Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

APLICACIÓN DE SIMULINK Y MATLAB PARA ANÁLISIS CINEMÁTICO Y DINÁMICO, CONTROL Y COMUNICACIONES DE ROBOTS INDUSTRIALES

Autor:

Arévalo Fernández, Sandra

Tutor:

Herreros López, Alberto

Departamento de Ingeniería de Sistemas y Automática

Valladolid, junio de 2020.

AGRADECIMIENTOS

En primer lugar, quiero agradecer su ayuda y dedicación a mi tutor del Trabajo Fin de Grado, Alberto Herreros López.

En segundo lugar, gracias a mi familia por el apoyo y paciencia recibida, por confiar en mí de principio a fin. No hay palabras suficientes para mostrar mi agradecimiento hacia vosotros.

Y, por último, gracias Mari, has sido la mejor compañera de camino que podría haber tenido durante estos años.

RESUMEN

En el presente Trabajo Fin de Grado se plantea el proceso de desarrollo y validación de un software destinado a su implementación como herramienta educativa en el entorno de la robótica. Este ha sido desarrollado a través de MATLAB y SIMULINK, entorno ampliamente conocido cuyas características le presentan como una poderosa herramienta para la creación de aplicaciones orientadas al estudio de los robots manipuladores.

Con esta aplicación se pretende poner al alcance del alumno la posibilidad de poner en práctica las bases teóricas de la robótica, fomentando el diseño de una estación robotizada a través de la creación de una librería particular, así como la realización del estudio cinemático y dinámico de esta a través de su programación y simulación, estudiando las relaciones y transformaciones de cuerpos que tienen lugar.

Además, se establecerá la comunicación OPC a través de RobotStudio para que el alumno pueda comprobar la interacción entre el robot virtual y el robot real.

PALABRAS CLAVE

MATLAB, SIMULINK, robótica, simulación, OPC.

ABSTRACT

In this Final Degree Project the process of development and validation of a software used as a teaching tool oriented towards the filed of robotics is presented. The work has been carried out with MATLAB and SIMULINK, which is a widely known work environment whose features lead it to be a powerful force in order to create robotics applications.

This software application aims to provide students with the possibility of putting theoretical foundations of robotics into practise, promoting the design of a robotic station through special library creation, studying its kinematics and dynamics via programming and simulation, and analysing the relations and transformations of bodies that take place.

Moreover, the OPC communication will be established through RobotStudio for the purpose of allowing students to test the interaction between virtual robot and real robot.

KEY WORDS

MATLAB, SIMULINK, robotics, simulation, OPC.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN, OBJETIVOS Y ESTADO DEL ARTE

1.1 INTRODUCCIÓN

Desde que en 1948 saliera a la luz el primer robot manipulador, el robot *Unimate*, la evolución de los robots industriales ha sido vertiginosa. Con el paso del tiempo, esta misma patente fue evolucionando hacia nuevos prototipos que se iban adaptando a las mejoras tecnológicas. *Unimate* consiguió convertirse en el robot programable conocido como *PUMA,* el cual representa hoy las bases de la mayoría de los elementos electrónicos generados dentro del mundo de la robótica.

La investigación y desarrollo de la robótica industrial ha dado lugar a que los robots tomen posiciones en casi todas las áreas productivas y tipos de industria. El impulso de este sector ha generado que aumente de forma significativa la implantación de robots industriales en diversos procesos. Esto conlleva a que sea requerido un estudio previo del sistema robotizado en cuestión.

La programación y simulación fuera de línea supone una gran ventaja puesto que permite analizar, comprobar y corregir el comportamiento de los sistemas.

Actualmente, la programación de robots hace uso de herramientas de simulación que permiten reproducir la dinámica del robot, ya sea para preparar al personal que lo utiliza previo a su operación, así como para eliminar movimientos erróneos antes de su implementación.

En cuanto al ámbito educativo, en los últimos años se han desarrollado softwares que proporcionan aplicaciones orientadas a la robótica, permitiendo simular los movimientos del robot en un entorno gráfico. Las materias relacionadas con la robótica tienen como objetivo instruir a los alumnos sobre las bases teóricas, así como aprender sobre la programación de un robot a través de distintos lenguajes.

A lo largo de los años se han ido desarrollando distintas propuestas para satisfacer ambos objetivos, entre ellas, el uso de aplicaciones de MATLAB y SIMULINK, y otras desarrolladas particularmente para robótica. Estos dos primeros entornos suponen unas herramientas muy poderosas para el estudio de la robótica. Es por ello, que el presente proyecto se ha centrado en ambos para el desarrollo de esta nueva aplicación.

