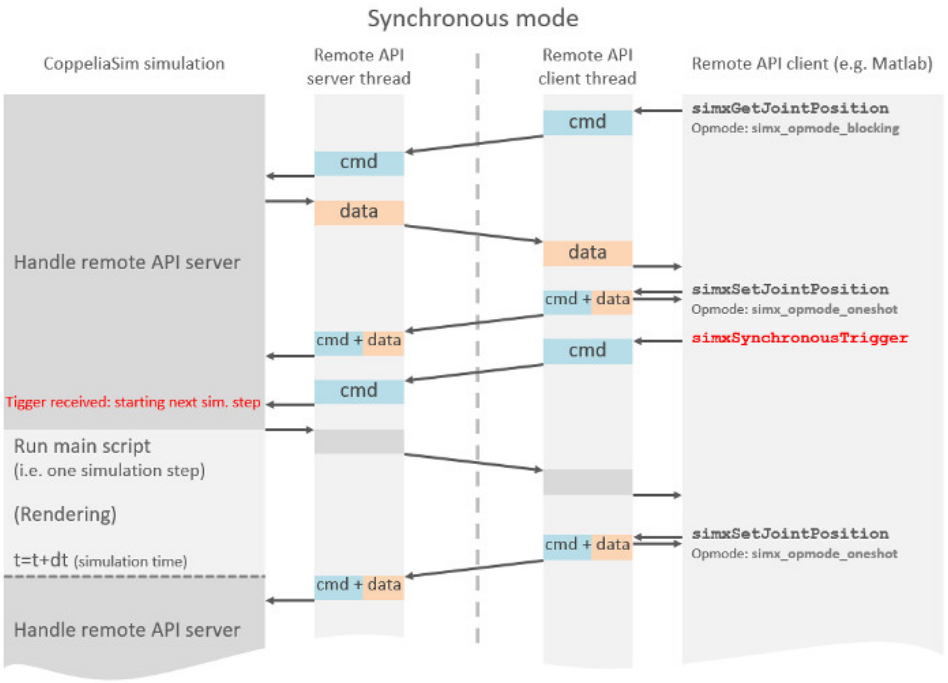


Figura 2.15: Ejecución síncrona

Si se desea conectar a CoppeliaSim, se debe tener un servidor en marcha que acepte la conexión. Por defecto, cuando se lanza la aplicación se abre un servidor en el puerto 19997 al que se puede conectar el cliente de forma inmediata. No obstante es posible abrir más conexiones mediante la API indicando otros puertos.

```
simRemoteApi.start(portNumber, maxPacketSize, debug, preEnableTrigger)
```

Disponer de un servidor continuamente abierto resulta extremadamente útil ya que se puede operar con la aplicación de forma completamente remota con solo abrirla.

Los programas externos deberán conocer la dirección IP de la máquina donde se está ejecutando CoppeliaSim. Una vez conectados, se les proporcionará el identificador de cliente que se deberá emplear para comunicarse con el servidor. Por ejemplo, para el caso de Matlab la conexión se inicia de la siguiente manera.

```
clientID=sim.simxStart('127.0.0.1',19997,true,true,5000,5);
```

Para poder utilizar la API remota será necesario añadir una serie de librerías a la carpeta de trabajo que se esté utilizando en el programa externo. Estas librerías se listan en la documentación oficial de la aplicación [10].

2.3.3 API remota BlueZero

BlueZero (BØ) es un middleware multi-plataforma desarrollado por los autores de CoppeliaSim que permite interconectar diferentes fragmentos de código que pueden ejecutarse en múltiples hilos, múltiples procesos o incluso en múltiples máquinas.

Se trata de una interfaz de comunicación que consiste en un bróker o gestor de paquetes, y distintos nodos y que soporta diferentes paradigmas de comunicación (cliente/servidor y publicador/suscriptor). De esta manera, CoppeliaSim puede actuar como uno de los diferentes nodos con los que otros nodos se pueden comunicar mediante servicios BØ, publicadores BØ o suscriptores BØ.

De forma similar a la API remota clásica, a la hora de trabajar con la API BlueZero es necesario solicitar un identificador de sesión al bróker. En este caso el identificador no es solo un número que se debe pasar a las funciones, si no que constituye una clase (estructura de programación) que incorpora multitud de métodos.

```
client=b0RemoteApi('b0RemoteApi_matlabClient','b0RemoteApi');
```

Cuando se invoca uno de los métodos disponibles es necesario indicar el *topic* o canal de comunicación que se debe emplear para ejecutar la llamada a la función deseada en CoppeliaSim. Los canales disponibles por defecto son los siguientes:

- *simxServiceCall*: Este canal permite realizar llamadas con espera de respuesta bloqueando la ejecución del código.
- *simxDefaultPublisher*: Este canal permite ejecutar una función sin esperar una respuesta.
- *simxDefaultSubscriber*: Este canal informa al servidor de que debe ejecutar una función y devolver el resultado de forma continua. No se realiza espera, cuando se recibe una respuesta se invoca a una función de tipo *callback* para que pueda ser leída.
- *simxCreatePublisher*: Este canal permite crear un nuevo canal independiente con la misma funcionalidad que *simxDefaultPublisher*.
- *simxCreateSubscriber*: Este canal permite crear un nuevo canal independiente con la misma funcionalidad que *simxDefaultSubscriber*.

También es posible realizar simulaciones de forma síncrona, para ello se deberá llamar de nuevo a la función *simxSynchronousTrigger* cuando se desee realizar el siguiente paso de simulación.

En último lugar, al igual que con la API remota clásica, el nodo Blue Zero podrá ser implementado en diversos lenguajes de programación. Será necesario añadir las librerías correspondientes [11] al directorio donde se encuentren los archivos de la aplicación externa.

2.3.4 Elementos de modelado

En el entorno de simulación de CoppeliaSim se pueden encontrar varios elementos, formas, articulaciones, sensores, *dummies*, cámaras, rutas, etc. A continuación, se van a presentar los elementos más relevantes para la realización de este trabajo: las formas, las articulaciones y los *dummies*. Mediante estos tres elementos es posible modelar el sistema mecánico de cualquier robot.

Existen distintos tipos de formas, las cuales se pueden clasificar en dos grupos principales, formas genéricas y formas puras.

- **Formas genéricas :** permiten representar cualquier tipo de geometría, pero su simulación tiene un coste computacional elevado. En esta categoría se incluyen modelos importados que se han generado de forma externa.

- **Formas puras :** solo admiten geometrías sencillas basadas en cubos, cilindros, esferas o planos, sin embargo, son mucho más eficientes para la simulación.

Las formas generadas o importadas en CoppeliaSim, por defecto, no guardan ninguna relación entre sí. Aunque a simple vista pueda parecer que los componentes de los modelos están conectados, en el explorador de objetos se comprueba que efectivamente no lo están ya que no hay ninguna jerarquía entre ellos.

Independientemente del tipo, las formas se pueden fusionar creando una única forma con propiedades homogéneas o agrupar formando un grupo (/) donde cada forma conserva sus propiedades. Si se mezclan formas puras y genéricas el resultado es siempre un grupo o forma genérica. Otra manera de conseguir uniones fijas es arrastrar un objeto dentro de otro, creando una jerarquía padre-hijo y manteniendo los sistemas de referencia de cada elemento por separado. En este último caso, si se realiza un movimiento del elemento padre el hijo también se moverá, pero no al contrario.

Para permitir ciertos grados de movimiento entre las formas, se pueden añadir, desde `Add`, `Joint`, tres tipos de pares cinemáticos o *joints*:

- **Pares cinemáticos de revolución** : Este tipo de articulaciones tienen un solo grado de libertad y permiten representar la cantidad de rotación sobre el eje Z del sistema de referencia principal. De forma adicional, se pueden obtener pares helicoidales si se indica un paso de hélice distinto de cero.

- **Pares cinemáticos prismáticos** : Las articulaciones prismáticas tienen un grado de libertad y permiten representar la cantidad de traslación sobre el eje Z del sistema de referencia principal.

- **Pares cinemáticos esféricos** : Los pares esféricos presentan 3 grados de libertad y permiten representar rotaciones en los tres ejes coordenados del sistema de referencia principal.

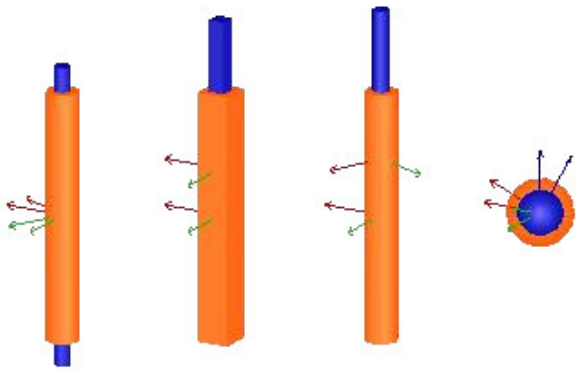


Figura 2.16: Articulaciones de revolución, prismática, helicoidal y esférica

En el entorno de simulación también incluye un tipo de elemento conocido como *dummy*  , que entre otras funciones, permite restringir movimientos entre distintos objetos. En CoppeliaSim solo es posible crear jerarquías con cadenas cinemáticas abiertas, pero gracias los *dummies* es posible cerrar dichas cadenas, indicando que dos puntos de la geometría deben ser coincidentes.

Para cerrar una cadena cinemática se deben crear dos *dummies* en los extremos que se desee unir y definir el tipo de enlace entre ellos.

- **Cinemática inversa (*IK, tip-target*)**, para simulaciones cinemáticas. Este enlace aparece representado con una línea roja punteada en la jerarquía de la escena (ver figura 3.14).

- **Dinámico (*Dynamics, overlap constraint*)**, para simulaciones dinámicas. Este enlace se muestra con una línea azul punteada en la jerarquía de la escena (ver figura 3.19).

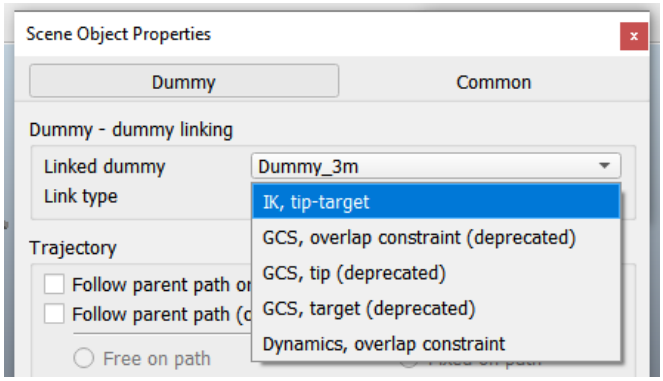


Figura 2.17: Selección del tipo de enlace entre *dummies*

2.3.5 Modos de simulación en CoppeliaSim

CoppeliaSim permite simular modelos de forma puramente cinemática o de forma dinámica, empleando uno de los cuatro motores de físicas que se incluyen. La decisión de realizar un tipo u otro de simulación implica una configuración específica para los objetos y pares cinemáticos, así como una jerarquía distinta para el ensamblado del modelo.

Los objetos del entorno de simulación se pueden clasificar en cuatro grupos dependiendo de sus propiedades dinámicas, tal como se muestra en la figura 2.18.

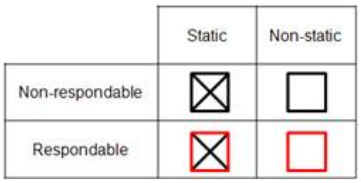


Figura 2.18: Clasificación de los objetos según sus propiedades dinámicas

Durante la simulación, los objetos estáticos (*static*) no están afectados por la gravedad y otras restricciones, mientras que los objetos no estáticos (*non-static*) o dinámicos sí que lo están e incluyen parámetros como la masa y los momentos de inercia principales. Por otro lado, los objetos reactivos (*respondable*) pueden producir una fuerza de reacción sobre otro objeto del mismo tipo, en cambio, los objetos no reactivos (*non-respondable*) no presentan este comportamiento. Existe además, una máscara de reactividad, que puede ser usada para limitar la interacción con determinados objetos. Las propiedades dinámicas de los objetos se pueden configurar seleccionando la opción *Show dynamic properties dialog* en la ventana de configuración.

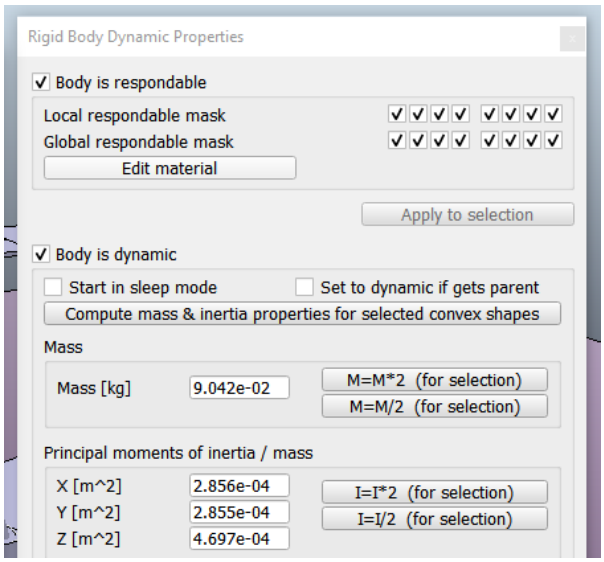


Figura 2.19: Propiedades dinámicas de un objeto

Los pares cinemáticos también ofrecen diversas opciones de configuración. En primer lugar, se puede especificar el rango de movimiento y la posición inicial de la articulación. Por otro lado, se puede seleccionar uno de los cuatro modos de operación disponibles.

- **Modo pasivo**: En este modo el valor de la articulación no se calcula automáticamente pero puede ser cambiado por el usuario.

- **Modo de cinemática inversa**: En este modo el valor de la articulación se calcula a partir de las posiciones y orientaciones de los objetos con los que está enlazada.

- **Modo dependiente**: En este modo el valor de la articulación está vinculado al valor de otra articulación mediante una ecuación de dependencia.

- **Modo par/fuerza**: En este modo, si está habilitada de forma dinámica, la articulación se simula mediante uno de los motores de físicas que se incluyen en el programa.

También es posible realizar simulaciones dinámicas si se activa la operación híbrida de la articulación en los modos pasivo, cinemática inversa o dependiente.

Para que una articulación esté habilitada de forma dinámica, esta debe cumplir una serie de requisitos. En primer lugar debe estar configurada en modo par/fuerza o híbrido. Además debe tener un objeto padre y un único objeto hijo que debe ser dinámico. Por último, si la articulación forma parte de una cadena cinemática cerrada, la relación de los *dummies* que la cierran debe ser de tipo dinámico (*dynamics, overlap constraint*). Estos dos últimos requisitos definen como debe ser la jerarquía de ensamblado del modelo.

Sí la articulación está configurada en modo dinámico es posible especificar también sus propiedades dinámicas desde la opción *Show dynamic properties dialog*, al igual que para los objetos. Cabe destacar que las propiedades estarán habilitadas o no dependiendo del modo de operación elegido.

La primera opción que se puede configurar es si la articulación tiene un motor activo o no. En caso afirmativo es posible especificar el par/fuerza máximo que puede proporcionar y la velocidad objetivo. De esta forma se puede realizar un control de velocidad. La segunda opción permite activar el control en bucle cerrado para realizar un control de posición mediante PID o para que la articulación actúe como un muelle-amortiguador.

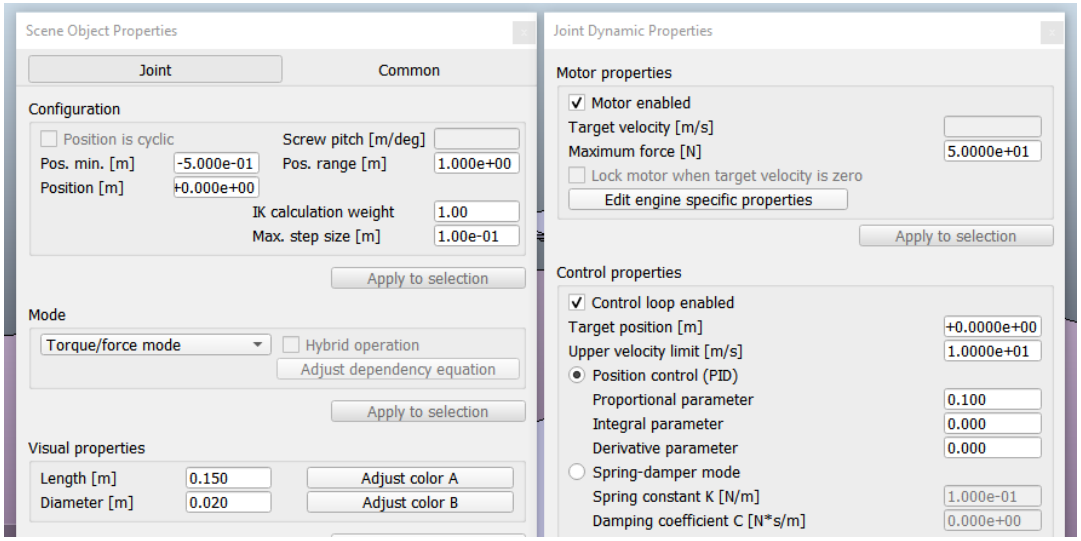


Figura 2.20: Ventana de configuración (izquierda), propiedades dinámicas (derecha)

En CoppeliaSim no es posible realizar un control por par/fuerza de forma directa, por lo que, si se necesita este tipo de control, se debe recurrir al siguiente método. En primer lugar, la articulación debe tener el motor activo y el control de posición desactivado. Seguidamente se debe especificar una velocidad objetivo muy elevada, por ejemplo $10000 m/s$. Al iniciarse la simulación, la aplicación tratará de llevar las articulaciones a la velocidad deseada saturando en la fuerza máxima especificada, variando dicha fuerza es posible realizar el control deseado. Cabe destacar que el sentido del movimiento de la articulación vendrá dado por el signo de la velocidad, mientras que la fuerza se deberá indicar siempre en valor absoluto.

De esta forma, para la realización de una simulación puramente cinemática, los objetos deben ser estáticos y no reactivos, las articulaciones deben operar en modo pasivo, cinemática inversa o dependiente, y los posibles *dummies* deben tener un enlace de tipo cinemática inversa. Por otro lado, si se desea realizar una simulación dinámica, los objetos (al menos los que no son fijos) deben ser dinámicos, las articulaciones deben estar habilitadas dinámicamente y el enlace de los posibles *dummies* debe de ser de tipo dinámico. Durante la simulación aparecerá el icono al lado de los objetos dinámicos, está es una manera sencilla de comprobar que la configuración del modelo es correcta.

Para gestionar la simulación se dispone de varias opciones en los menús de la ventana principal. Mediante ellos es posible seleccionar, en primer lugar, el motor de físicas que se desea emplear, la precisión de la simulación y el tiempo entre cada paso de simulación. Seguidamente se dispone de los botones de inicio, pausa y finalización de la simulación y la opción de operar en tiempo real. Finalmente, para acelerar la simulación se puede seleccionar cada cuantos pasos de simulación se actualiza la ventana gráfica o desactivarla completamente para una velocidad aun mayor.

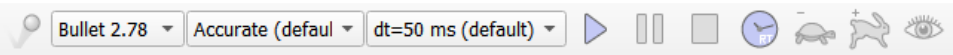


Figura 2.21: Opciones de simulación

Las opciones de simulación se pueden configurar de forma más detallada desde las ventanas *Calculation Modules* y *Simulation Settings* que son accesibles pulsando sobre y respectiamente o desde las opciones *Simulation* y *Tools* del la ventana principal.