1.2 OBJETIVOS

Este proyecto está concebido desde un principio con fines didácticos, siendo el objetivo principal el desarrollo de una aplicación que posibilite el estudio de los sistemas robóticos. Esta aplicación se plantea como el resultado de la fusión de librerías de MATLAB y SIMULINK, permitiendo complementar entre sí los mejores aspectos de cada una de ellas. Se busca, principalmente, que el software cuente con la capacidad de cálculo que presenta la *Robotic Toolbox* de MATLAB, así como con el entorno de simulación proporcionado por la herramienta *Simscape Multibody* en SIMULINK.

Con este objetivo principal en mente, se han ido desarrollando distintos objetivos intermedios que han ido dando cuerpo a la aplicación desarrollada, verificando su utilidad como herramienta didáctica:

- Crear una librería de iconos para el entorno SIMULINK compuesta por una serie de bloques que representen los elementos propios de una estación robótica: robots manipuladores, objetos de trabajo, cargas y herramientas. De manera que se ponga al alcance del alumno la posibilidad de crear su propio modelo de estación mediante la incorporación de bloques y la definición de sus parámetros.

- Consolidar las herramientas de localización espacial de un sólido rígido que permitan entender al alumno las relaciones y transformaciones de cuerpos que tienen lugar en el desarrollo de los modelos cinemáticos y dinámicos de los robots móviles.

- Comprobar y desarrollar el estudio de la cinemática directa e inversa de los robots, analizando los valores tomados por las coordenadas articulares en relación con la posición y orientación del extremo final de este, y viceversa. Para ello, se busca analizar la herramienta que permite sincronizar las librerías; objeto *Kin*.

- Estudiar la relación entre las fuerzas aplicadas sobre el robot y el movimiento de este, analizando la influencia que supone la incorporación de un sistema de control sobre el modelo.

- Comprobar el software propuesto mediante la comunicación OPC entre el servidor real ABB, representado con RobotStudio, y el cliente de MATLAB.

1.3 ESTADO DEL ARTE

Este apartado tiene especial interés en el presente trabajo puesto que en él se van a mostrar las propuestas desarrolladas a lo largo de los años como herramientas de aprendizaje en el ámbito de la robótica para MATLAB y SIMULINK, las cuales han impulsado el desarrollo del software expuesto.

Como veremos, estas permiten moldear y simular robots, siendo algunas de ellas genéricas, válidas para cualquier tipo de robot definido a través de un método sistemático (Denavit-Hartenberg), y otras basadas en archivos de visualización para el estudio de la cinemática y dinámica de robots concretos.

También se explicará el entorno de simulación RobotStudio, puesto que será utilizado a lo largo del proyecto para la verificación de la comunicación con el robot real, y se hablará del conocido entorno ROS.

1.3.1 MATLAB & SIMULINK

Cuando se habla de MATLAB [1] y SIMULINK [2], se habla de dos marcos de cálculo numérico y visualización de datos pertenecientes a la compañía MathWorks. Estos presentan amplias posibilidades relacionadas con el diseño de sistemas dinámicos y su simulación.

MATLAB es una herramienta de software matemático de desarrollo integrado (IDE) que presenta un lenguaje de programación propio, el lenguaje M. Este, junto a las herramientas y funciones incorporadas, permite una gran flexibilidad y facilidad a la hora de representar datos, implementar algoritmos, crear interfaces de usuario, manipular matrices…

Figura 1: Icono de MATLAB

Debido a los cálculos y visualizaciones gráficas de alta dimensión y resolución permitidas, MATLAB se posiciona como una potente herramienta en cuanto al desarrollo de aplicaciones orientadas a la robótica. A pesar de que esta posee infinidad de usos y de herramientas adicionales, se profundizará en aquellas que resulten más interesantes para este proyecto.

El paquete MATLAB dispone de una toolbox especial de gran relevancia y aplicación: SIMULINK. Esta sirve para simular el comportamiento de los sistemas dinámicos; puede realizar la simulación de sistemas lineales y no lineales, así como de modelos en tiempo continuo y discreto, y, sistemas híbridos de todos ellos. Es un entorno gráfico en el que el modelo a simular se construye mediante la incorporación de los diferentes bloques de las librerías que presenta, las cuales se han ido ampliando con las nuevas versiones. Además, algunas toolboxes de MATLAB también incorporan bloques de SIMULINK. Los modelos SIMULINK son guardados en ficheros con extensión *.mdl.

Cuando MATLAB y SIMULINK son utilizados conjuntamente, se está combinando programación textual y gráfica para diseñar su sistema en un entorno de simulación.