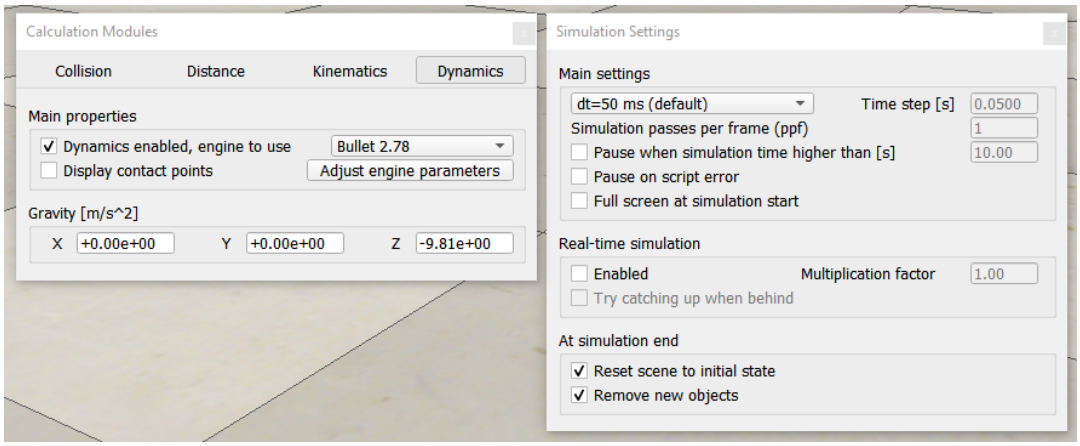


Figura 2.22: Opciones de simulación ampliadas

Capítulo 3

Desarrollo práctico

3.1 Incorporación del robot a CoppeliaSim

3.1.1 Importación del modelo

Para tener una representación visual del robot se va a partir de un modelo creado y ensamblado en SolidWorks por el doctorando Pau Zamora. Es importante disponer del ensamblaje del robot ya que así se definen las posiciones y orientaciones iniciales de cada componente, aunque posteriormente estos puedan ser animados de forma independiente. El montaje también se podría haber hecho con CoppeliaSim, no obstante, SolidWorks dispone de herramientas más potentes que permiten realizar este proceso más fácilmente.

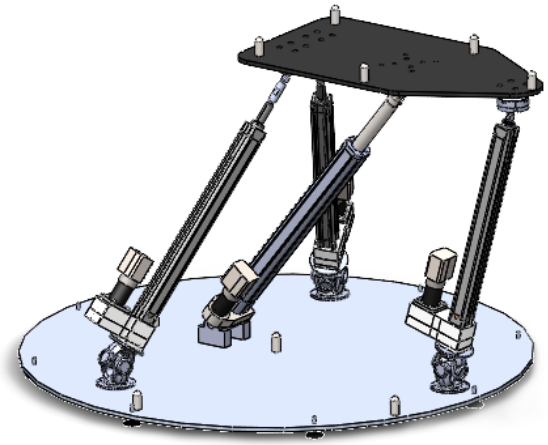


Figura 3.1: Ensamblaje del robot dentro de SolidWorks

De esta manera, se guarda el modelo de SolidWorks como una colección de archivos en formato `STL` que puedan ser leídos por CoppeliaSim. Cada archivo contendrá un único componente del modelo manteniendo las relaciones geométricas originales.

Los archivos generados se importan a CoppeliaSim desde el menú *`File, Import, Mesh...`*. Es posible seleccionar todos los archivos de forma conjunta puesto que el ensamblado ya se ha realizado previamente. Durante el proceso de importación se debe indicar que la dirección vertical del modelo se corresponde con el eje Y, igual que en SolidWorks. El resultado es un modelo correctamente ensamblado pero con apariencia por defecto y cuyos componentes han sido renombrados con un nombre genérico (ver figura 3.2).

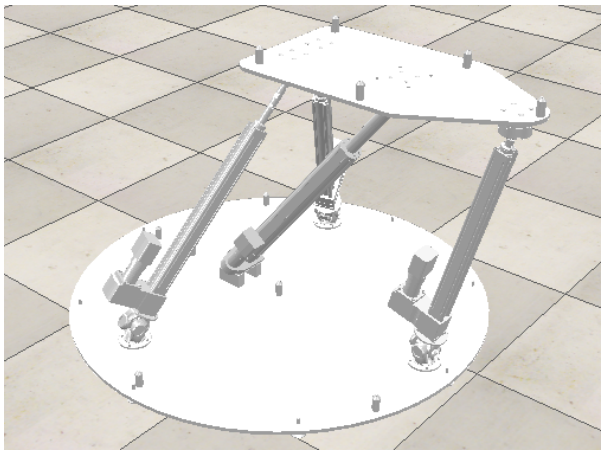


Figura 3.2: Importación inicial del robot 3UPS-RPU en CoppeliaSim

A continuación, se realizan una serie de modificaciones que, si bien no tienen un efecto directo en la simulación, permiten organizar el espacio de trabajo y hacer que el robot tenga un aspecto más realista. En primer lugar, se renombran y se agrupan de forma preliminar los elementos asociados a los distintos eslabones. La jerarquía final del modelo se examinará más adelante en este documento. Por otro lado, se desplaza el robot hasta hacer coincidir el centro de la plataforma fija con el origen de coordenadas global, se orienta igual que en el diseño teórico y se aplica color a los componentes.

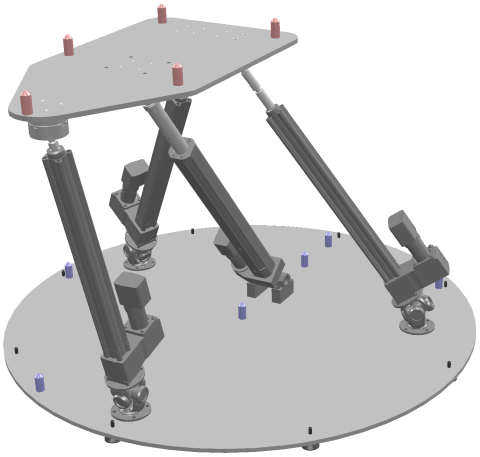


Figura 3.3: Ensamblaje del robot 3UPS-RPU dentro de CoppeliaSim

3.1.2 Creación del modelo simplificado

Hasta ahora se ha trabajado con formas genéricas, que han sido el resultado de la importación de los archivos `STL`, pero de cara a poder realizar las simulaciones de forma más rápida, se va a crear un modelo simplificado del robot. Este actuará como un esqueleto sobre el cual se colocarán las distintas geometrías importadas para que el robot tenga una apariencia más realista, pero que no intervendrán en los cálculos de los eslabones del robot.

Existen diferentes alternativas para generar el modelo simplificado, por ejemplo crear formas simples de manualmente teniendo en cuenta las dimensiones de cada componente del robot. No obstante, se va a optar por la generación automática de las formas simples con la herramienta de edición de formas que incorpora CoppeliaSim. Este método es mucho más rápido ya que no requiere orientar y posicionar las formas creadas para que coincidan con los componentes originales. Para generar una forma simple, se selecciona una de las piezas del robot y se pulsa sobre  abriéndose el editor de formas.

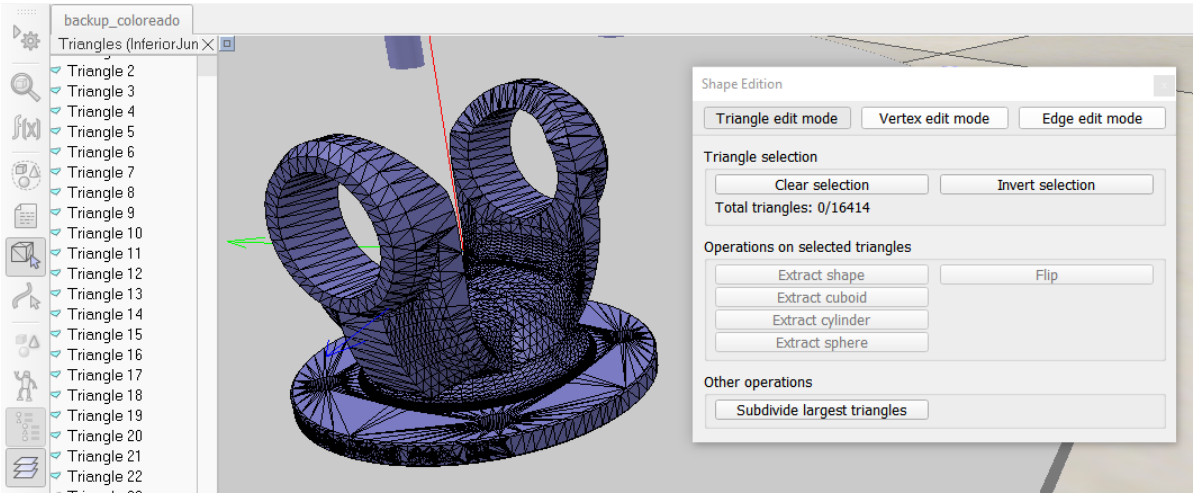


Figura 3.4: Herramienta de edición de formas

En la figura 3.4 se observa que la pieza elegida aparece representada por una malla de triángulos, estos podrán ser seleccionados para obtener una forma simple acorde a la geometría desde la opción *Triangle edit mode*. Si se desea seleccionar la pieza por completo, el método más rápido consiste en invertir la selección original. Una vez seleccionados, los elementos se mostrarán un color diferente, tal como se observa en la figura 3.5.

Es posible seleccionar solo una parte de todos los triángulos disponibles o en su lugar, seleccionar los vértices y aristas que los forman con las opciones de *Vertex edit mode* y *Edge edit mode* respectivamente. Será conveniente utilizar el editor de vértices cuando se desee hacer una conversión parcial del elemento, ya que permite una selección más precisa. Por ejemplo, para el caso de la parte inferior de las patas, es posible orientar la pieza de tal forma que se pueda realizar una selección únicamente de la parte principal, generando de esta manera un cilindro simple.

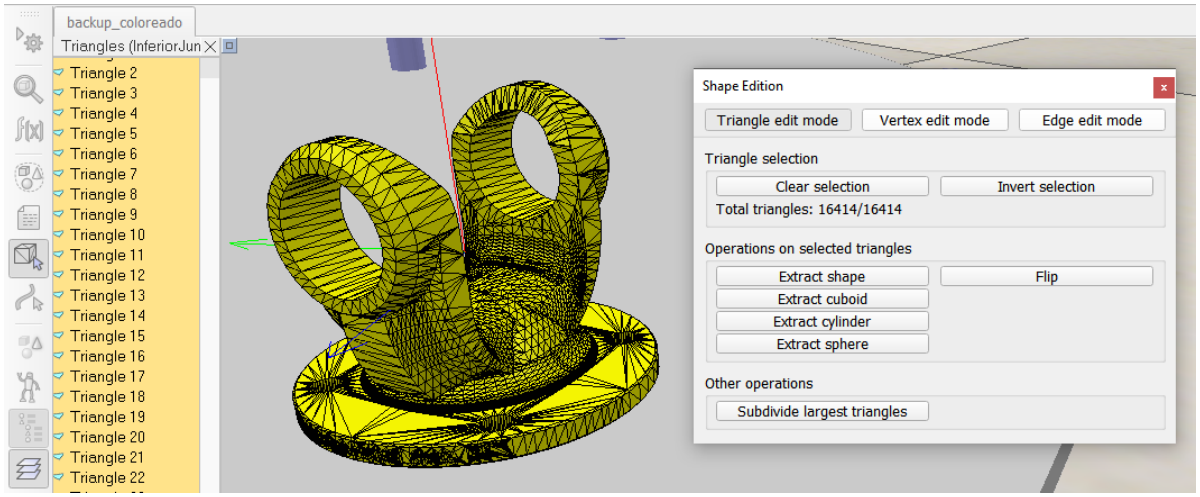


Figura 3.5: Selección de contornos

Una vez seleccionados los elementos es posible elegir el tipo de forma que se desea extraer. Para la extracción de formas puras se deberá optar por la geometría que mejor se aproxime a la pieza original: cubo, cilindro o esfera.

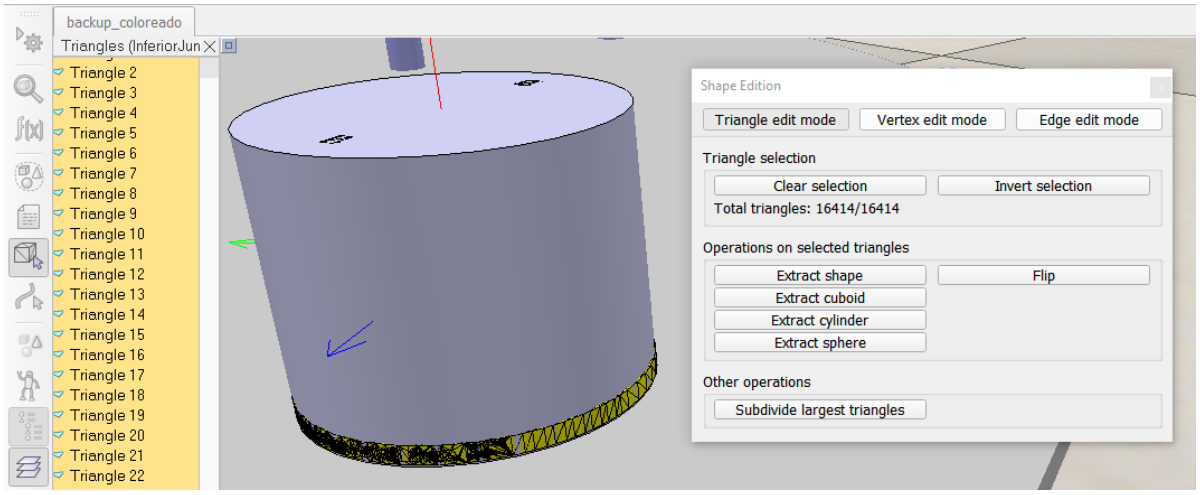


Figura 3.6: Resultado de la generación de formas

Si la extracción se ha realizado de forma correcta, la nueva forma pura generada tendrá la misma posición y orientación que la pieza original y unas dimensiones similares, tal como se muestra en la figura 3.6. Ambos componentes aparecen solapados, por lo que se deberá configurar la visibilidad de la forma pura para evitar que se muestre.

CoppeliaSim dispone dos vistas, una principal que se muestra por defecto y una invertida, que se pueden configurar pulsando sobre el icono de capas de visibilidad . Cada vista presenta distintas capas que es posible activar o desactivar y que se corresponden a los distintos elementos del entorno de simulación, por ejemplo las formas, las articulaciones o el suelo.

Por otro lado, dentro de la configuración de cada elemento se puede seleccionar en que capas será visible dicho elemento. El menú de configuración es accesible haciendo doble-click sobre el icono del elemento dentro del explorador.

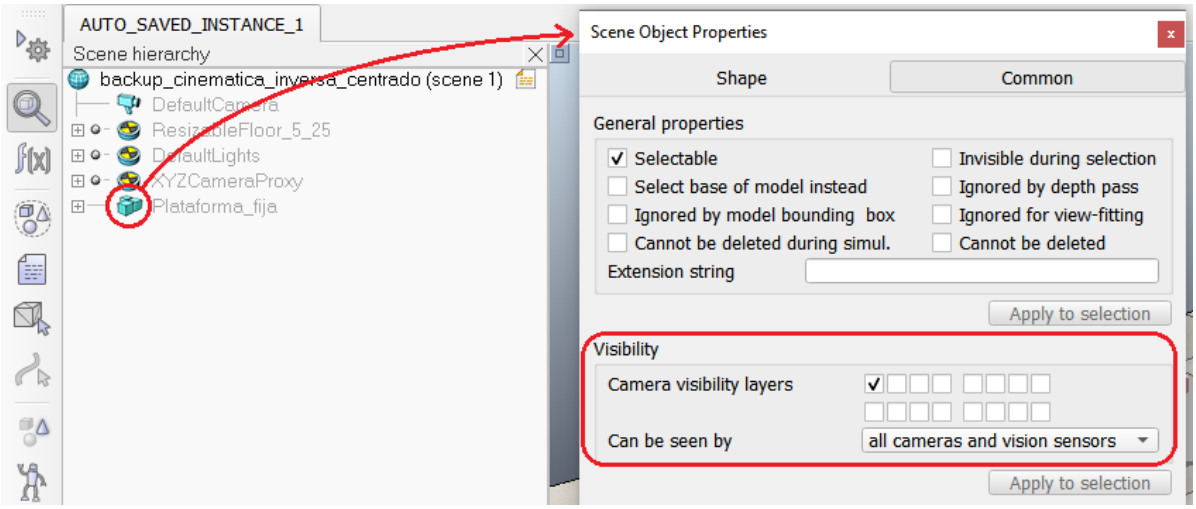


Figura 3.7: Propiedades de objetos

Teniendo esto en cuenta, todas las formas puras que componen el esqueleto del robot se colocan en la primera capa de la vista secundaria, mientras que las piezas importadas se colocan en la misma capa de de la vista principal.

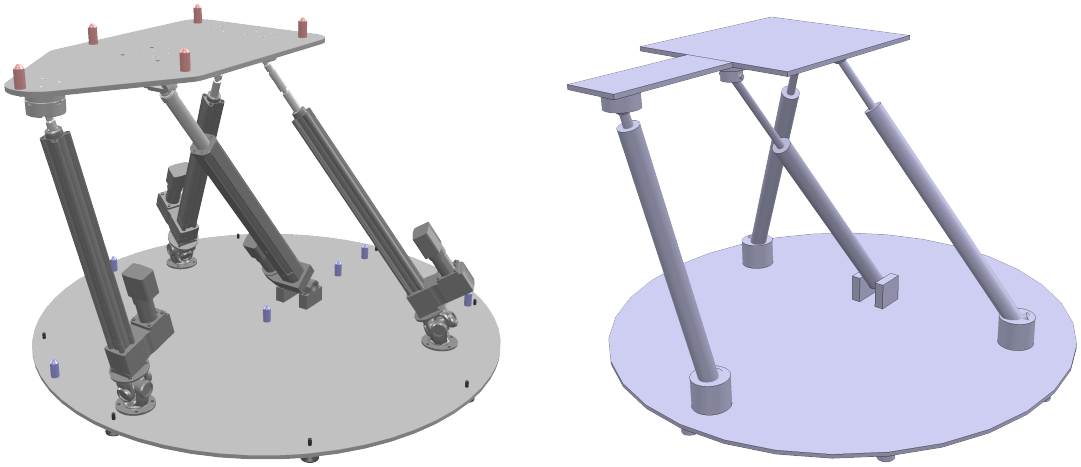


Figura 3.8: Modelo original (izquierda) y esqueleto (derecha)

Como resultado final se obtienen dos modelos con geometrías idénticas pero en dos vistas diferentes, tal como se muestra en la figura 3.8. El modelo importado y el modelo generado dentro de CoppeliaSim tendrán que ser configurados de manera distinta atendiendo a su función dentro de la simulación. Las propiedades y configuraciones de los componentes se abordarán de forma más detallada en posteriores apartados.

3.1.3 Ensamblado del modelo cinemático

En este apartado se va desarrollar el modelo en configuración cinemática. Este modelo tendrá implementado dos modos de funcionamiento: modo de cinemática directa, en el cual la posición y orientación de la plataforma superior del robot vendrán determinadas por las entradas proporcionadas a las articulaciones activas, y modo de cinemática inversa, en el cual la extensión de los actuadores se calculará de forma automática a partir de la posición y orientación de la plataforma móvil. Hay que destacar que en el contexto de este trabajo la resolución de la cinemática directa con CoppeliaSim tiene una aplicación práctica como se verá en la sección 3.2, mientras que el problema cinemático inverso solo se resuelve para demostrar la versatilidad del programa.

Se comienza creando la jerarquía del modelo, agrupando los elementos necesarios e incorporando los distintos tipos de pares cinemáticos. Los componentes del modelo original se colocan como hijos de los correspondientes componentes del esqueleto y se nombran con su mismo nombre más el sufijo "mod" (modelo). Todos los objetos que componen el modelo se configuran como estáticos y no reactivos.

En lo referente a los pares cinemáticos, sus ejes de movimiento deben coincidir con los ejes de las articulaciones reales del robot. Esto se consigue seleccionando primero el par cinemático que se desea ajustar y luego una forma pura cuyo centro geométrico coincida con la posición deseada. A continuación se pueden utilizar las herramientas de traslación y rotación para hacer coincidir la posición y orientación de la articulación mediante la opción *Apply to selection*.