Respecto al diseño y simulación de sistemas robotizados, diversos autores han desarrollado aplicaciones de software libre usando MATLAB y SIMULINK.

1.3.1.1 *Robotic Toolbox* de Peter Corke

La herramienta más extendida en el estudio de la robótica en MATLAB es la llamada "Robotic Toolbox" creada por Peter Corke [3], en la cual se exponen los contenidos del libro del autor [4]. Esta librería es increíblemente útil para el estudio de sistemas robóticos de no demasiada dificultad desde MATLAB. Incorpora cerca de 400 funciones de todo tipo para el estudio de los robots, las cuales se van actualizando, mejorando y, también, añadiendo nuevas según se publican actualizaciones de la librería.

Esta toolbox permite la definición de robots manipuladores mediante la representación de una serie de articulaciones sucesivas, llamadas *links*, unidas a través de segmentos en formas de barra, mediante estructuras *serial-link*. Además, facilita la manipulación de datos en forma de vectores, transformaciones homogéneas y cuaternios, los cuales participan en la representación de la localización espacial.

La representación cinemática y dinámica de los robots se lleva a cabo mediante el método de Denavit-Hartenberg (D-H), definiendo sucesivamente las coordenadas de las articulaciones que lo forman respecto de la anterior. Los parámetros D-H están encapsulados en objetos MATLAB. El usuario puede crear objetos de robot para cualquier manipulador *serial-link.*

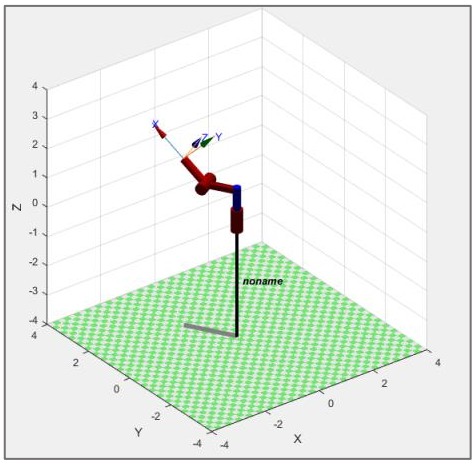


Figura 2: Estructura robótica definida mediante Robotic Toolbox de Peter Corke

La representación gráfica de los modelos es muy simple y genérica, ya que como vemos en la *figura 2*, solo se representan los ejes y uniones entre ellos.

1.3.1.2 Librería *ARTE*

Figura 3: Icono de la librería ARTE

ARTE (*A Robotic Toolbox for Education*) es una librería para MATLAB orientada a la docencia de robots manipuladores, creada en la Universidad Miguel Hernández de Elche (Alicante) por Arturo Gil [5].

Esta aplicación expone el comportamiento cinemático de los robots manipuladores mediante la representación D-H. Además, permite evaluar la dinámica de estos, proporcionando datos dinámicos de muchos de ellos. También posibilita la planificación de trayectorias y la programación robótica en un lenguaje industrial, pudiéndose utilizar para la simulación de trayectorias de cualquier robot industrial ABB cuando es programado en RAPID.

Como característica que vamos a destacar en esta aplicación, es que presenta gran variedad de modelos 3D de robots, lo que sí que permite en este caso la representación realista de los eslabones del robot como objetos sólidos. La información gráfica de cada brazo del robot es proporcionada a través de ficheros con extensión *.stl.

1.3.1.3 *Robotics System Toolbox* de MATLAB

A partir de la versión 2015a de MATLAB, se incorpora la llamada "Robotics System Toolbox" [6]. Esta librería, propia del software, proporciona herramientas y algoritmos que permiten el estudio de la cinemática y dinámica mediante una representación del robot como árbol de cuerpo rígido, *RigidBody Tree*, permitiendo la comprobación de colisiones y generación de trayectorias.

Esta toolbox también proporciona algoritmos y conectividad con hardware para el desarrollo de aplicaciones autónomas de robótica móvil. Posibilita la representación de mapas, planificación de trayectorias, el seguimiento de rutas y control de movimiento. También incluye una biblioteca de modelos de robots industriales disponibles comercialmente que pueden ser importados, visualizados y simulados. En este proyecto se va a destacar por su capacidad de cálculo, ya que el entorno de simulación no es el más favorable.