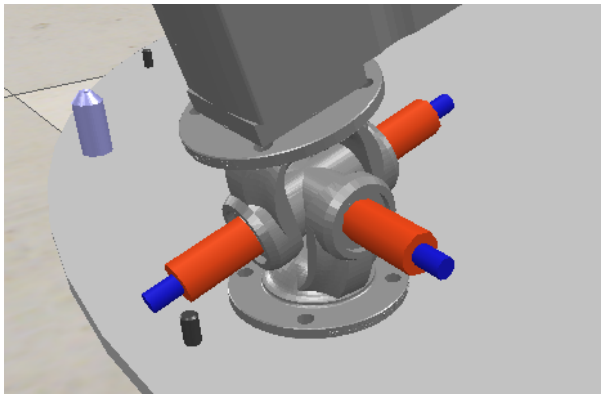


Figura 3.9: Ejemplo de alineación de los pares cinemáticos

Cabe destacar que en general la orientación de los pares cinemáticos debe coincidir en dirección pero no necesariamente en sentido excepto para la articulación q42, cuyo eje Z debe coincidir con el eje Z de los objetos que conecta, y las articulaciones q13, q23 y q33, cuyos ejes Z se deberán orientar justo al contrario. Esto es debido al sentido en el que se cerrarán la cadenas cinemáticas, de esta forma valores positivos para estas últimas articulaciones resultarán en la "extensión" su actuador correspondiente.

A continuación, se debe especificar el rango y posición inicial de los actuadores desde el menú de configuración. El rango se obtiene a partir del catálogo del fabricante y corresponde a una longitud de 400 mm. La posición inicial determina restando la longitud de cada actuador en su estado de mínima extensión a la longitud que presenta en el modelo importado. Las medidas necesarias se toman desde SolidWorks.

Es necesario realizar todas las operaciones de traslación y rotación, y el ajuste de los rangos y posiciones antes de crear los vínculos padre-hijo, ya que de lo contrario los elementos hijo se moverían junto al elemento padre.

Los pares cinemáticos activos, correspondientes a los actuadores lineales de las patas del robot, se configuran en modo pasivo, para que puedan ser controlados de forma externa. El resto de pares cinemáticos se configuran en modo cinemática inversa.

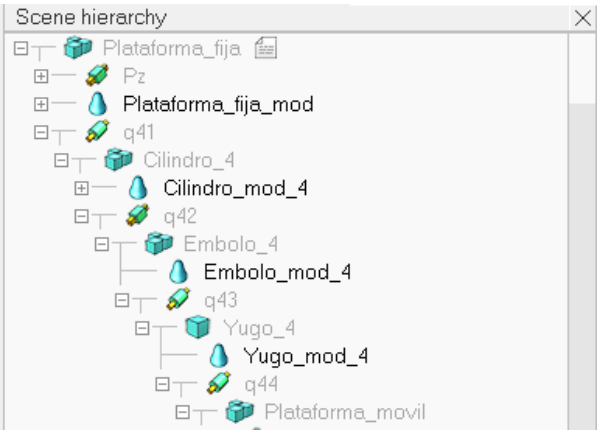


Figura 3.10: Ejemplo de la jerarquía de los componentes del modelo

El siguiente paso es cerrar las cadenas cinemáticas tal como se muestra en la figura 3.11. Se emplean tres parejas de *dummies*, colocando un *dummy* de cada pareja en la base del robot y otro en el extremo de cada pata exterior y uniéndolos mediante un enlace de cinemática inversa. Los *dummies* enlazados deben tener la misma posición y orientación.

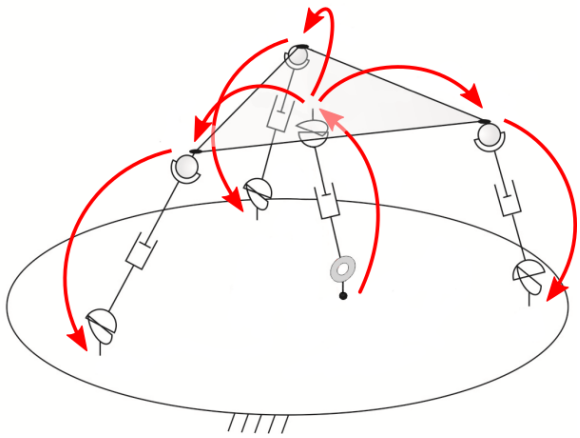


Figura 3.11: Cierre de las cadenas cinemáticas del modelo cinemático

Después de conectar los *dummies* es necesario crear un grupo de cinemática inversa para indicar al programa que debe ajustar los valores de las articulaciones de tal forma que los dos *dummies* de cada pareja se solapen. Esta técnica permitirá obtener el efecto de una cadena cinemática cerrada. El grupo se crea desde ***Tools***, ***Calculation module properties*** y luego, desde la opción ***Edit IK elements*** se añaden los *dummies* ubicados en los extremos de las patas. Como objetivo (***Target***) de cada *dummy* se indica su correspondiente pareja fija.

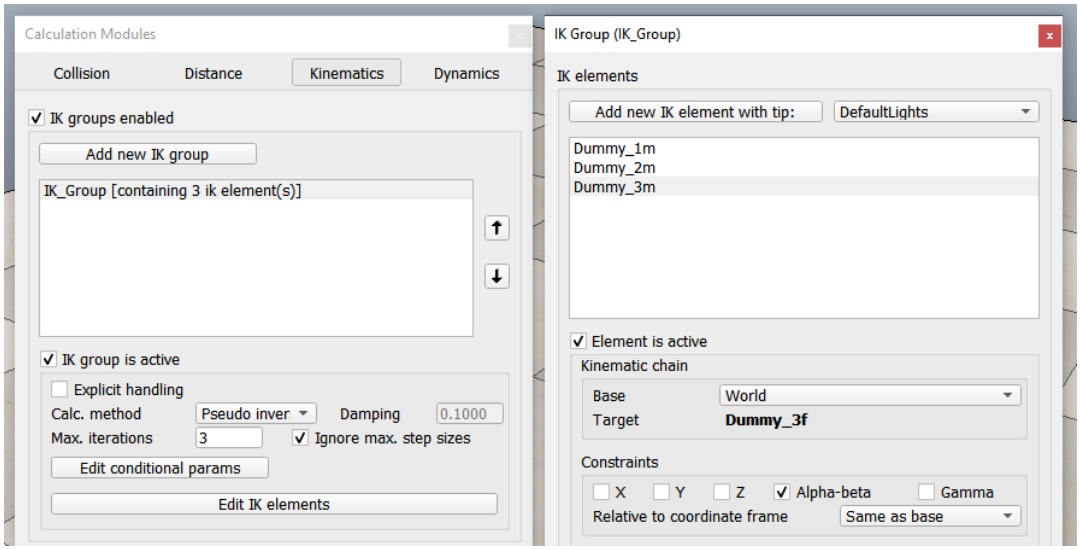


Figura 3.12: Configuración del módulo cinemático de CoppeliaSim

Con el cierre de las cadenas cinemáticas queda resuelto el problema cinemático directo. En este momento es posible proporcionar un valor a las articulaciones activas del robot para moverlo a una nueva posición, tal como se ilustra en la figura 3.13.

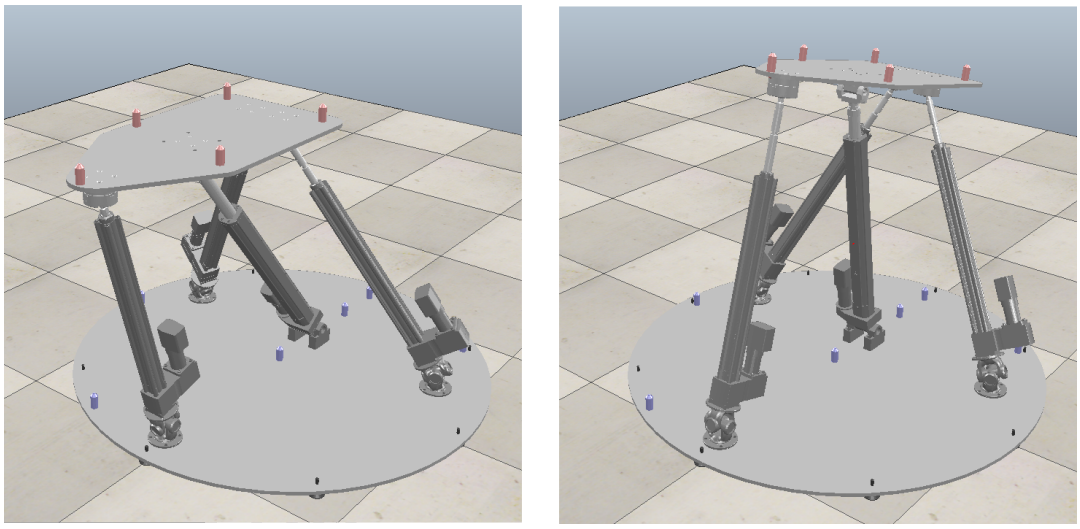


Figura 3.13: Movimiento del robot en modo cinemática directa

Para resolver el problema cinemático inverso es necesario realizar unos ajustes adicionales. Como primer paso se añaden cuatro pares cinemáticos en modo pasivo, dos de traslación (ejes X y Z) y dos de rotación (ejes Y y Z) los cuales representarán los cuatro grados de libertad del robot. Seguidamente, se añade una pareja de *dummies*, uno ubicado en la plataforma móvil, que hará de seguidor, y otro como hijo de los pares cinemáticos anteriores, que hará de referencia. Estos *dummies* se conectan con un enlace de tipo dinámico y se añaden al grupo de cinemática inversa creado anteriormente. Por último, se configuran las articulaciones pertenecientes al robot en modo cinemática inversa para que su valor se calcule automáticamente al indicar la posición y rotación de la plataforma. La jerarquía final se muestra en la figura 3.14, donde las lineas rojas representan los vínculos entre los *dummies* (consultar subsección 2.3.4).

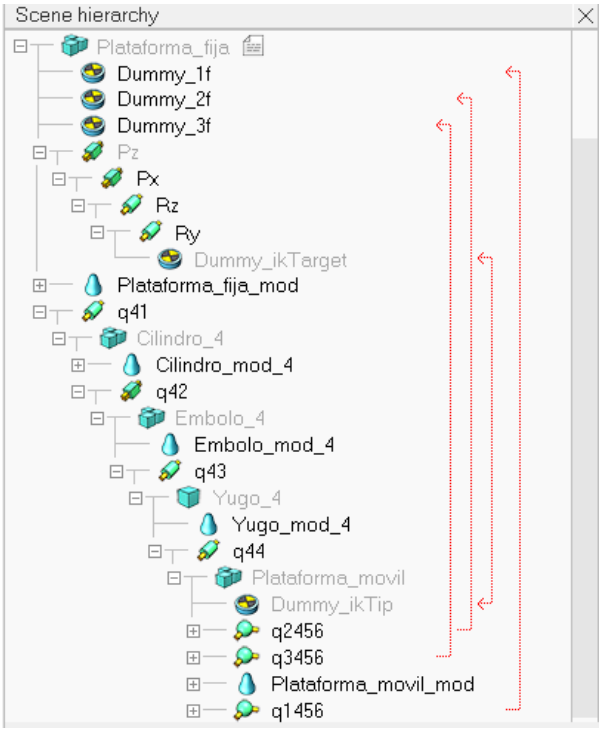


Figura 3.14: Jerarquía del modelo cinemático

La incorporación del modo de cinemática inversa junto al existente modo de cinemática directa origina configuraciones que son incompatibles entre sí, como por ejemplo el modo de operación de las articulaciones activas del robot. En un principio, estas configuraciones deberán ajustarse manualmente antes de lanzar la simulación, pero afortunadamente el reajuste del modelo se puede automatizar por medio de un script.

Se emplea un script (ver Apéndice B) de tipo *non threaded* asociado a la base del robot en el cual se incluyen dos funciones para seleccionar el modo de operación. En un primer lugar, se crea la función *setFkMode* permite configurar el robot en modo cinemática directa. En esta función se desactiva el vínculo entre los *dummies* seguidor y referencia, que se crearon para la cinemática inversa, y establece las articulaciones activas del robot en modo pasivo.

Por otro lado, se implementa la función *setIkMode* que activará el modo de cinemática inversa. En este caso se activa el enlace de los *dummies* y se establecen las restricciones correspondientes a los pares cinemáticos auxiliares (traslación los ejes X y Z y rotación en los ejes Y y Z). Seguidamente se configuran las articulaciones activas en modo de cinemática inversa.

De esta forma concluye la creación del modelo para simulaciones cinemáticas en CoppeliaSim. Con un mismo archivo es posible resolver los problemas de cinemática directa e inversa invocando las funciones *setFkMode* y *setIkMode* respectivamente al comienzo o incluso durante la simulación. De esta forma, se consigue un resultado similar al de programas comerciales como RobotStudio de ABB Robotics, posibilitando la creación de diversas aplicaciones, como por ejemplo la programación de trayectorias sin necesidad de código. Un ejemplo de las simulaciones de cinemática directa e inversa se puede consultar en el siguiente enlace:

`https://upvedues-my.sharepoint.com/:f:/g/personal/dane2_upv_edu_es/EmkFiOHWa-ZLlXovsAUT3z0BoZ8-_6hDKHR-sWDTXHqvpg?e=ny9iUj`

3.1.4 Ensamblado del modelo dinámico

En esta sección se va crear el modelo en configuración dinámica, abordándose en concreto el problema dinámico directo, cuya aplicación se estudiará en la sección 3.3. Se parte del modelo cinemático, al que además de añadir las propiedades dinámicas de los componentes, se le deberán realizar ciertas modificaciones en su jerarquía.

Las formas genéricas se dejan en su configuración por defecto, ya que de nuevo solo tienen una función estética y seguirán los movimientos del esqueleto compuesto por formas puras.

Por su parte, en las formas puras que constituyen el esqueleto del robot se deben configurar las propiedades dinámicas. La masa, la ubicación del centro de masas y los momentos de inercia de los componentes más relevantes del robot, ya se han determinado previamente mediante con respecto a los sistemas de referencia locales establecidos en el modelado dinámico (subsección 2.2.3 y subsección 2.2.4) y se recopilan en el Apéndice A.

Sin embargo, las medidas de los parámetros físicos del robot, a excepción de la masa, son relativos a sistemas de referencia diferentes de los que se dispone en CoppeliaSim, estando estos últimos ubicados en los centros geométricos de cada pieza (ver figura 3.15). Además, la aplicación solo permite introducir el tensor de momentos de inercia respecto al sistema de referencia absoluto (mundo). Si se desea introducir respecto al sistema de referencia relativo es necesario obtener las componentes principales de inercia y reorientar los ejes.

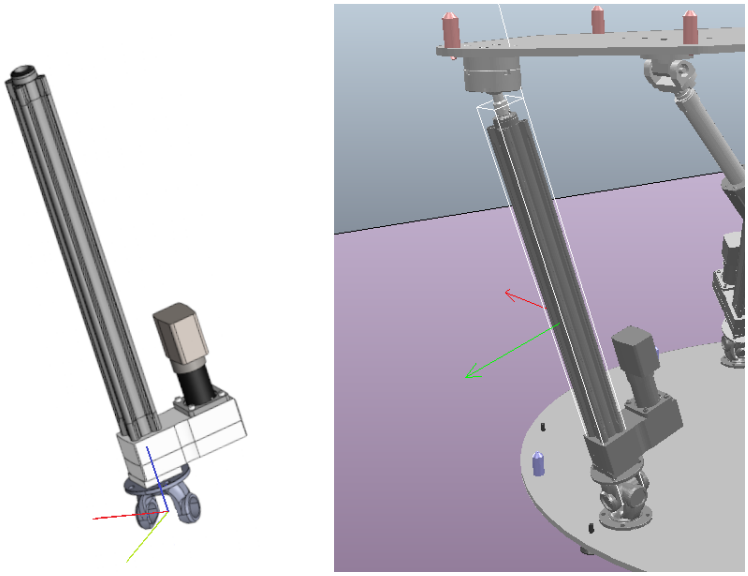


Figura 3.15: Sistemas de referencia original (izquierda) y en CoppeliaSim (derecha)

Dado que las funcionalidades de manipulación de sistemas de referencia son bastante limitadas en CoppeliaSim, se propone una estrategia que permite poder trabajar con los datos originales y además limita las transformaciones que se deben aplicar al modelo.

- **Paso 1.** Se guarda como una copia el archivo de CoppeliaSim.
- **Paso 2.** Se aíslan el componente del cual se quieren configurar las propiedades dinámicas y un punto del modelo que represente el origen del sistema de referencia original. Este punto puede ser por ejemplo, el centro geométrico de una articulación.

- **Paso 3.** Se orientan los ejes coordenados del componente para que coincidan con los ejes coordenados originales.

- **Paso 4.** Empleando el terminal de LUA integrado se obtiene el desplazamiento del origen de coordenadas original respecto al centro geométrico del componente, que se representa por el vector $\vec{r}$. Los valores muy pequeños se sustituyen por cero. Por ejemplo, para la pata 1 junto con la articulación q12:

```
> q = sim.getObjectHandle('q12')
> c = sim.getObjectHandle('Cilindro_1')
> r = sim.getObjectPosition(q,c)
```

- **Paso 5.** Se mueve el componente haciendo coincidir en posición y orientación su sistema de referencia con el sistema de referencia global.

- **Paso 6.** Se desplaza el componente según $-\vec{r}$. En este momento el sistema de referencia global coincide con el sistema de referencia original desde el punto de vista del componente.

- **Paso 7.** Se introducen las propiedades dinámicas, relativas al sistema de referencia global en CoppeliaSim, siendo únicamente necesario dividir el tensor de inercias entre la masa.

- **Paso 8.** Se copian los parámetros relativos calculados al componente original.

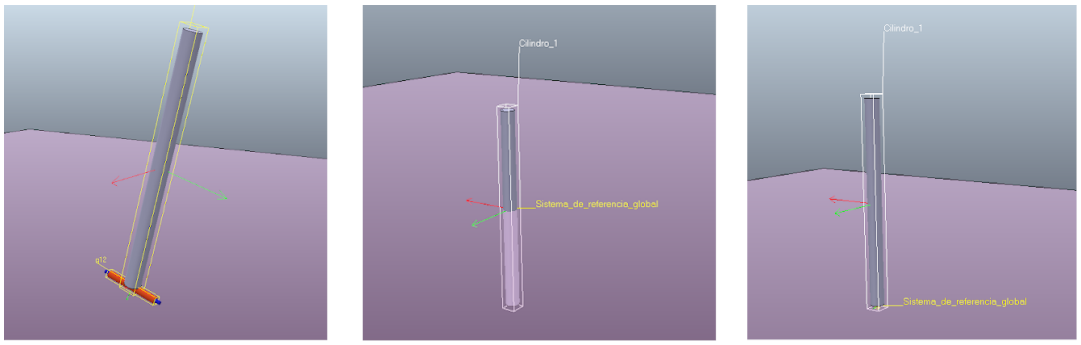


Figura 3.16: Traslación del componente (pasos 2, 3, 4, 5 y 6)

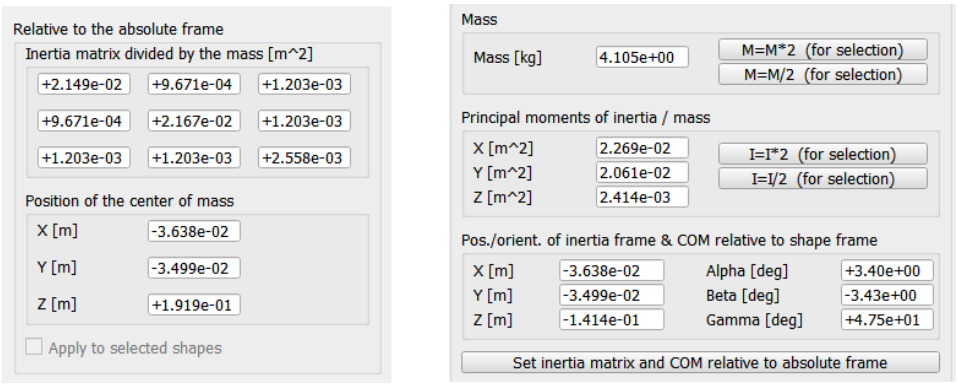


Figura 3.17: Configuración de las inercias (pasos 7 y 8)

Este procedimiento se repite desde el segundo paso para todos los componentes del robot excepto para las crucetas de las juntas universales de las patas exteriores y para el yugo de la pata central, que no se han considerado en el modelo dinámico original.

No obstante, según la documentación de CoppeliaSim, los cuerpos que intervienen en una simulación dinámica deben tener unas propiedades físicas de una magnitud similar, es decir, no es posible indicar un peso nulo para los componentes menos relevantes. De no ser así, se producirían errores en el cálculo de las posiciones de estos componentes, afectando finalmente a la geometría del robot. Por este motivo se establece un peso orientativo de 300 gramos para estos componentes. Los momentos de inercia se calculan a partir de la geometría y del peso cada pieza de forma automática.

Esto puede originar ciertas discrepancias con el modelo dinámico original, pero supone una aproximación más realista al modelo real del robot. En cualquier caso, las propiedades dinámicas de las crucetas y del yugo no tienen un gran efecto sobre en la respuesta dinámica del robot ya que, especialmente en el caso de las crucetas, se trata de componentes que experimentan un movimiento limitado a velocidades bajas.

A continuación, se deben modificar las propiedades de las uniones móviles del modelo. Las articulaciones dependientes, se configuran en modo cinemática inversa activando la casilla de operación híbrida ya que se va a trabajar con un modelo dinámico. En el caso de las articulaciones activas se configuran en modo par/fuerza alineando sus ejes Z en dirección y sentido con los ejes Z de las formas que unen.

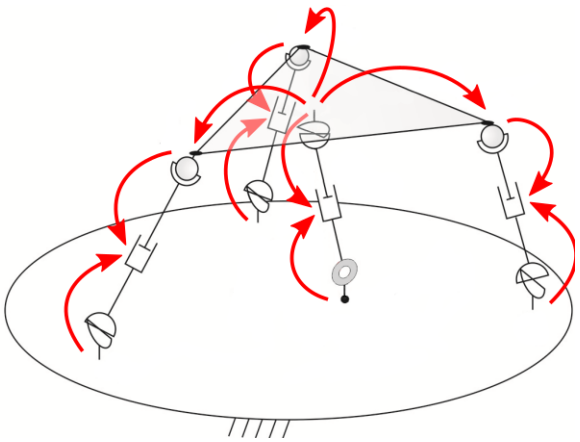


Figura 3.18: Cierre de las cadenas cinemáticas del modelo dinámico

Seguidamente, se cierran las cadenas cinemáticas atendiendo a los requisitos necesarios para realizar una simulación dinámica que se han presentado en la subsección 2.3.5. El cierre se realiza en cada una de las patas del robot tal como indica en la figura 3.11, añadiéndose un *dummy* en el cilindro y su correspondiente pareja en el émbolo. Los *dummies* de cada grupo se vinculan mediante un enlace de tipo dinámico. Con este último paso se completa el modelo dinámico del robot, cuya jerarquía final se muestra en la figura 3.19. En dicha figura se observan los vínculos dinámicos de los *dummies* representados con una línea azul (ver subsección 2.3.4).

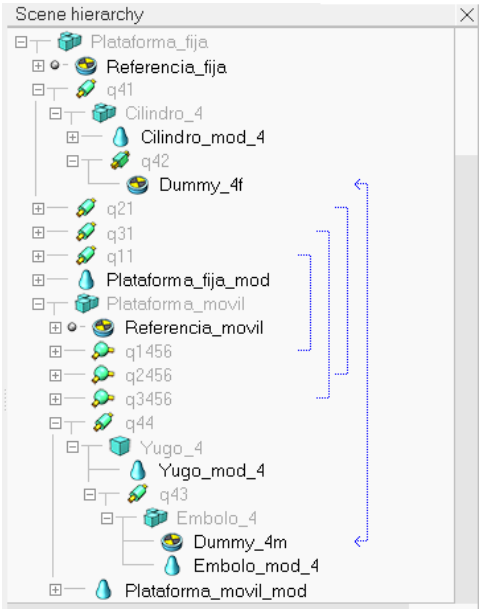


Figura 3.19: Jerarquía del modelo dinámico

A partir de este momento es posible ejecutar la simulación y verificar que el ensamblaje ha sido correcto. Como en este caso se tiene un modelo dinámico, se empleará uno de los motores físicos disponibles en CoppeliaSim, en concreto se utilizará Bullet 2.78, que es el que ha permitido obtener una mejor respuesta.

Se comprueba que aunque se alcancen las posiciones deseadas, aparecen pequeños artefactos visuales si se pretende que el robot realice grandes movimientos de forma abrupta empleando los reguladores PID por defecto de CoppeliaSim. Afortunadamente, este no es el objetivo del robot real y además en la simulación final se emplearán trayectorias suaves y reguladores mejor sintonizados, por lo que se podrán obtener resultados más realistas.

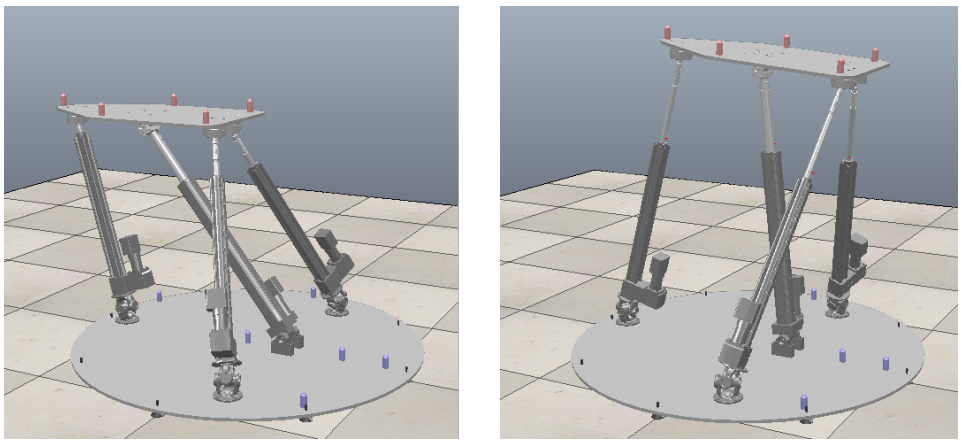


Figura 3.20: Movimiento del robot en modo dinámico

De esta manera se finaliza la creación del modelo para simulaciones dinámicas en CoppeliaSim. Cabe destacar que en este apartado no se ha implementado el control por par/fuerza, esta tarea se realizará posteriormente directamente desde Matlab empleando la API remota.

3.2 Aplicación 1: reproducción de movimientos

El objetivo de esta primera aplicación es conseguir que el modelo cinemático implementado en CoppeliaSim reproduzca los movimientos del prototipo en tiempo real. El control del robot real se realiza mediante un único ordenador que utiliza ROS2, un *middleware* que permite crear una red de nodos que pueden publicar y suscribirse a diversos *topic* o canales para compartir información. La estructura de control empleada se muestra en la figura 3.21, donde los elementos elípticos corresponden a los nodos y los elementos rectangulares corresponden a los *topics*.

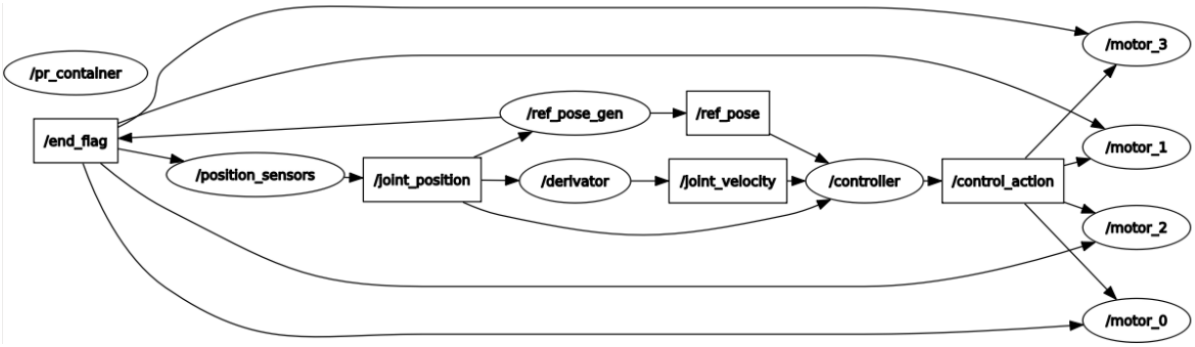


Figura 3.21: Diagrama de nodos ROS

Aunque en CoppeliaSim se puede trabajar directamente con ROS2, solo soporta la versión ROS2 *Foxy*, mientras que la usada en el control es ROS2 *Eloquent*, por tanto se ha optado por el uso de *sockets* TCP/IP como una alternativa más universal y sencilla de implementar. En concreto se va a programar un cliente que se ejecutará en un ordenador externo. Este ordenador también incorpora un nodo ROS2 suscrito a todos los *topics* y se encargará de redirigir la información al cliente, el cual finalmente la enviará a CoppeliaSim (ver figura 3.22).

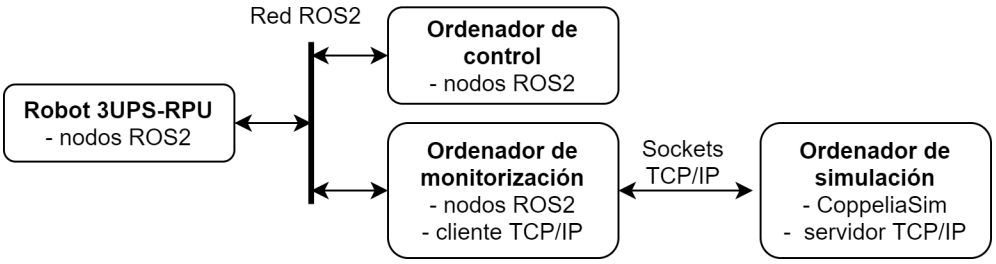


Figura 3.22: Estructura de control distribuido y simulación

3.2.1 Implementación del cliente TCP/IP

El cliente se desarrolla en lenguaje C y podrá ser compilado para múltiples sistemas operativos, ya que también se dispone del código fuente de las librerías de CoppeliaSim necesarias para su implementación. En este caso concreto se compilará para el sistema operativo Ubuntu 18.04.

Se implementan tres funciones, cuyo código se puede consultar en el Apéndice B. La función *start_ coppelia_ client* establece una conexión con el servidor, el cual deberá estar ejecutándose previamente (esto se consigue al iniciar CoppeliaSim como se describe en la subsección 2.3.2).

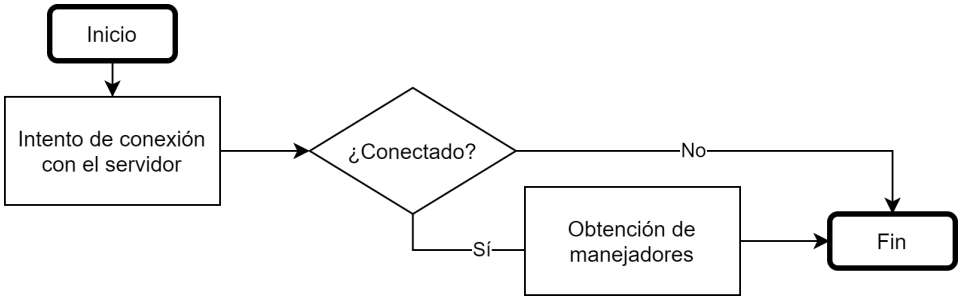


Figura 3.23: Diagrama de flujo *start_ coppelia_ client*

Cada vez que se recibe un nuevo dato de posición de los actuadores este es enviado a CoppeliaSim mediante la función *step_ coppelia_ client*. Para que todas las posiciones se actualicen en el mismo paso de simulación, se pausa la comunicación, se acumulan las solicitudes en una cola y después se reanuda la comunicación enviando toda la cola.

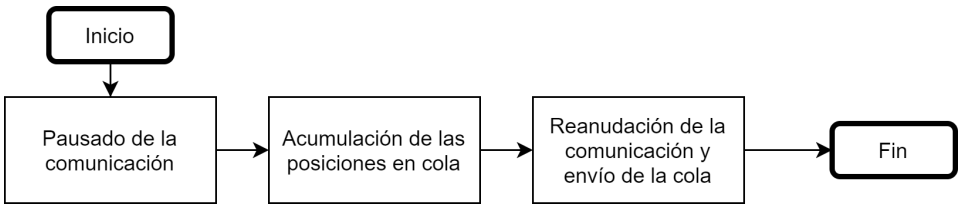


Figura 3.24: Diagrama de flujo *step_ coppelia_ client*

Finalmente, cuando ya no se quieran enviar nuevas posiciones, el cliente se deberá desconectar empleando la función *stop_ copppelia_ client*, que simplemente cierra la sesión de comunicación.

3.2.2 Resultados

Para verificar que se pueden reproducir los movimientos del robot real se realizan dos ensayos en los que el modelo simulado operará en modo cinemática directa. En el primer ensayo se siguen unas trayectorias predefinidas utilizando un controlador no lineal por pasividad (ver subsección 3.3.1), mientras que en el segundo se desfrenan los motores y el robot se mueve libremente de forma manual.

En primer lugar se lanza la simulación en CoppeliaSim, que tendrá un paso menor o igual al periodo con el que se desee actualizar la posición desde el cliente (en este caso de 5ms), y suficientemente pequeño como para que los movimientos del modelo simulado sean fluidos. A continuación, se inicia el cliente y se comprueba que la conexión ha sido correcta. Finalmente se puede activar el robot real y observar como el robot simulado se mueve de forma coordinada.

Tras la finalización de los ensayos, se comprueba que se obtienen resultados satisfactorios, el modelo consigue reproducir fielmente los movimientos del robot sin prácticamente ningún retardo. Los vídeos correspondientes a estos ensayos son accesibles desde la siguiente dirección:

`https://upvedues-my.sharepoint.com/:f:/g/personal/dane2_upv_edu_es/En81Ei9h6vNNsMiWVYY83T0BpDbP07--joj68UIS6Oxyhg?e=d2CqVJ`

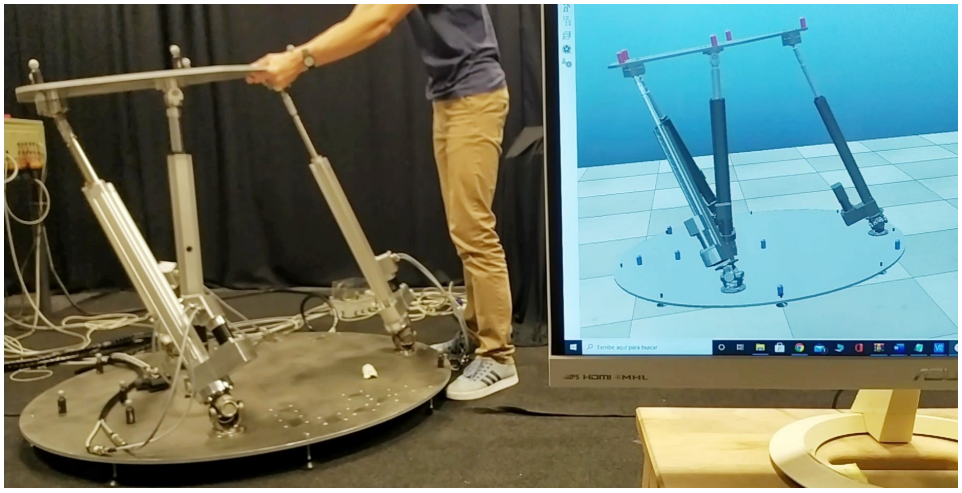


Figura 3.25: Ensayo con movimientos manuales

Por último, se hace una comparación aproximada del ancho de banda necesario para la solución desarrollada y una alternativa como puede ser una videollamada en la que únicamente se transmite la señal de vídeo en una dirección. El tráfico de datos se mide con el monitor de recursos del sistema operativo Windows 10, ejecutándose en el equipo receptor, en el cual también se realiza la simulación de CoppeliaSim. Este equipo presenta un procesador Intel Core i5-2450M (2.5GHz), SSD y 6GB de RAM (DDR3) y se conecta a una red WiFi de 2.4GHz.

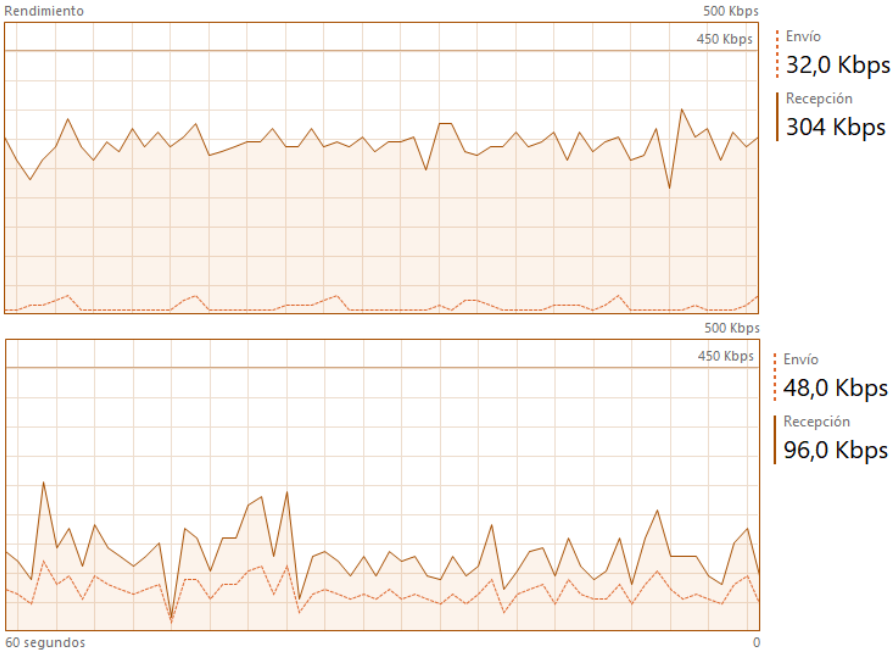


Figura 3.26: Transferencia de datos con videollamada (arriba) y con CoppeliaSim (abajo)

En las figura anterior se comprueba que si se emplea CoppeliaSim junto a un cliente TCP/IP efectivamente se genera un tráfico de datos total menor. Cabe destacar además, que la ejecución de la simulación se realiza de forma fluida incluso en un equipo de prestaciones modestas.

3.3 Aplicación 2: co-simulación

En esta aplicación se plantea como objetivo la co-simulación del modelo dinámico del robot junto con un programa externo, que hará las labores de control. En concreto se va a trabajar con Matlab/Simulink, donde previamente se han implementado diversos reguladores y un modelo dinámico del robot. En la figura 3.27 se muestra el diagrama de bloques principal, en el que se implementa un control en bucle cerrado, con bloques de referencia, controlador y planta.

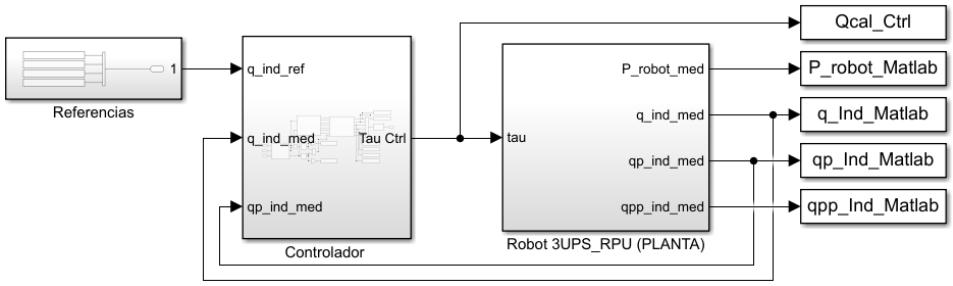


Figura 3.27: Diagrama de bloques base

En el bloque de la planta, a partir de la suma de las fuerza en los actuadores, los términos centrífugos y de Coriolis, gravitacionales, del entorno y la fuerza de rozamiento, se determina la fuerza equivalente compacta a la que está sometido el robot. Con esta fuerza, se calculan las aceleraciones que son integradas consiguiendo la velocidad y la posición de los actuadores y de la plataforma. Por otro lado, se obtienen los Jacobianos y el complemento ortogonal, que junto con los datos anteriores sirven para calcular los parámetros dinámicos con los que se realimenta el sistema. Las posiciones, velocidades y aceleraciones constituyen las salidas de la planta.