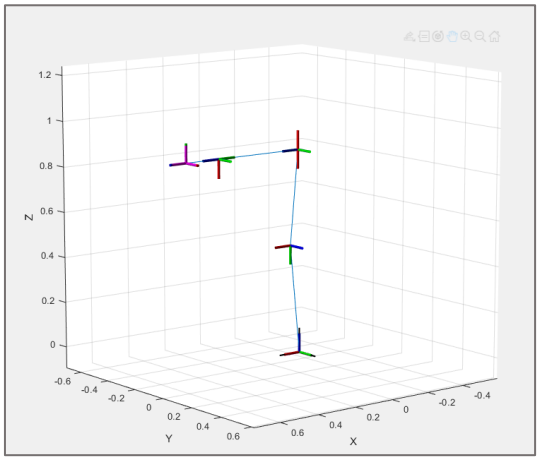


Figura 4: Estructura de robot RigidBody Tree

1.3.1.4 *Simscape Multibody* de SIMULINK.

Simscape Multibody consiste en un entorno de simulación para sistemas mecánicos en 3D [7]. Permite modelar sistemas multicuerpo, '*Multibody*', utilizando bloques que representan cuerpos, uniones, restricciones, articulaciones, elementos de fuerzas y sensores.

El programa permite importar archivos CAD que se presentan en la simulación como cuerpos sólidos, donde la información de la estructura está en ficheros de extensión *.xml y la información sobre su geometría en ficheros de extensión *.stl. Simscape Multibody reconoce la geometría del modelo importado y puede matematizar y resolver las ecuaciones sobre cuál sería su comportamiento en el entorno de simulación.

Además, permite cambiar parámetros como la densidad, peso, rozamiento o color de los cuerpos.

Con esta información, la aplicación genera una animación en 3D, que se refleja en la herramienta 'Mechanics Explorer', donde se puede visualizar la dinámica del sistema. Al contrario que la librería anterior, destaca por su entorno de simulación, siendo su capacidad de cálculo muy desfavorable.

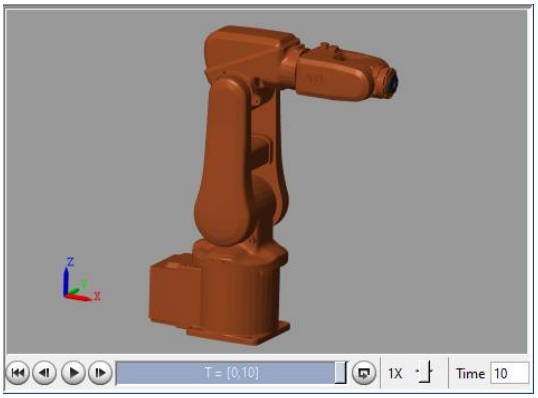


Figura 5: Animación de robot obtenido con Simscape Multibody

1.3.1.5 Otras aplicaciones

Aunque el desarrollo del proyecto ha tomado como base las librerías anteriormente descritas destinadas a la robótica, también cabe mencionar la existencia de otras dos herramientas particulares de MATLAB las cuales se han utilizado para mejorar y completar el estudio de la aplicación desarrollada.

- **App Designer:**

  Se trata de un entorno de desarrollo interactivo [8], integrado totalmente en el editor de MATLAB, que permite el diseño de aplicaciones y la programación de su comportamiento. Cuenta con un gran conjunto de componentes interactivos, entre ellos, se proporcionan componentes comunes, como ejes para crear gráficos o botones y controles deslizantes que responden a interacciones, así como contenedores y herramientas como paneles y pestañas que ayudan a administrar el diseño de la interfaz. También presenta instrumentos para visualizar estados como indicadores y lámparas.

  Está basado en el uso de devoluciones de llamada, Callabacks. Una Callback es una función que se ejecuta cuando se interactúa con un componente en la aplicación. La mayoría de los componentes pueden tener al menos una devolución de llamada, aunque algunos como las etiquetas y lámparas no las presentan puesto que solo muestran información.

- **OPC Toolbox:**

El software OPC Toolbox [9] aplica un enfoque jerárquico orientado a objetos que posibilita la comunicación de Matlab con los servidores OPC utilizando los estándares Historical Data Access (HDA) y OPC Data Access (DA). Permite trabajar con datos de servidores en tiempo real e historiadores de datos que cumplen estos estándares, siendo posible leer, escribir o registrar datos OPC de dispositivos como sistemas de control distribuido, de supervisión y adquisición de datos, o controladores lógicos programables.

Proporciona una conexión segura a través de distintos algoritmos, métodos de seguridad y modos de autenticación.

1.3.2 ROBOTSTUDIO

La aplicación desarrollada en el presente trabajo se ha llevado a cabo mediante MATLAB y SIMULINK, sin embargo, se ha utilizado el entorno de simulación RobotStudio [10] para la verificación de la comunicación con el robot real como veremos más adelante.