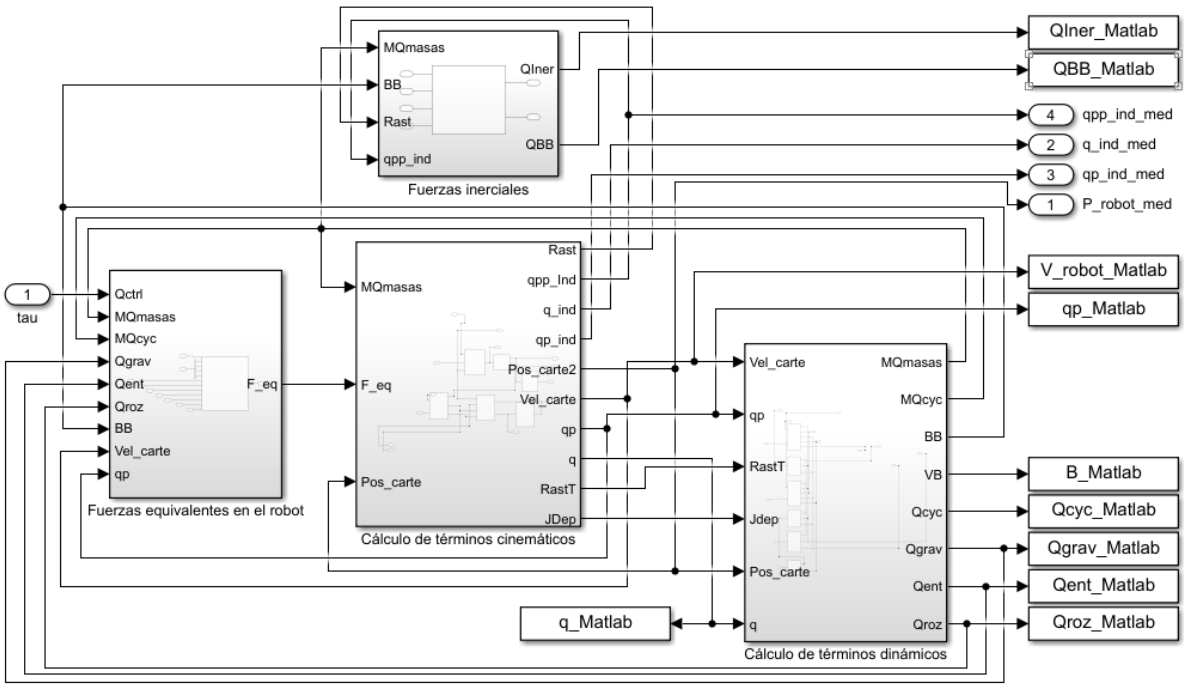


Figura 3.28: Diagrama de bloques de la planta

Se desea sustituir el modelo dinámico de Simulink por el modelo obtenido en CoppeliaSim y realizar un seguimiento de trayectorias para comprobar que su respuesta es adecuada. Una vez validado el modelo de CoppeliaSim este podrá ser empleado para probar nuevos controladores, que no necesariamente deberán estar implementados en Matlab/Simulink gracias a la diversidad de lenguajes de programación que soporta la API remota de este programa.

3.3.1 Controladores empleados

En primer lugar, se propone un controlador no lineal por pasividad, que trata de modificar la energía natural del sistema de forma que esta tenga un mínimo en la posición deseada [12]. En concreto se empleará un controlador punto por punto, cuya expresión general es:

$$\tau = -K_p \cdot e - K_d \cdot \nu - g \tag{3.1}$$

Donde τ es la acción de control, correspondiente a la fuerza en la Ecuación 2.37.

Dependiendo de la definición de los parámetros ν y g de la ecuación es posible obtener distintos controladores. En este caso se empleará un controlador proporcional derivativo con compensación del efecto de la gravedad (PD+G) cuya ley de control viene expresada por:

$$\tau = -K_p \cdot e - K_d \cdot \dot{q} + G(q, \theta) \tag{3.2}$$

La implementación en Simulink de este controlador se muestra a continuación. En primer lugar, se observa un retenedor con un periodo de muestreo de 10 milisegundos que permite simular el comportamiento del controlador real. Seguidamente se tiene un estimador de la velocidad de primer orden que se ha añadido debido a limitaciones a la hora de obtener directamente las velocidades del robot real. A partir de las posiciones medidas y de referencia se computa el error de posición que junto con la velocidad sirve de entradas a un regulador proporcional derivativo estándar. Por otro lado, en base a la posición medida se obtiene el término gravitacional que se suma a la salida del regulador anterior constituyendo la acción de control.

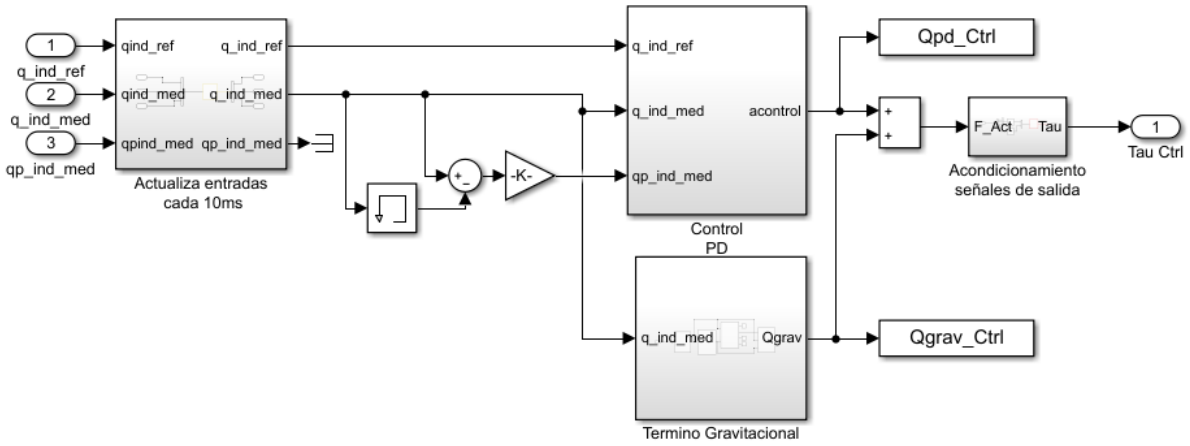


Figura 3.29: Diagrama de bloques del controlador PD+G

Como segunda propuesta, también se tiene un controlador no lineal pero en este caso de dinámica inversa, que cancela las no linealidades del sistema a controlar para que su dinámica en bucle cerrado sea lineal [13].

Para este tipo de reguladores, dado un modelo dinámico definido por:

$$\vec{x}^{(n)} = f(\vec{x}) + b(\vec{x}) \cdot \vec{u} \tag{3.3}$$

Donde $f(\vec{x})$ y $b(\vec{x})$ son funciones no lineales y $\vec{u}$ es la entrada de control. Si dicha entrada de control se puede expresar como:

$$\vec{u} = \frac{1}{b(\vec{x})}[\vec{a} - f(\vec{x})] \tag{3.4}$$

Entonces cancelarán las no-linealidades y se obtendrá una relación entrada-salida simple:

$$\vec{x}^{(n)} = \vec{a} \tag{3.5}$$

Donde $\vec{a}$ podrá tomar diferentes valores proporcionando controladores distintos.

De esta forma, para poder utilizar un controlador de dinámica inversa se debe comprobar que es posible escribir la ecuación dinámica del robot (Ecuación 2.37) en la forma canónica controlable (Ecuación 3.3). Si de la expresión original se despeja el vector de aceleraciones $\ddot{\vec{q}}$:

$$\ddot{\vec{q}} = M^{-1}(\vec{q}) \left(\vec{\tau} - \vec{C}(\vec{q}, \dot{\vec{q}} \cdot \dot{\vec{q}} - \vec{G}(\vec{q}) \right) \tag{3.6}$$

Se pueden definir los siguientes términos:

$$f(\vec{x}) = M^{-1}(\vec{q}) \left(-\vec{C}(\vec{q}, \dot{\vec{q}} \cdot \dot{\vec{q}} - \vec{G}(\vec{q}) \right) \tag{3.7}$$

$$b(\vec{x}) = M^{-1}(\vec{q}) \tag{3.8}$$

Los cuales se sustituyen en la Ecuación 3.4:

$$\vec{\tau} = \frac{1}{M^{-1}(\vec{q})} \left[\vec{a} - M^{-}1(\vec{q}) \left(-\vec{C}(\vec{q}, \dot{\vec{q}} \cdot \dot{\vec{q}} - \vec{G}(\vec{q}) \right) \right] \tag{3.9}$$

Finalmente, simplificando se obtiene la ley de control del regulador:

$$\vec{\tau} = M(\vec{q}) \cdot \vec{a} + \vec{C}(\vec{q}, \dot{\vec{q}} \cdot \vec{q} + \vec{G}(\vec{q}) \tag{3.10}$$

En este caso el valor de $\vec{a}$ se ajustará para obtener un controlador de trayectoria, que dependerá de la aceleración y los errores de velocidad y posición:

$$\vec{a} = \ddot{\vec{q}} - K_q \cdot \dot{\vec{e}} - K_p \cdot \vec{e} \tag{3.11}$$

En base a las expresiones anteriores, se implementa el controlador en Simulink. Al igual que en el caso previo, se tiene un retenedor con el mismo periodo de muestreo. A partir de las velocidades y posiciones medidas en los actuadores, se obtienen dichos parámetros en las coordenados generalizadas del robot además del Jacobiano y el complemento ortogonal. Por otro lado, se calculan los errores de velocidad y posición que junto con los parámetros anteriores permiten obtener los diferentes términos dinámicos del robot. Mediante su suma finalmente se obtiene la acción de control.

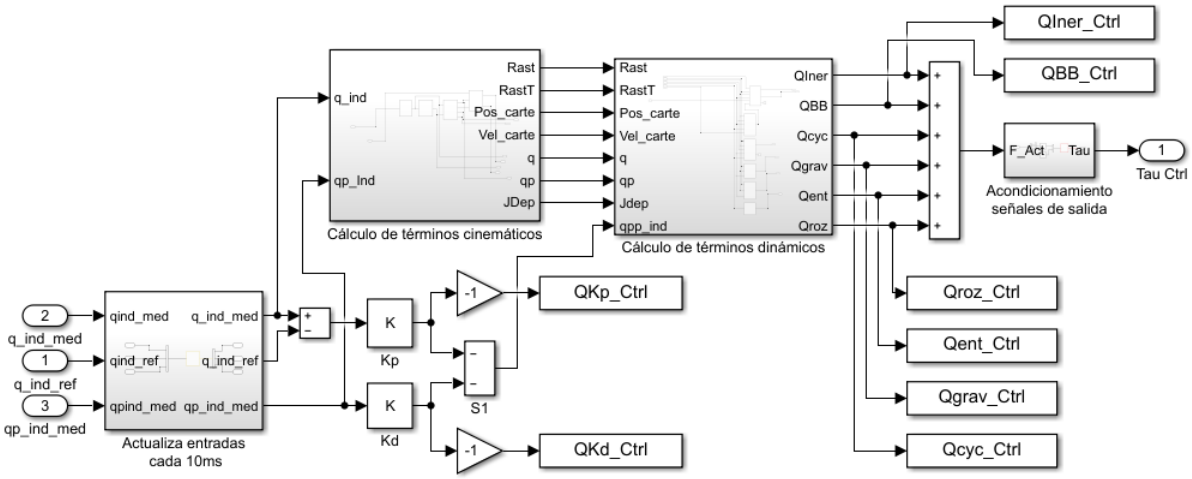


Figura 3.30: Diagrama de bloques del controlador de dinámica inversa

3.3.2 Conexión de CoppeliaSim con Matlab/Simulink

En la co-simulación se pretende que Matlab haga de cliente tomando las lecturas necesarias y proporcionando las acciones de control al robot en CoppeliaSim que hará de servidor. Se debe realizar una simulación síncrona para asegurarse que cliente y servidor operan en el mismo instante de tiempo, para ello se tendrán en cuenta los modos de simulación explicados en la subsección 2.3.5.

El cliente se creará en Matlab empleando la API remota de CoppeliaSim, y será el encargado de controlar la evolución de la co-simulación. En cada iteración el cliente deberá esperar a que el servidor responda para asegurarse que se reciben y se envían los datos correctamente, por lo tanto se emplearán llamadas con bloqueo de ejecución.

Inicialmente se implementó el cliente mediante la API remota clásica y se realizó un test sencillo de seguimiento de trayectorias empleando los controladores internos de CoppeliaSim. Aunque los resultados obtenidos fueron adecuados, la co-simulación requirió mucho tiempo para completarse, haciendo poco viable la realización de múltiples ensayos en un tiempo razonable. En la figura 3.31 se muestra una prueba incompleta de seguimiento de trayectorias.

Tras un análisis de los componentes de la co-simulación se determinó que el cuello de botella fue la comunicación entre cliente y servidor mediante *sockets*. El tiempo necesario para la respuesta fue elevado, por lo que en combinación con la gran cantidad de pasos de simulación necesarios se obtuvo una ejecución muy lenta. No se puede concluir que la comunicación mediante *sockets* sea lenta, pero sí que CoppeliaSim necesita un tiempo considerable para poder devolver una respuesta empleando este método.

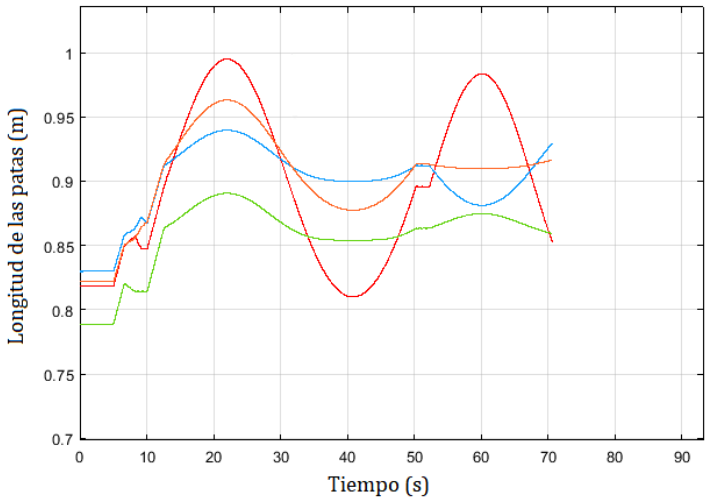


Figura 3.31: Co-simulación incompleta empleando *sockets*

Afortunadamente se dispone de otras formas de comunicación con las que se obtiene una respuesta más rápida, como puede ser BlueZero. Como se ha introducido en la subsección 2.3.3, la red de BlueZero se compone de un bróker y diversos nodos. En este caso habrá dos nodos, uno implementado en CoppeliaSim y otro en Matlab.

La forma más sencilla de crear un nodo en CoppeliaSim es emplear la herramienta *B0 Remote API Server* que se puede encontrar en `Model browser, tools`. Se trata simplemente de un *dummy* con un *script* asociado que trata de conectar un nuevo nodo a un bróker existente. El *script* también inicia el bróker si detecta que no hay ninguno activo.

El nodo de Matlab se debe implementar de forma manual mediante código (ver Apéndice B), además debe desarrollar el algoritmo que permita ejecutar de forma síncrona las simulaciones de ambos programas. Se comienza creando la función *startCoppeliaSimB0* que inicia el nodo de Matlab y arranca la simulación de forma síncrona en CoppeliaSim.

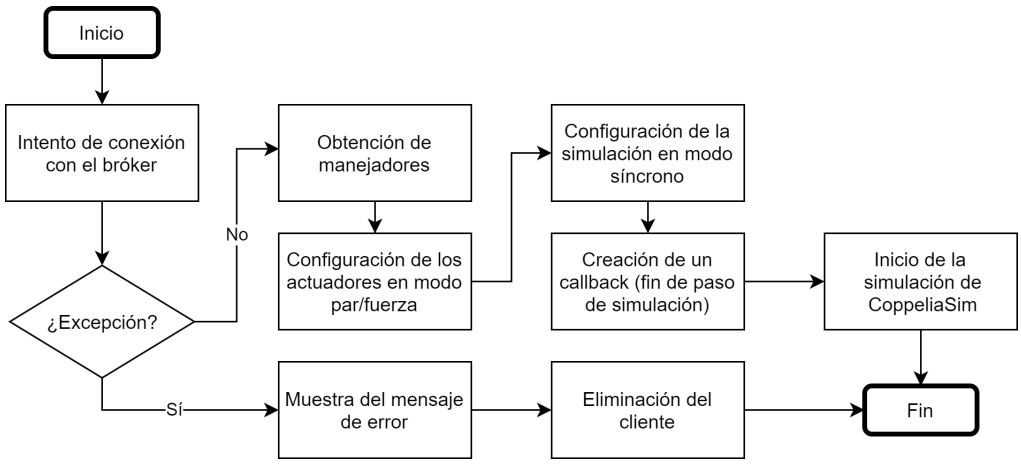


Figura 3.32: Diagrama de flujo *startCoppeliaSimB0*

Seguidamente, se define la función que detiene la simulación en CoppeliaSim y desconecta el nodo del bróker.

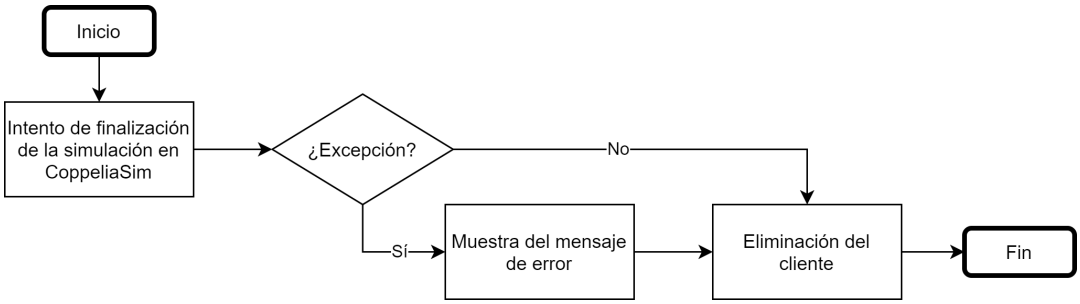


Figura 3.33: Diagrama de flujo *stopCoppeliaSimB0*

Las funciones de inicio y parada se añaden a las `Callbacks` dentro del menú `Model properties` del modelo de Simulink, para que al arrancar o finalizar su ejecución, automáticamente se inicie o se termine la simulación en CoppeliaSim. En este momento se puede ejecutar la macro de Matlab que lanza la simulación y comprobar que los resultados son adecuados, el robot de CoppeliaSim no se moverá pero la simulación se habrá iniciado y luego detenido.

Una vez se ha determinado que es posible la conexión entre ambos programas, se procede a la implementación de la planta del robot mediante un bloque *Interpreted Matlab Function*. Como este bloque solo permite tener una entrada y una salida se creará un subsistema para facilitar las conexiones con el resto de bloques. La entrada a este subsistema será un vector con las acciones de control (fuerzas) mientras que para las salidas se tendrá un vector con la posición y rotación de la plataforma móvil y por otro lado dos vectores con las posiciones y velocidades de los actuadores (ver figura 3.35).

A continuación, se implementa el algoritmo que aplica las acciones de control y obtiene las medidas de interés. Este algoritmo se divide en una función principal que se añade directamente dentro del bloque *Interpreted Matlab Function*, y una serie de funciones auxiliares.

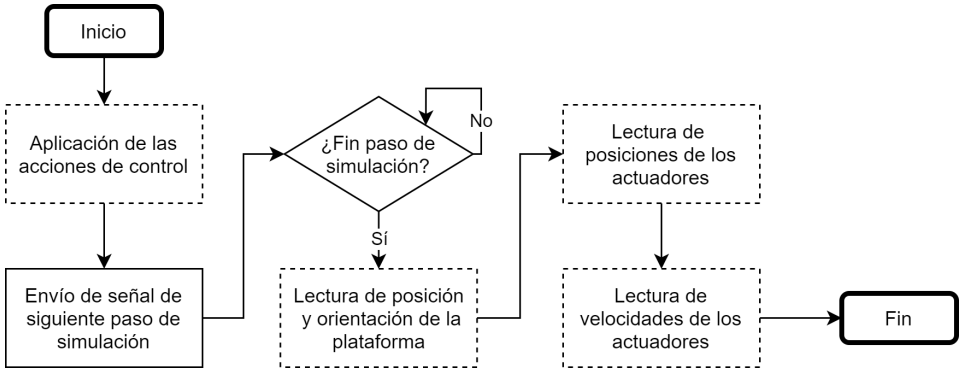


Figura 3.34: Diagrama de flujo *stepCoppeliaSimB0*

Al comienzo de la simulación no se dispone de las medidas de los parámetros del robot, por ello se añade un bloque *Memory* después del bloque *Interpreted Matlab Function* cuyas condiciones iniciales son la primera posición de la trayectoria que se quiere seguir y una velocidad nula. Finalmente, se obtiene el siguiente diagrama de bloques para el robot, el cual puede sustituir directamente la planta original mostrada en la figura 3.28.