El programa RobotStudio es otro software que permite la creación, programación y simulación de robots industriales, permitiendo analizar su comportamiento, diseñado y patentado por la empresa ABB [11]. Utiliza el lenguaje RAPID.

RobotStudio permite trabajar con un controlador virtual; un controlador IRC5 que se ejecuta de manera local en el PC. Este es una copia exacta del software real usado en los robots en producción. Esto supone una gran ventaja puesto que la programación fuera de línea permite analizar, comprobar y corregir el comportamiento de los sistemas sin necesidad de parar la producción en la industria, de manera que se exporten los resultados obtenidos en simulación a la estación real.

El programa proporciona herramientas para aumentar la rentabilidad del sistema robótico mediante tareas como formación, programación y optimización. El beneficio radica en el hecho de que la planta real de trabajo no está expuesta a posibles fallos, lo cual evita posibles daños mecánicos, personales u otros tipos de daños que pudieran darse en una estación de trabajo real, además de proporcionar un arranque más rápido y transiciones más cortas.

Entre las diversas funcionalidades que proporciona, RobotStudio consta de una vista gráfica en 3D en la cual se pueden crear simulaciones del entorno, permite la importación de datos del formato CAD, la generación automática y optimización de trayectorias, también permite analizar el alcance del manipulador a determinadas posiciones o la detección de colisiones, entre otras muchas.

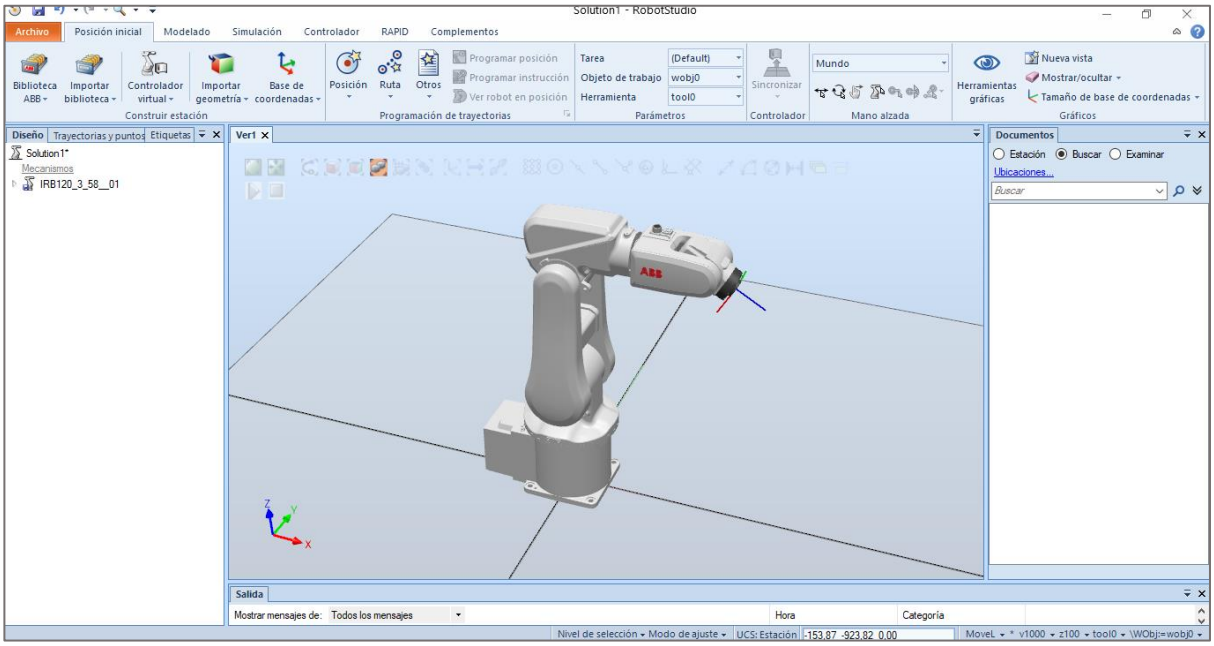


Figura 6: Entorno de trabajo de RobotStudio

1.3.2.1 Lenguaje de programación RAPID

Para que los robots industriales ABB realicen determinadas tareas es necesario que al software RobotStudio se le proporcione un programa donde se encuentren especificadas las instrucciones que queremos que realice el robot. Este programa se implementa con un lenguaje de programación de alto nivel llamado RAPID [12].

Este lenguaje permite llevar a cabo instrucciones tales como activar o desactivar salidas, leer entradas, tratar eventos o establecer la comunicación con otro operador. Una aplicación RAPID está compuesta por un programa y una serie de módulos del sistema.

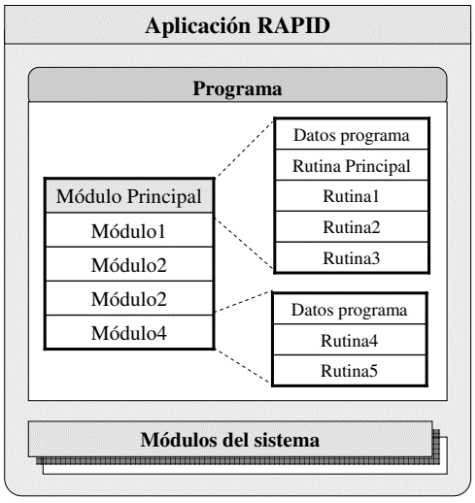


Figura 7:Estructura de una aplicación RAPID

Podemos encontrar dos tipos de módulos:

- Módulo de programación: es el módulo principal, el que contiene el procedimiento global llamado “main”. Cuando se ejecuta el programa, se ejecuta el procedimiento principal “main”, desde el cual se llama a las subrutinas. El programa puede presentar distintos módulos, pero sólo uno de ellos contiene el procedimiento principal.
- Módulo de sistema: es el módulo donde se guardan los datos y rutinas normales, como, por ejemplo, las herramientas o sistema de coordenadas mundo.

En general, las instrucciones llevan ligado un conjunto de argumentos que definen qué debe ocurrir con una instrucción determinada El programa se ejecuta de manera secuencial, es decir, una instrucción tras otra.

Se pueden distinguir entre tres tipos de rutinas:

- Procedimientos: son utilizados como subprogramas que no devuelven ningún valor.
- Funciones: son utilizadas como argumento de una instrucción devolviendo un valor de un tipo específico.
- Rutinas TRAP: proporcionan una forma de procesar las interrupciones. Puede asociarse a una determinada interrupción de tal manera que cada vez que esta ocurra durante la simulación se ejecute de manera automática.

RAPID permite almacenar información en datos, ya sea de manera global (se puede acceder a ellos desde distintos módulos del programa) o de manera local (el acceso se permite desde un único módulo). El número de datos está limitado sólo por la capacidad de la memoria utilizada.

Existen tres tipos de datos:

- Constantes: representan valores fijos, los cuales se asignan en el momento de la declaración y no se modifican de nuevo.
- Variables: tienen un valor asignado que puede ser modificado durante la ejecución del programa.
- Variables persistentes: su valor de inicialización es actualizado a medida que el programa es ejecutado. Al guardarse el programa, el valor de inicialización se corresponde con el valor actual de la variable persistente.

1.3.2.2 ABB IRC5 OPC Configuration

La aplicación ABB IRC5 OPC Server Configuration [13] es utilizada para crear y administrar alias para controladores de robot ABB IRC5. Un alias es conocido como un descriptor de uso fácil que representa una interfaz de comunicaciones para un controlador de robot ABB IRC5.

El número de controladores que pueden conectarse a un mismo servidor OPC viene limitado por el rendimiento del sistema, aunque ABB aconseja que no exceda el número máximo de 30 a una PC dedicada exclusivamente como servidor. Para poder configurarse se debe tener en cuenta que el robot y el ordenador con el cual se comunica deben estar ambos conectados a la misma red.

El servidor OPC hace uso del llamado '*report mode*' para publicar las variables OPC, lo que supone que cuando una variable o señal cambia de valor, este es reportado al cliente tan pronto como la consulte en el próximo tiempo de muestreo. Así, los clientes pueden incluso configurar un tiempo de muestreo de 0ms, notificando el servidor el cambio tan pronto como sea posible.

En este proyecto usaremos la comunicación OPC para establecer la conexión, a través de esta aplicación, entre un OPC cliente de Matlab y el servidor OPC ABB de RobotStudio.

1.3.3 ROS *(Robot Operating System)*

ROS se presenta como una colección de frameworks para el desarrollo de software de robots, siendo una fuente de librerías y herramientas [14]. Su uso no se limita solo a robots, pero la mayoría de las herramientas proporcionadas se enfocan en trabajar con hardware periférico. ROS proporciona un modo de conectar una red de procesos (nodos) con eje central. Los nodos se pueden ejecutar en dispositivos múltiples y se conectan a ese eje de diversas maneras.