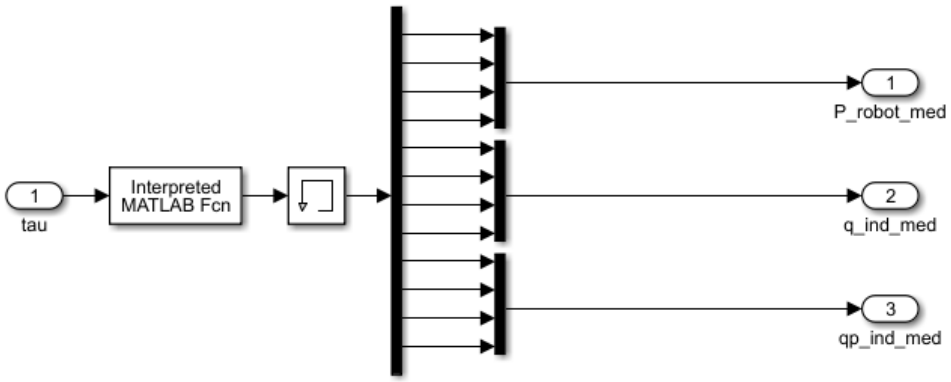


Figura 3.35: Diagrama de bloques del robot de CoppeliaSim

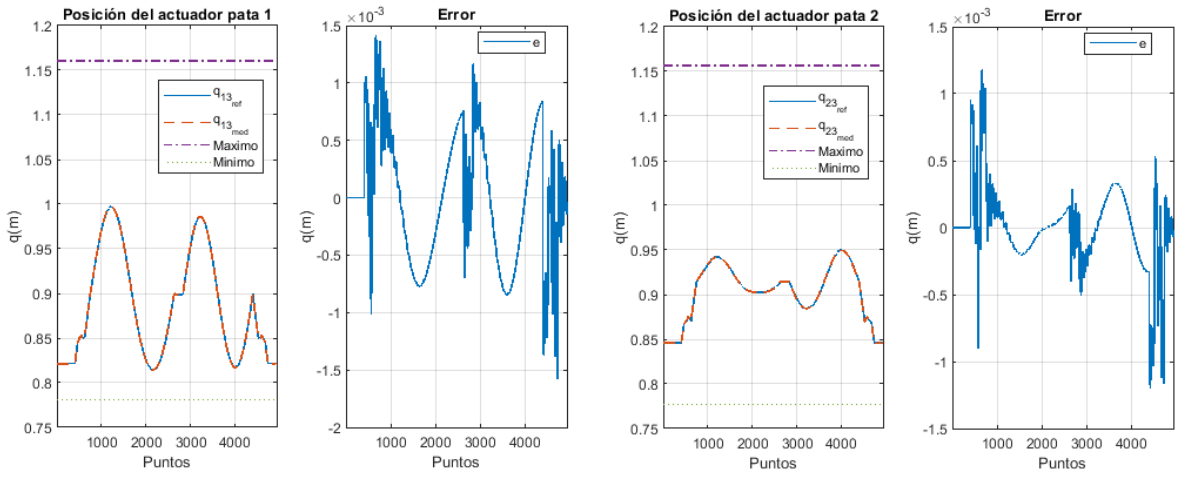
De esta forma concluye la conexión del robot realizado en CoppeliaSim con el entorno de Matlab/Simulink. Si se ejecuta la simulación, puede comprobar que el tiempo necesario para su finalización es mucho menor que en el caso de emplear la API remota clásica de CoppeliaSim.

3.3.3 Resultados

Para tener un orden de magnitud de la bondad de los resultados obtenidos con el modelo de CoppeliaSim se van a analizar primero las respuestas obtenidas con la planta de Simulink y con el robot real para una misma trayectoria de referencia empleando el regulador PD+G. Posteriormente, se analizará la respuesta del modelo de CoppeliaSim también con el controlador de dinámica inversa.

El control se realizará con un periodo de 10 milisegundos en todos los casos. En el caso de la simulación del robot en Simulink y en CoppeliaSim se establecerá un paso de simulación de 5 milisegundos, para poder aproximar el comportamiento de estos modelos a un proceso continuo en comparación con el regulador.

Tendiendo en cuenta lo anterior, se realiza primero el ensayo con Simulink observándose un error absoluto máximo de 1.5 milímetros en cualquiera de los actuadores.



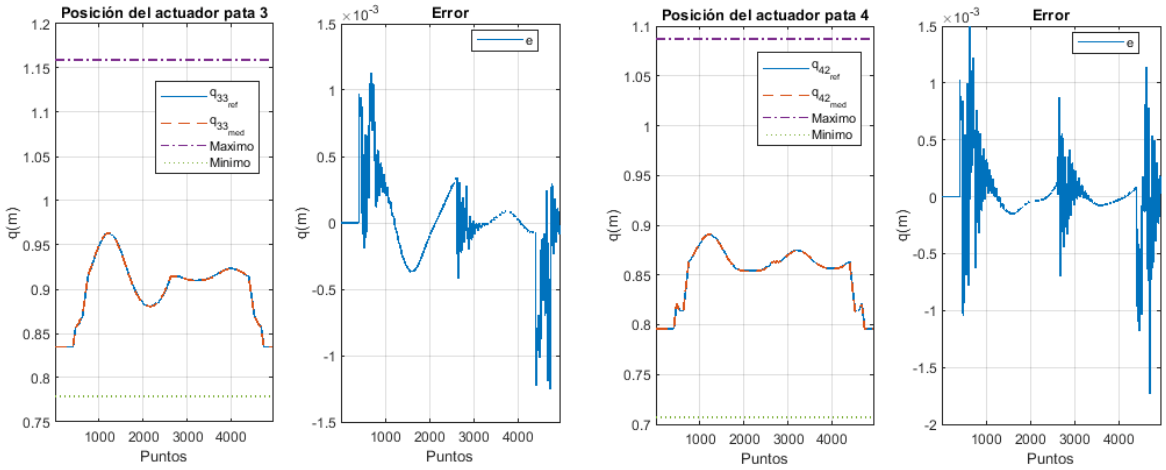


Figura 3.36: Posición de los actuadores y error cometido (PD+G / planta Simulink)

En el ensayo con robot real se obtiene un error de una magnitud similar, presentando un valor absoluto de menos de 3 milímetros.

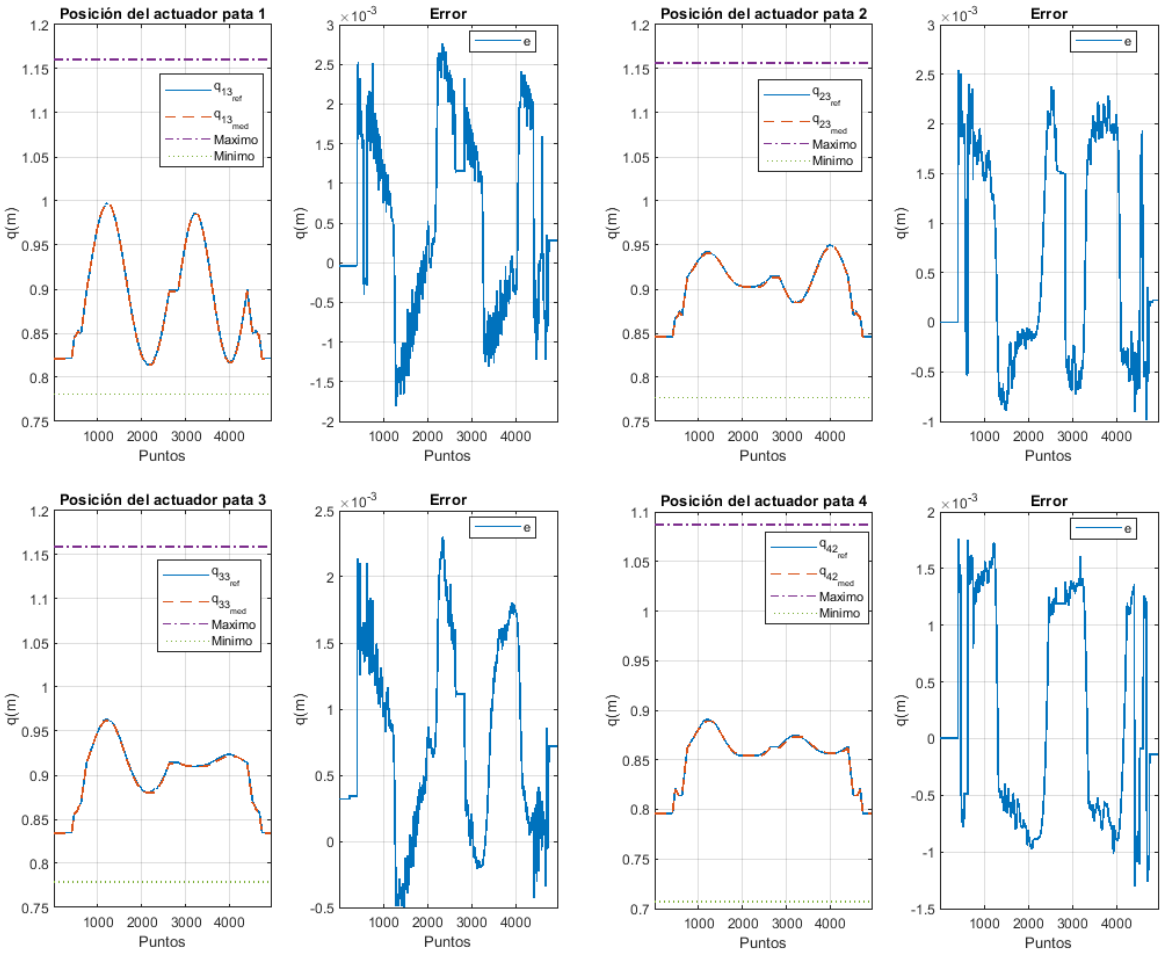


Figura 3.37: Posición de los actuadores y error cometido (PD+G / robot real)

En el caso de la simulación con CoppeliaSim se observa que el error absoluto máximo es de 3,2 milímetros, muy similar al cometido en las pruebas anteriores.

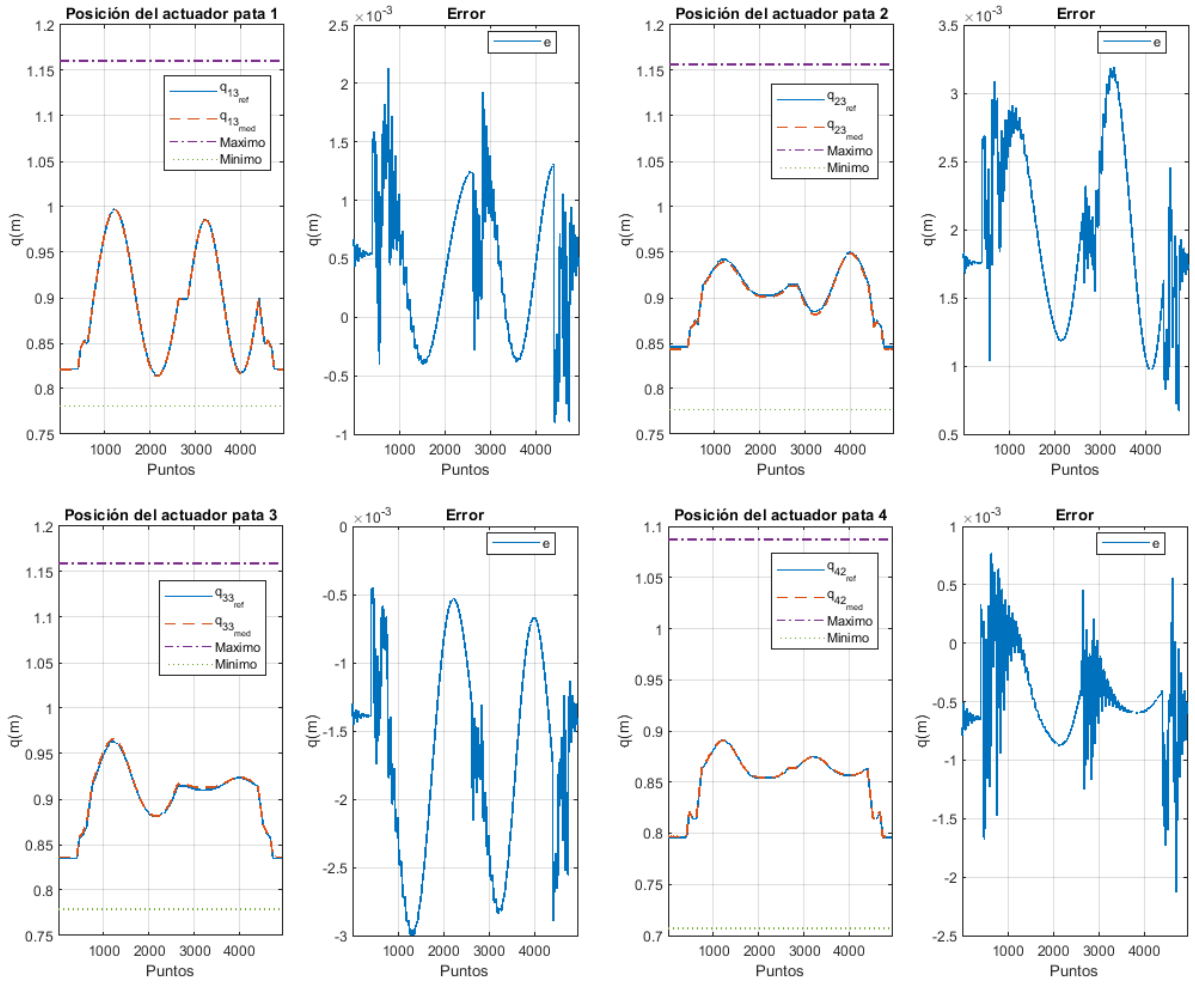


Figura 3.38: Posición de los actuadores y error cometido (PD+G / robot CoppeliaSim)

Se trata de unos errores pequeños en comparación con el tamaño del robot. Hay destacar que se ha realizado un ensayo con una velocidad elevada (0.04m/s y 4°/s), por lo que el efecto de los términos dinámicos es más notable. Se trata de una velocidad alta para aplicaciones de rehabilitación, en un caso real se trabajaría a velocidades más reducidas (0.02m/s y 2°/s).

Por otro lado, se comprueba si la plataforma es capaz de alcanzar las posiciones y orientaciones deseadas en la simulación de CoppeliaSim. Este análisis es más crítico, ya que determina si el robot se ha modelado con la geometría adecuada. De nuevo se comprueba que se consiguen seguir las referencias satisfactoriamente, por lo tanto el ensamblaje en CoppeliaSim es correcto.

Se observa un pequeño error de *offset* en la posición de la plataforma móvil en el eje X, que puede ser debido a la diferencia entre los parámetros dinámicos que utiliza CoppeliaSim y los que se obtuvieron en SolidWorks. Este fenómeno es menos apreciable en la posición en el eje Z. Una posible solución para esta situación, podría ser calibrar nuevamente el controlador para que trabaje específicamente con el modelo de CoppeliaSim.

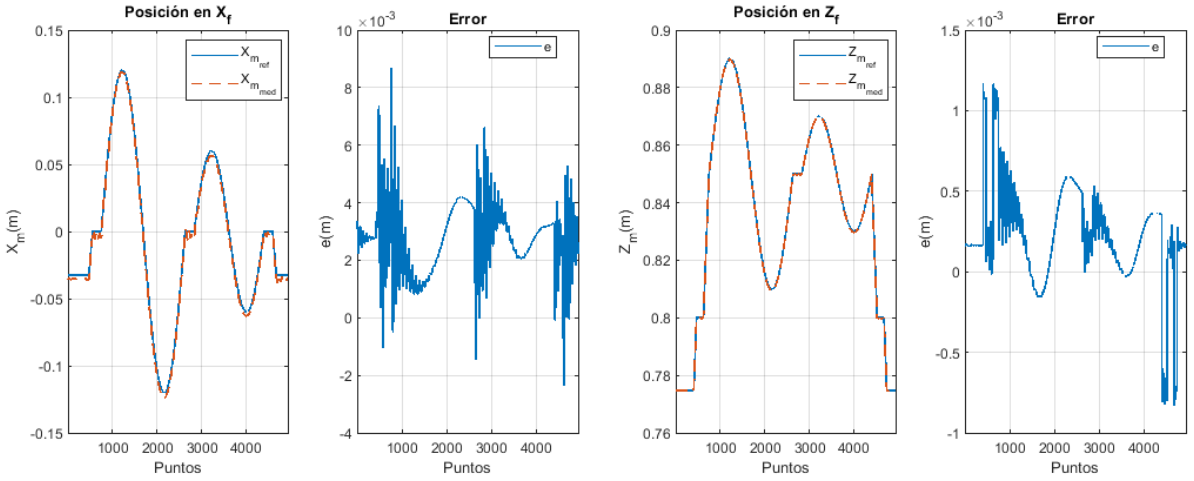


Figura 3.39: Posiciones de la plataforma móvil (PD+G / robot CoppeliaSim)

En lo que respecta a las orientaciones de la plataforma, se comprueba que la inclinación o ángulo Θ coincide con la referencia. Sin embargo, para la rotación o ángulo Ψ se comete un error sobretodo en los extremos del rango de movimiento. El ángulo Ψ es la referencia más complicada de seguir debido a la geometría del robot.

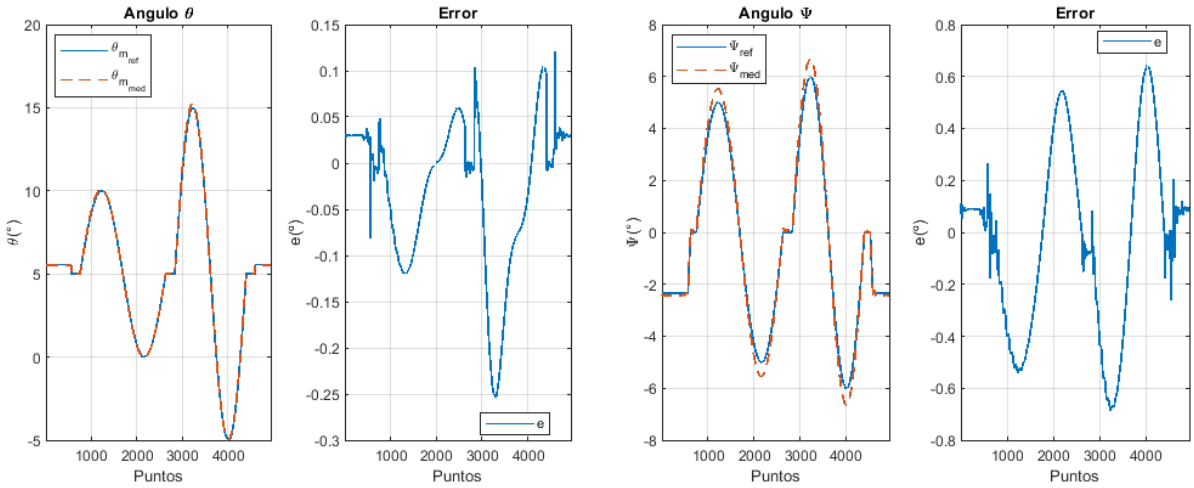


Figura 3.40: Orientaciones de la plataforma móvil (PD+G / robot CoppeliaSim)

Seguidamente, se realiza una co-simulación con CoppeliaSim empleando el controlador por par calculado. Se empleará el mismo paso de simulación que en los ensayos anteriores y se seguirá exactamente la misma trayectoria para poder realizar una comparación posterior de las respuestas.