Hoy en día es ampliamente usado en robótica, sobre todo en el marco de navegación de robots móviles. Ofrece una serie de características que lo hace único: permite que los procesos se ejecuten en diferentes computadoras, las cuales se interconectan en tiempo real siguiendo una topología de comunicación punto a punto, los desarrolladores pueden trabajar con ROS independientemente del lenguaje de programación utilizado, ofrece la posibilidad de reutilizar los códigos y controladores generados durante el desarrollo de sus proyectos y, para gestionar su complejidad, sus desarrolladores implementaron una gran cantidad de pequeñas herramientas que permiten compilar y ejecutar varios componentes de ROS.

1.3.4 ANTECEDENTES

En la Escuela de Ingenierías Industriales de la Universidad de Valladolid se han realizado varios trabajos hasta la fecha usando herramientas destinadas a los sistemas robotizados con el fin de su aplicación en la docencia.

Miguel Ángel Mato San José [15] realizó en 2014 su proyecto final de carrera en el cual se validaba el software *Robotic Toolbox* de Peter Corke de Matlab como herramienta de estudio, control y simulación de robots industriales, mediante el desarrollo de una interfaz gráfica y la comunicación OPC con el robot real.

En 2015, Juan Antonio Ávila Herrero [16] basó su proyecto en el modelado de una célula robótica compuesta por un robot ABB, IRB-120, con una serie de elementos adicionales destinados a la educación. Más tarde, en 2016, Álvaro Galindo de los Santos [17], también desarrolló el modelado de la célula robotizada utilizando RobotStudio, y llevó a cabo la comunicación a través de socket entre el robot y una tablet con sistema operativo Android. Ambos proyectos han dado lugar a la estación robótica con la que se cuenta actualmente en la escuela.

En 2019, Carlos Jiménez Jiménez [18] realizó su trabajo final de máster que consistía en el desarrollo de un sistema robótico educativo capaz de jugar al ajedrez con un robot industrial, empleando el protocolo TCP/IP para la comunicación del robot con Matlab. Otro trabajo fin de máster fue realizado en 2020 por Víctor Lobo Granado [19], en el cual también se planteaba el proceso de diseño, simulación y fabricación de entornos de trabajo para una estación robotizada orientada a la docencia, en el que ordenador y robot se comunicaban mediante un entorno grafico que permitía simular la estación con Simulink.

Además, la Escuela dispone en uno de sus laboratorios de una instalación formada por un robot ABB-IRB-120 y distintos elementos y entornos que permiten la realización de diversas prácticas. Profesores del Departamento de Automática han desarrollado también proyectos educativos sobre esta plataforma. El profesor, y tutor del presente proyecto, Alberto Herreros López, dotó al robot de la capacidad de escribir sobre un papel el texto leído de un fichero, pudiéndose emplear diversos tipos de letra y planos de escritura.

2. FUNDAMENTOS PREVIOS

2.1 CONCEPTOS BÁSICOS DE ROBOT MANIPULADOR

En robótica, el término *manipulador* se refiere a mecanismos creados con el fin de realizar tareas tales como desplazar y sostener objetos, lo cual define a los robots manipuladores como funcionales. Estos han sido creados con el objetivo de eliminar esfuerzos provocados por el levantamiento de mercancía y están destinados a diversos campos.

Los robots manipuladores son manejados de forma básica y sencilla por humanos en un dispositivo externo, siendo capaces de realizar las tareas indicadas por estos. Está compuesto por los siguientes elementos: estructura mecánica, trasmisiones, sistemas de accionamiento, sistema de control y sensorial, y herramientas terminales, *tools*. Su constitución física guarda cierto parecido con el brazo humano, por lo que para hacer referencias a sus distintos componentes se usan términos como cuerpo, brazo, codo y muñeca.

Algunas de las definiciones y conceptos a tener en cuenta de los robots manipuladores:

- Eslabón: componente físico que existe entre articulaciones.
- Articulación: puntos que ponen en contacto los eslabones. Pueden ser de tipo prismático, de revolución o esféricas.
- Tool Center Point (TCP): punto central de la herramienta del robot, define el alcance del robot.
- Área de trabajo de un robot: volumen espacial al que puede acceder el extremo del robot.
- Grado de libertad (GDL): indica cada uno de los movimientos independientes que puede realizar una articulación de un robot respecto a la anterior. Determina la accesibilidad de un robot y su capacidad para orientar sus herramientas, soliendo coincidir con el número de articulaciones del robot.
- Capacidad de carga: carga que es capaz de manipular el robot. Viene condicionada por el tamaño, la configuración y el sistema de accionamiento del robot, teniendo que considerar también en algunos casos los momentos de inercia.