Se analizan las posiciones de los actuadores y se comprueba que el nuevo controlador es capaz de proporcionar unos mejores resultados que el controlador por pasividad, consiguiéndose un error máximo incluso menor que en todos los casos anteriores, de menos de un milímetro.

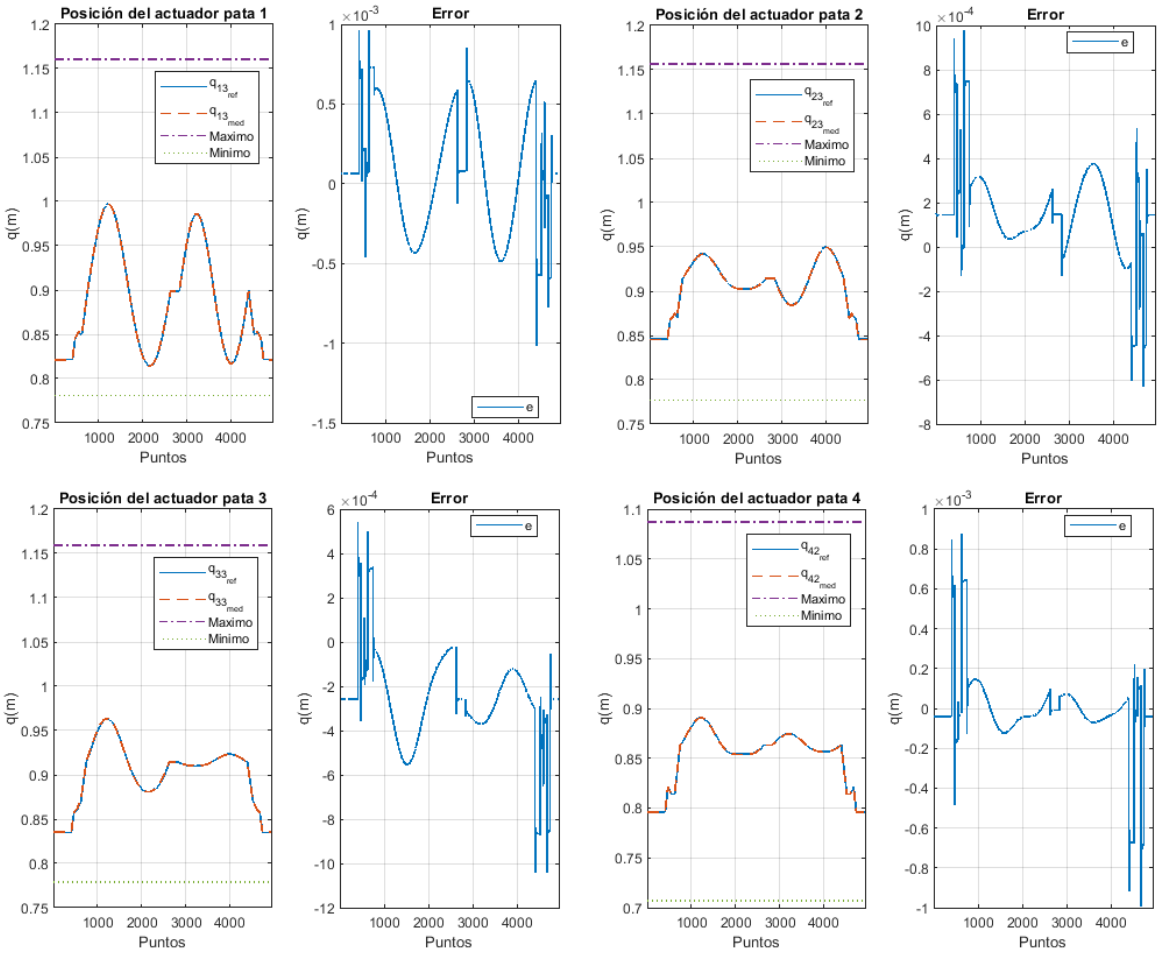


Figura 3.41: Posición de los actuadores y error cometido (Din. Inv. / robot CoppeliaSim)

La reducción del error en los actuadores también tiene un efecto positivo en las posiciones y orientaciones de la plataforma móvil.

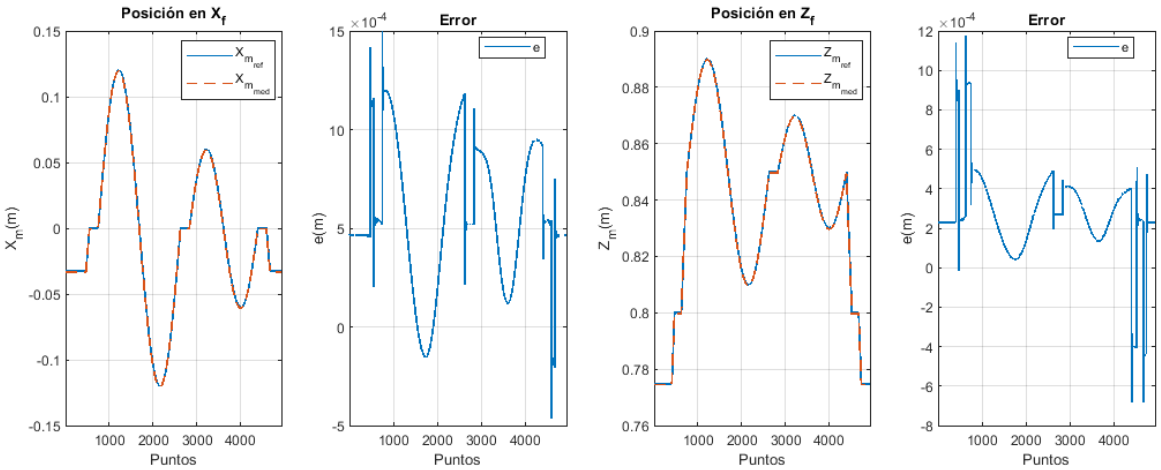


Figura 3.42: Posiciones de la plataforma superior (Din. Inv. / robot CoppeliaSim)

La plataforma consigue seguir satisfactoriamente las referencias de posición en ambos ejes, con un error mínimo que apenas supera el milímetro.

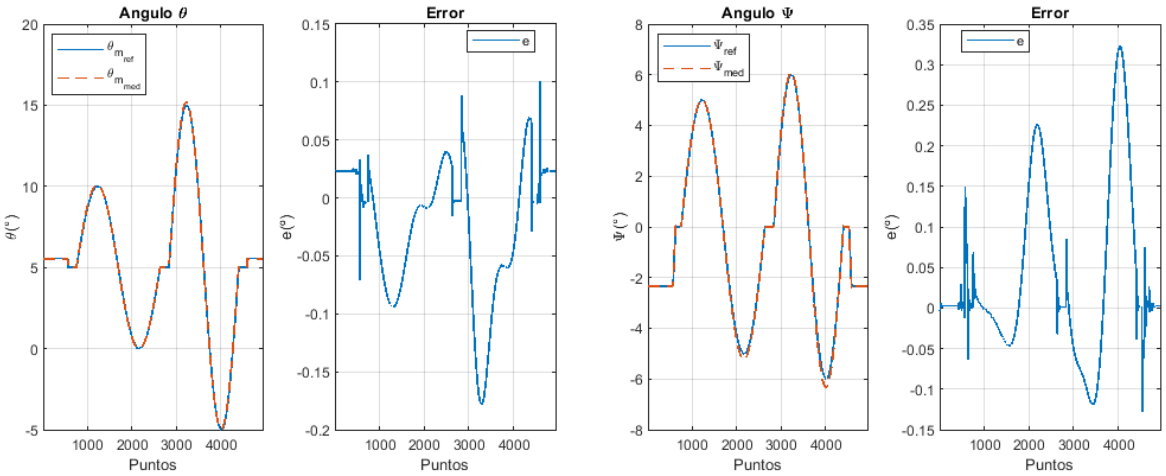


Figura 3.43: Orientaciones de la plataforma superior (Din. Inv. / robot CoppeliaSim)

Un resultado similar se observa en las orientaciones de la plataforma. En el ángulo Θ se comete un error parecido al obtenido con el controlador PD+G, sin embargo, el ángulo Ψ presenta una respuesta mucho mejor, con un error reducido incluso en los extremos del rango de movimiento.

Como último punto de este apartado, se presenta una serie de tablas resumen con los errores cometidos en los ensayos. En concreto se calculan el error medio y el error cuadrático medio:

$$error\,medio = \frac{\sum e_i}{n} \qquad error\,cuadratico\,medio = \sqrt{\frac{\sum {e_i}^2}{n}} \tag{3.12}$$

El primer parámetro analizado es el error de posición cometido en las patas del robot.

Ensayo	**Regulador**	**Error medio**			
		1 (m)	**2 (m)**	**3 (m)**	**4 (m)**
Simulink	PD+G	-5.2665E-07	-1.3177E-06	6.1074E-07	1.7029E-07
Real	PD+G	5.0119E-04	6.3432E-04	7.4650E-04	1.7865E-04
Coppelia	PD+G	4.6810E-04	1.9000E-03	-1.7000E-03	-5.0886E-04
Simulink	Din. Inv.	1.6026E-10	9.7412E-11	1.8228E-10	7.0380E-11
Coppelia	Din. Inv.	7.0521E-05	1.4983E-04	-2.5813E-04	-1.3168E-05

Tabla 3.1: Error medio cometido en la posición de las patas del robot

Ensayo	**Regulador**	**Error cuadrático medio**			
		1 (m)	**2 (m)**	**3 (m)**	**4 (m)**
Simulink	PD+G	5.7091E-04	3.0695E-04	3.1081E-04	3.1256E-04
Real	PD+G	1.2619E-03	1.2300E-03	1.0284E-03	9.4925E-04
Coppelia	PD+G	7.5488E-04	2.0482E-03	1.8127E-03	6.3949E-04
Simulink	Din. Inv.	3.6746E-04	1.9899E-04	1.9996E-04	1.9378E-04
Coppelia	Din. Inv.	3.7360E-04	2.5281E-04	3.3365E-04	1.9454E-04

Tabla 3.2: Error cuadrático medio cometido en la posición de las patas del robot

Por otro lado, se comparan los errores cometidos en las posiciones y orientaciones de la plataforma móvil. En este caso no se dispone información del robot real.

Ensayo	**Regulador**	**Error medio**			
		X (m)	**Z (m)**	**Θ(°)**	**Ψ(°)**
Simulink	PD+G	-2.7188E-06	6.4537E-06	-9.2242E-05	5.5224E-04
Coppelia	PD+G	2.8000E-03	2.2316E-04	-2.6700E-02	-1.9200E-02
Simulink	Din. Inv.	-7.9033E-08	1.3331E-07	-3.2863E-06	1.7752E-05
Coppelia	Din. Inv.	5.9774E-04	2.6481E-04	-2.2300E-02	3.9200E-02

Tabla 3.3: Error medio cometido en la posición y orientación de la plataforma móvil

Ensayo	**Regulador**	**Error medio**			
		X (m)	**Z (m)**	**Θ(°)**	**Ψ(°)**
Simulink	PD+G	1.1000E-03	2.9566E-04	4.4000E-02	3.7900E-02
Coppelia	PD+G	3.1000E-03	3.8390E-04	8.4500E-02	3.5280E-01
Simulink	Din. Inv.	3.5418E-04	2.1744E-04	2.9800E-02	2.2700E-02
Coppelia	Din. Inv.	7.0211E-04	3.4286E-04	6.0700E-02	1.1340E-01

Tabla 3.4: Error cuadrático medio cometido en la posición y orientación de la plataforma móvil

Se observa que los errores cometidos son de un mismo orden de magnitud y por tanto es posible concluir que se ha obtenido un modelo dinámico adecuado, aún teniendo en cuenta las limitaciones de edición de propiedades dinámicas de CoppeliaSim. También, se comprueba que el controlador de dinámica inversa consigue un mejor seguimiento de referencias, sin embargo, cabe destacar que tiene un coste computacional más elevado que en el caso del PD+G, que también logra unos resultados satisfactorios.

Capítulo 4

Conclusiones

Tras finalizar este trabajo, se puede afirmar que se han cumplido satisfactoriamente todos los objetivos propuestos inicialmente. Se ha realizado un proyecto que ha involucrado diferentes tecnologías y áreas de conocimiento: física, modelado 3D, simulación de modelos, programación y sistemas de comunicación.

En primer lugar, se ha podido incorporar el diseño CAD del robot a CoppeliaSim y obtener un modelo cinemático y uno dinámico, resolviéndose los problemas de cinemática y dinámica directa, y aunque solo como prueba de concepto, también el problema de cinemática inversa.

A continuación, se ha desarrollado una herramienta visualización remota a partir del modelo cinemático, que permite realizar el seguimiento de los movimientos del robot de rehabilitación en tiempo real sin necesitar un gran ancho de banda para la comunicación. Por otro lado, se han sentado las bases para la realización de co-simulaciones mediante CoppeliaSim junto a programas externos, en este caso junto a Matlab/Simulink. De esta forma se ha podido probar diversos controladores no lineales y validar el modelo dinámico.

CoppeliaSim ha demostrado ser una plataforma de simulación muy potente. Se trata de un programa muy versátil, que permite la simulación de cualquier tipo de robot y la integración con otros programas de forma sorprendentemente sencilla gracias sus extensas APIs remotas. Pese a algunas limitaciones, la facilidad de uso más el hecho de que se puede tener un acceso gratuito al software, hacen de CoppeliaSim una alternativa muy atractiva.

De esta forma se ha abierto la puerta al desarrollo de futuras aplicaciones en las que se pueda visualizar y controlar el robot a distancia con una baja latencia, como la telerehabilitación y la telemedicina. También se ha creado un entorno de simulación permitirá el diseño de reguladores avanzados sin necesidad de recurrir a ensayos con el robot real.

Por último, se puede concluir que con el desarrollo de este Trabajo Final de Máster, ha sido posible familiarizarse con la simulación de robots obteniendo una visión global de las técnicas necesarias para poder llevar a cabo esta tarea con éxito. La experiencia adquirida mediante la realización de este proyecto, junto con los conocimientos obtenidos a lo largo de la titulación, sirven de punto de partida para el desarrollo profesional como futuro ingeniero especializado en control, automática y sistemas robotizados.

Bibliografía

[1] Aaron M. Dollar y Hugh Herr. «Lower Extremity Exoskeletons and Active Orthoses: Challenges and State-of-the-Art». En: *IEEE TRANSACTIONS ON ROBOTICS* 24 (1 feb. de 2008) (vid. pág. 7).

[2] H. y H. Alaranta Tropp. «Proprioception and Coordination Training in Injury Prevention». En: *Sports Injuries: Basic Principles of Prevention and Care* (1993) (vid. pág. 8).

[3] Grigore C. Burdea y Mourad Bouzit Michael J. Girone. «THE "RUTGERS ANKLE" ORTHOPEDIC REHABILITATION INTERFACE». En: *Proceedings os the ASME international Mechanical Enge. Congr. Dyn. Syst. Control Dive* (1999) (vid. pág. 8).

[4] DR. EMILIO L. JUAN GARCÍA. *Principios básicos de la movilidad articular*. 2018 (vid. pág. 8).

[5] Ángel Valera Fernández Joan Olucha Salla. «Modelado y simulación de un robot paralelo de 4 grados de libertad. Aplicación a un robot de rehabilitacion». 2015 (vid. pág. 8).

[6] T. J. y W. B. Kibler Chandler. «Muscle Training in Injury Prevention». En: *Sports Injuries: Basic Principles of Prevention and Care* (1993) (vid. pág. 9).

[7] Wikipedia. *Stewart platform*. `https://en.wikipedia.org/wiki/Stewart_platform`. Accedido el: 22-05-2021 (vid. pág. 9).

[8] A. Barrientos L. F. Peñín C. Balaguer y R. Aracil. *Fundamentos de Robótica, 2nd ed.* 2007 (vid. págs. 10, 11).

[9] Javad Enferadi Alireza Akbarzadeh y Mahdi Sharifnia. «Dynamics analysis of a 3-RRP spherical parallel manipulator using the natural orthogonal complement». En: (2013) (vid. pág. 19).

[10] Coppelia Robotics. *Enabling client*. `https://www.coppeliarobotics.com/helpFiles/en/remoteApiClientSide.htm`. Accedido el: 30-05-2021 (vid. pág. 24).

[11] Coppelia Robotics. *Enabling B0 node*. `https://www.coppeliarobotics.com/helpFiles/en/b0RemoteApiClientSide.htm`. Accedido el: 30-05-2021 (vid. pág. 25).

[12] P. Tomei. «Adaptive PD Controller for Robot Manipulators». En: *Trans. Robot. Autom* (1991) (vid. pág. 48).

[13] G. Basting C. Canudas B. Siciliano. *Theory of Robot Control*. London: Springer-Verla, 1996 (vid. pág. 49).

Parte II

PRESUPUESTO

Capítulo 1

Presupuesto

1.1 Consideraciones previas

En este presupuesto se aborda el análisis económico del proyecto desarrollado siguiendo las recomendaciones del Centro de apoyo a la innovación, la investigación y la transferencia de tecnología (CTT) de la Universitat Politècnica de València.

1.2 Análisis económico

1.2.1 Gastos materiales

La amortización de los equipos informáticos y paquetes informaticos empleados se calcula en base a la siguiente expresión.

$$\text{coste amortización} = \textit{precio} \cdot \frac{\text{tiempo de uso}}{\text{periodo de amortización}} \tag{1.1}$$

El periodo de amortización establecido por el CTT es de 6 años, por lo tanto se obtiene el siguiente importe para gastos materiales.

Ítem	**Precio (€)**	**Uso (meses)**	**Amort. (años)**	**Importe (€)**
Ordenador portátil	600	6	6	50.00
Monitor externo	100	6	6	8.33
Matlab R2020b	3350	6	6	279.17
CoppeliaSim 4.1.0	0	6	6	0.00
			Total	**337.50**

Tabla 1.1: Gastos materiales

1.2.2 Gastos de personal

Los gastos de personal incluyen las diversas tareas realizadas durante el desarrollo del proyecto. Se asume un coste horario de 24.99 €, tomando el máximo valor del rango indicado por el CTT para titulados superiores contratados como personal eventual. Este importe incluye los gastos de seguridad social e indemnización.

Tarea	**€/Hora**	**Horas**	**Importe**
Estudio bibliográfico	24.99	20	499.80
Ensamblado CoppeliaSim	24.99	80	1999.20
Programación	24.99	30	749.70
Ensayos	24.99	5	124.95
Análisis de resultados	24.99	5	124.95
Redacción informe	24.99	40	999.60
Total			**4498.20**

Tabla 1.2: Gastos de personal

1.2.3 Presupuesto total

Para el cálculo del presupuesto total se incluyen los costes directos y los costes indirectos calculados como un 25% de los costes directos. Sobre este subtotal se aplica el 21% de IVA.

Categoría	**Subtotal (€)**	**Importe (€)**
Gastos materiales	337.50	
Gastos de personal	4498.20	
Gastos directos		**4835.70**
Gastos indirectos		**1208.93**
		6044.63
IVA		**1269.37**
Total		**7314.00**

Tabla 1.3: Presupuesto total

El presupuesto total del proyecto asciende a la cantidad de: **SIETE MIL TRESCIENTOS CATORCE EUROS**.

Parte III

ANEXOS

Anexo A

Propiedades dinámicas

A.1 Plataforma móvil

Peso (Kg)	8.5558		
CdM (m)	X = 0.0494	Y = -0.0003	Z = 0.038
Tensor de inercias (Kg·m2)	Lxx = 0.09251	Lxy = 0.00108	Lxz = -0.0012
	Lyx = 0.00108	Lyy = 0.3504	Lyz = 0.00007
	Lzx = -0.0012	Lyz = 0.00007	Lzz = 0.43934
Desplazamiento r (m)	X = 0	Y = -0.245259	Z = 0
CdM CoopeliaSim (m)	X = 0.0494	Y = -0.003	Z = 0.01674
Giros CoppeliaSim (º)	α = -0.0426	β = -0.198	γ = -0.24
Inercias CoppeliaSim (m2)	Ix = 0.01081	Iy = 0.04096	Iz = 0.05135

Tabla A.1: Propiedades dinámicas plataforma móvil

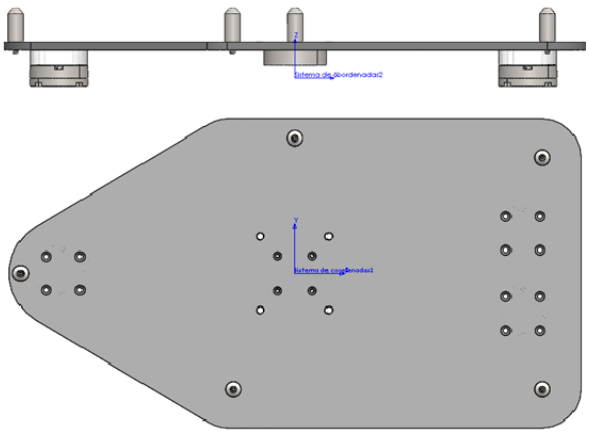


Figura A.1: Plataforma móvil

A.2 Parte inferior patas exteriores

Peso (Kg)	4.105		
CdM (m)	X = -0.03638	Y = -0.03499	Z = 0.19185
Tensor de inercias (Kg·m2)	Lxx = 0.08822	Lxy = 0.00397	Lxz = 0.00523
	Lyx = 0.00397	Lyy = 0.08895	Lyz = 0.00494
	Lzx = 0.00523	Lyz = 0.00494	Lzz = 0.0105
Desplazamiento r (m)	X = 0	Y = 0	Z = -0.333209
CdM CoopeliaSim (m)	X = -0.3638	Y = -0.03499	Z = -0.1414
Giros CoppeliaSim (º)	α = 3.4	β = -3.43	γ = 47.5
Inercias CoppeliaSim (m2)	Ix = 0.00269	Iy = 0.002061	Iz = 0.002414

Tabla A.2: Propiedades dinámicas parte inferior patas exteriores

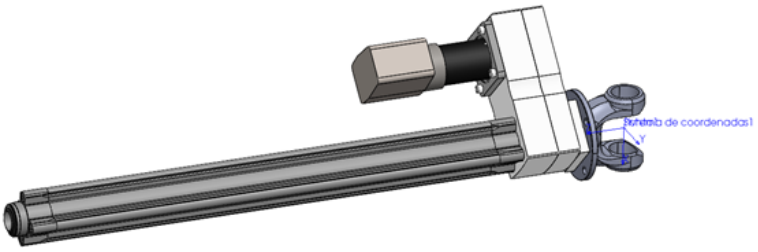


Figura A.2: Parte inferior patas exteriores

A.3 Parte superior patas exteriores

Peso (Kg)	1.262		
CdM (m)	X = 0	Y = 0	Z = -0.17851
Tensor de inercias (Kg·m2)	Lxx = 0.04441	Lxy = 0	Lxz = 0
	Lyx = 0	Lyy = 0.04441	Lyz = 0
	Lzx = 0	Lyz = 0	Lzz = 0.00007
Desplazamiento r (m)	X = 0	Y = 0	Z = 0.285738
CdM CoopeliaSim (m)	X = 0	Y = 0	Z = 0.1072
Giros CoppeliaSim (º)	α = 0.000000849	β = 0.000000429	γ = 0
Inercias CoppeliaSim (m2)	Ix = 0.03519	Iy = 0.03519	Iz = 0.00005547

Tabla A.3: Propiedades dinámicas parte superior patas exteriores

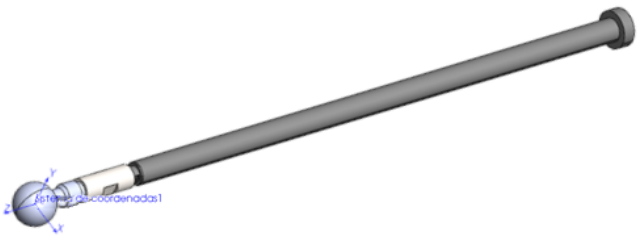


Figura A.3: Parte superior patas exteriores

A.4 Parte inferior pata central

Peso (Kg)	5.589		
CdM (m)	X = 0.0006	Y = 0.011	Z = 0.2575
Tensor de inercias (Kg·m2)	Lxx = 0.1889	Lxy = 0.0002	Lxz = -0.0003
	Lyx = 0.0002	Lyy = 0.1856	Lyz = -0.0082
	Lzx = -0.0003	Lyz = -0.0082	Lzz = 0.0.0067
Desplazamiento r (m)	X = 0	Y = 0	Z = -0.305815
CdM CoopeliaSim (m)	X = 0.0006003	Y = 0.011	Z = -0.04831
Giros CoppeliaSim (º)	α = -2.62	β = 0.0912	γ = 4.15
Inercias CoppeliaSim (m2)	Ix = 0.0338	Iy = 0.03327	Iz = 0.001132

Tabla A.4: Propiedades dinámicas parte inferior pata central

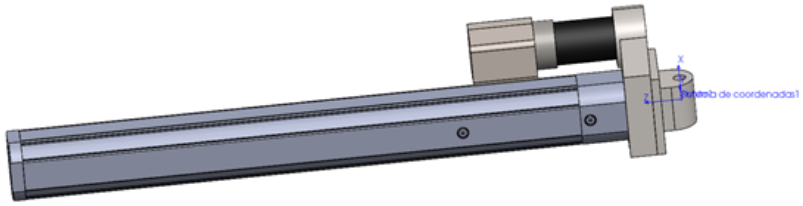


Figura A.4: Parte inferior pata central

A.5 Parte superior pata central

Peso (Kg)	1.845		
CdM (m)	X = 0	Y = 0.1922	Z = 0
Tensor de inercias (Kg·m2)	Lxx = 0.007	Lxy = 0	Lxz = 0
	Lyx = 0	Lyy = 0.0001	Lyz = 0
	Lzx = 0	Lyz = 0	Lzz = 0.007
Desplazamiento r (m)	X = 0	Y = -0.2452590466	Z = 0
CdM CoopeliaSim (m)	X = 0	Y = -0.05306	Z = 0
Giros CoppeliaSim (º)	α = 0.000000849	β = 0.000000429	γ = 0
Inercias CoppeliaSim (m2)	Ix = 0.001705	Iy = 0.00002436	Iz = 0.003794

Tabla A.5: Propiedades dinámicas parte superior pata central

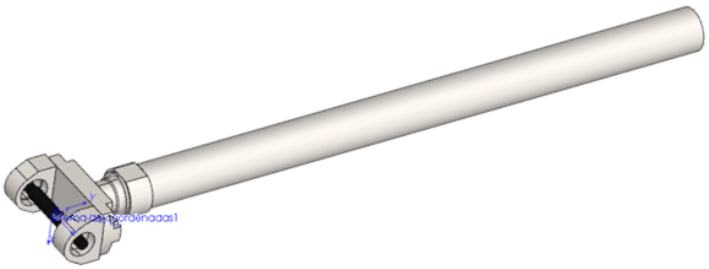


Figura A.5: Parte superior pata central

Anexo B

Manual de programación

B.1 Selección del modo de operación del modelo cinemático

La selección del modo de operación directo e inverso del modelo cinemático en CoppeliaSim se realiza mediante un *script* asociado a la base del robot. Este *script* escrito en LUA, se compone de las funciones presentadas a continuación.

La función *setFkMode* desactiva las restricciones posición y configura las articulaciones en modo pasivo para que se pueda especificar su coordenada generalizada.

```
function setFkMode()
    -- Desactivar las restricciones de posicion
    sim.setIkElementProperties(mainIkTask,ikModeTipDummy,0)
    -- Configurar las articulaciones en modo pasivo
    sim.setJointMode(fkDrivingJoints[1],sim.jointmode_passive,0)
    sim.setJointMode(fkDrivingJoints[2],sim.jointmode_passive,0)
    sim.setJointMode(fkDrivingJoints[3],sim.jointmode_passive,0)
    sim.setJointMode(fkDrivingJoints[4],sim.jointmode_passive,0)
end
```

La función *setIkMode* activa las restricciones posición y establece las articulaciones activas en modo cinemática directa para que su valor se calcule automáticamente.

```
function setIkMode()
    -- Activar las restricciones de posicion
    sim.setIkElementProperties(mainIkTask,ikModeTipDummy,sim.
        ↪ ik_x_constraint+sim.ik_z_constraint+sim.ik_alpha_beta_constraint)
    -- Configurar las articulaciones en modo cinematica inversa
    sim.setJointMode(fkDrivingJoints[1],sim.jointmode_ik,0)
    sim.setJointMode(fkDrivingJoints[2],sim.jointmode_ik,0)
    sim.setJointMode(fkDrivingJoints[3],sim.jointmode_ik,0)
    sim.setJointMode(fkDrivingJoints[4],sim.jointmode_ik,0)
end
```

La función *sysCall_ init* permite crear el grupo de cinemática inversa y obtener los manejadores de las articulaciones activas del robot. Esta función se invoca de forma automática cuando arranca la simulación.

```
function sysCall_init()
    -- Creacion del vinculo de cinematica inversa
    mainIkTask=sim.getIkGroupHandle('IK_Group')
    ikModeTipDummy=sim.getObjectHandle('Dummy_ikTip')
    -- Obtencion de los manejadores de las articulaciones activas
    fkDrivingJoints={-1,-1,-1,-1}
    fkDrivingJoints[1]=sim.getObjectHandle('q13')
    fkDrivingJoints[2]=sim.getObjectHandle('q23')
    fkDrivingJoints[3]=sim.getObjectHandle('q33')
    fkDrivingJoints[4]=sim.getObjectHandle('q42')
    -- Seleccion del modo cinematica directa
    setFkMode()
end
```

B.2 Cliente TCP/IP para reproducción de movimientos

El cliente para la aplicación de replicación de movimientos del robot real se desarrolla en C partiendo de una serie de librerías que se pueden encontrar en los archivos de instalación de CoppeliaSim. El código queda dividido en un archivo de cabecera y en un archivo fuente.

El archivo de cabecera, *clientApi.h*, contiene las declaraciones de las funciones que permiten iniciar y detener el cliente y por otro lado, actualizar la extensión de las patas del robot simulado.

```
#include <stdbool.h>

#define NUM_JOINTS 4

const float q1_min = 0.763;
const float q2_min = 0.763;
const float q3_min = 0.763;
const float q4_min = 0.6985;

// Inicia un cliente y se conecta al servidor de CoppeliaSim
bool start_coppelia_client(char *ip, int *clientID, int *h);

// Finaliza la comunicación con el servidor
void stop_coppelia_client(int clientID, bool block);

// Actualiza las posiciones de los actuadores del robot
void set_joint_pos_coppelia(int clientID, int *h, float q1, float q2, float
    ↪  q3, float q4);
```

El archivo fuente, *clientApi.c*, define las funciones declaradas en la cabecera. En primer lugar, se tiene la función *start_ coppelia_ client* que recibe como parámetros un array con la dirección IP de la máquina donde se está ejecutando el servidor de CoppeliaSim, un puntero a la variable que almacena el identificador del cliente y un puntero a un vector que almacena los manejadores de las articulaciones activas. La función comprueba si se ha podido establecer una conexión y obtener los manejadores deseados. En caso afirmativo muestra un mensaje en la consola de CoppeliaSim y devuelve el resultado de la ejecución.

```
#include "expApi.h"
#include "clientApi.h"

bool start_coppelia_client(char *ip, int *clientID, int *h) {
    *clientID=simxStart((simxChar*)ip,19997,true,true,2000,5);
    if (*clientID!=-1) {
        char c[][NUM_JOINTS] = {"q13","q23","q33","q42"};
        // Creamos los manejadores
        for (int i = 0; i < NUM_JOINTS; i++) {
            if (simxGetObjectHandle(*clientID, c[i], &h[i],
                ↪ simx_opmode_blocking) != simx_return_ok) {
            return false;
        }
        // Mandamos un mensaje a la consola de CoppeliaSim
        simxAddStatusbarMessage(*clientID,"Cliente conectado",
            ↪ simx_opmode_oneshot);
        return true;
    }
    return false;
}
```

Seguidamente, se implementa la función *stop_coppelia_client* que recibe como argumentos de entrada el identificador del cliente y una variable que permite especificar si se desea esperar a que finalice la transmisión de la última petición. Esta función manda un mensaje a la consola de CoppeliaSim y detiene el cliente.

```
void stop_coppelia_client(int clientID, bool block) {
    // Mandamos un mensaje a la consola de CoppeliaSim
    simxAddStatusbarMessage(clientID,"Cliente desconectado",
        ↪ simx_opmode_oneshot);
    if (block) {
        // Nos aseguramos que el último comando se ha recibido
        int pingTime;
        simxGetPingTime(clientID,&pingTime);
    }
        // Cerramos la sesión
        simxFinish(clientID);
}
```

Finalmente, se presenta la función *set_joint_pos_coppelia* que toma como argumentos el identificador del cliente, el puntero con los manejadores y los cuatro valores de los actuadores. Esta función pausa la comunicación, acumula las las nuevas posiciones de los actuadores en una cola y después reanuda la comunicación enviando todas las peticiones a la vez.

```
void set_joint_pos_coppelia(int clientID, int *h, float q1, float q2, float
    ↪  q3, float q4) {
    // Pausamos la comunicacion
    simxPauseCommunication(clientID,1);
    // Mandamos las posiciones restando la longitud minima de cada actuador
    simxSetJointPosition(clientID, h[0], q1 - q1_min, simx_opmode_oneshot);
    simxSetJointPosition(clientID, h[1], q2 - q2_min, simx_opmode_oneshot);
    simxSetJointPosition(clientID, h[2], q3 - q3_min, simx_opmode_oneshot);
    simxSetJointPosition(clientID, h[3], q4 - q4_min, simx_opmode_oneshot);
    // Reanudamos la comunicacion
    simxPauseCommunication(clientID,0);
}
```

B.3 Nodo BlueZero para co-simulación

Para la co-simulación de CoppeliaSim junto con Simulink se implementa un nodo BlueZero en el lenguaje propio de Matlab de forma que se pueda realizar la comunicación entre ambos programas. En este caso se implementan tres funciones principales para controlar la simulación y una serie de funciones auxiliares.

La función *startCoppeliaSimB0* inicia un nuevo nodo y trata de establecer una conexión con el bróker BlueZero. En caso afirmativo, se obtienen los manejadores necesarios, se establece el modo de simulación síncrona y se configuran las articulaciones activas en modo par/fuerza. Seguidamente, se añade una función de tipo *callback* que se ejecutará en cada iteración y se inicia la simulación.

```
function startCoppeliaSimB0()
    global client doNextStep fixed_ref mobile_ref plat_ref joints
        ↪ legs_min_len;
    doNextStep=true;
    legs_min_len = [0.763, 0.763, 0.763, 0.6985]; % Metros
    disp('Program started');
    try
        % Nos conectamos al broker BlueZero y obtenemos un cliente
        client=b0RemoteApi('b0RemoteApi_matlabClient','b0RemoteApi');
        % Mandamos un mensaje a la consola de CoppeliaSim
        client.simxAddStatusbarMessage('Cliente conectado',client.
            ↪ simxDefaultPublisher());

        % Obtenemos los manejadores necesarios
        q1 = client.simxGetObjectHandle('q13',client.simxServiceCall());
        q2 = client.simxGetObjectHandle('q23',client.simxServiceCall());
        q3 = client.simxGetObjectHandle('q33',client.simxServiceCall());
        q4 = client.simxGetObjectHandle('q42',client.simxServiceCall());
        joints = [q1{2}, q2{2}, q3{2}, q4{2}];
        f_ref = client.simxGetObjectHandle('Referencia_fija',client.
            ↪ simxServiceCall());
        fixed_ref = f_ref{2};
        m_ref = client.simxGetObjectHandle('Referencia_movil',client.
            ↪ simxServiceCall());
        mobile_ref = m_ref{2};
        p_ref = client.simxGetObjectHandle('Referencia_plataforma',client.
            ↪ simxServiceCall());
        plat_ref = p_ref{2};
        % Activamos la simulacion sincrona
        client.simxSynchronous(true);

        % Desactivamos el bucle de control de posición de las patas
        for i = 1:length(joints)
            client.simxSetObjectIntParameter(joints(i), 2001, 0, client.
                ↪ simxServiceCall());
        end
        % Vinculamos un callback a la finalizacion de un paso de simulacion
        client.simxGetSimulationStepDone(client.simxDefaultSubscriber(
            ↪ @stepEndCallback));
        % Iniciamos la simulacion
        client.simxStartSimulation(client.simxServiceCall());
    catch me
        client.delete();
        rethrow(me);
    end
end
```

La función *stopCoppeliaSimB0* detiene la simulación y destruye el nodo BlueZero.

```
function stopCoppeliaSimB0()
    global client;
    try
        client.simxStopSimulation(client.simxDefaultPublisher());
        client.delete();
    catch me
        client.delete();
        rethrow(me);
    end
    disp('Program ended');
end
```

La función de tipo *callback stepEndCallback* se ejecuta cuando termina cada paso de simulación en CoppeliaSim. Esta función simplemente actualiza una variable para indicar que se puede proseguir con la ejecución del resto del programa.

```
function simulationStepDone_CB(data)
    global doNextStep;
    doNextStep=true;
end
```

La función *stepCoppeliaSimB0* se ejecuta periódicamente y se encarga de aplicar las acciones de control, esperar a que concluya el paso de simulación en CoppeliaSim y después leer las diferentes medidas.

```
function y = stepCoppeliaSimB0force(joint_forces)
    global client doNextStep fixed_ref mobile_ref joints legs_min_len;

    % Aplicamos la acción de control
    setJointForces(client, joints, joint_forces');

    % Esperamos a que se finalize el paso de simulación en CoppeliaSim
    client.simxSynchronousTrigger();
    while doNextStep==false
        client.simxSpinOnce();
    end
    doNextStep=false;

    % Tomamos las distintas medidas
    platform_pos_orien = getPlatformPositionOrientation(client, mobile_ref,
        ↪ fixed_ref)';
    joint_positions = (getJointPositions(client, joints) + legs_min_len)';
    joint_velocities = getJointVelocities(client,joints)';

    y = [platform_pos_orien; joint_positions; joint_velocities];
end
```

En el código anterior se realizan llamadas a distintas funiones auxiliares que permiten interactuar con el robot de la simulación que se ejecuta en CoppeliaSim.

La función *setJointForces* toma como argumentos de entrada el cliente BlueZero, las articulaciones activas y la acción de control. En primer lugar esta función actualiza la velocidad del actuador a un valor muy grande con el mismo signo que la fuerza que se desea aplicar y luego establece la fuerza máxima que puede realizar el controlador en base al valor absoluto de la acción de control.

```
function setJointForces(client, joints, u)
    for i = 1:length(joints)
        client.simxSetJointTargetVelocity(joints(i), 10000 * sign(u(i)),
            ↪ client.simxServiceCall());
        client.simxSetJointMaxForce(joints(i), abs(u(i)), client.
            ↪ simxServiceCall());
    end
end
```

Mediante la funciones *getJointPositions* y *getJointVelocities* se obtienen las posiciones y velocidades de los actuadores respectivamente.

```
function pos = getJointPositions(client, joints)
    for i = 1:length(joints)
        r = client.simxGetJointPosition(joints(i), client.simxServiceCall()
            ↪ );
        pos(i) = r{2};
    end
end
```

```
function vel = getJointVelocities(client, joints)
    for i = 1:length(joints)
        r = client.simxGetObjectFloatParameter(joints(i), 2012, client.
            ↪ simxServiceCall());
        vel(i) = r{2};
    end
end
```

En último lugar, la función *getPlatformPositionOrientation* permite obtener la posición de la plataforma en los ejes X y Z y las rotaciones alrededor de Y y Z.

```
function pos_orien = getPlatformPositionOrientation(client, p_ref, m_ref,
    ↪ f_ref)
    r = client.simxGetObjectPosition(p_ref, f_ref, client.simxServiceCall()
        ↪ );
    pos = cell2mat(r{2});

    r = client.simxGetObjectOrientation(p_ref, f_ref, client.
        ↪ simxServiceCall());
    orien_theta = cell2mat(r{2});

    r = client.simxGetObjectOrientation(p_ref, m_ref, client.
        ↪ simxServiceCall());
    orien_psi = cell2mat(r{2});

    pos_orien = [pos([1 3]) orien_theta(2) orien_psi(3)];
end
```