- Resolución: mínimo incremento que puede aceptar la unidad de control del robot.

- Precisión: distancia media entre el punto programado y el punto realmente alcanzado, medida tras varios ciclos.

- Repetibilidad: precisión en la repetición de movimientos.

- Puntos singulares: puntos del espacio de trabajo del robot sobre los que no es posible realizar una trayectoria rectilínea.

En lo que respecta a su estructura mecánica, los robots manipuladores están compuestos por una serie consecutiva de eslabones y articulaciones que forman una cadena cinemática abierta. Estas articulaciones que unen los eslabones permiten el movimiento y la firmeza del brazo robótico, el cual es comparable con un brazo humano y es por ello por lo que esta útil herramienta actúa de su sustituto en distintas tareas en la industria.

La cadena cinemática abierta está formada por una primera articulación, la cual sirve para formar la base, seguida por conexiones sucesivas entre articulaciones y eslabones. En el extremo final del último eslabón no hay articulación, sino que está destinado a colocar la herramienta de trabajo para llevar a cabo la tarea especificada. El extremo final del robot no se encuentra conectado físicamente a la base.

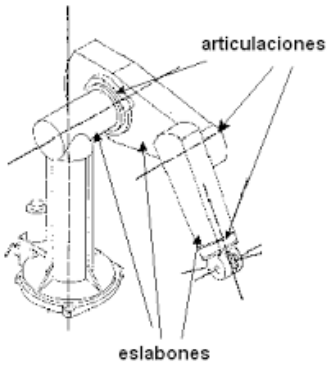


Figura 8: Estructura robótica

El movimiento de cada articulación puede ser de distintos tipos: de desplazamiento, de giro, o una combinación de ambos. Una articulación de revolución permite la rotación relativa entre dos eslabones mientras que una prismática permite el movimiento lineal relativo entre dos eslabones. Los movimientos independientes que puede realizar cada articulación vienen determinados por su grado de libertad. En la práctica, en robótica, solo se emplean las articulaciones de rotación y prismáticas, cuya nomenclatura viene especificada como R para el tipo rotacional y P para el prismático.

2.2 DEFINICIÓN MATEMÁTICA DE UN ROBOT

2.2.1 ALGORITMO DE DENAVIT-HARTENBERG

Un robot manipulador comprende un conjunto de eslabones en serie conectados a través de articulaciones. Cada una de estas articulaciones tiene un grado de libertad, ya sea traslacional (una articulación deslizante o prismática) o rotacional (una articulación giratoria). El movimiento de la articulación supone el movimiento de sus enlaces vecinos.

Jacques Denavit y Richard Hartenberg propusieron, en 1955, una manera de describir la geometría de una cadena de eslabones y uniones en serie, dando lugar a la llamada notación Denavit-Hartenberg (D-H). Este método permite establecer, sistemáticamente, el sistema de coordenadas ligado a cada eslabón de una cadena articulada, permitiendo pasar de uno a otro mediante 4 transformaciones básicas que dependerán de las características geométricas de cada eslabón.

El conjunto de trasformaciones que permite relacionar el sistema de referencia del eslabón *j* con respecto al sistema del eslabón *j-1* es el siguiente:

- Rotación alrededor del eje z_{j-1} un ángulo θ_j.
- Traslación a lo largo de z_{j-1} una distancia d_j.
- Traslación a lo largo de x_j una distancia a_j.
- Rotación alrededor del eje x_j un ángulo α_j.

$$A_j^{j-1} = Rotz(\theta_j)T(0,0,d_j)T(a_j,0,0)Rotx(\alpha_j) \qquad (2.1)$$

Cabe remarcar que las transformaciones se deben realizar en el orden establecido, puesto que el producto matricial no es conmutativo.

Desarrollando la expresión (2.1) se obtiene la siguiente matriz:

$$A_j^{j-1} = \begin{bmatrix} C\theta_j & -S\theta_j & 0 & 0 \\ S\theta_j & C\theta_j & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_j \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_j & -S\alpha_j & 0 \\ 0 & S\alpha_j & C\alpha_j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$A_j^{j-1} = \begin{bmatrix} C\theta_j & -C\alpha_j S\theta_j & S\alpha_j S\theta_j & a_j C\theta_j \\ S\theta_j & C\alpha_j C\theta_j & -S\alpha_j C\theta_j & a_j S\theta_j \\ 0 & S\alpha_j & C\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

siendo θ_j, d_j, a_j, α_j los parámetros D-H del eslabón *j*.

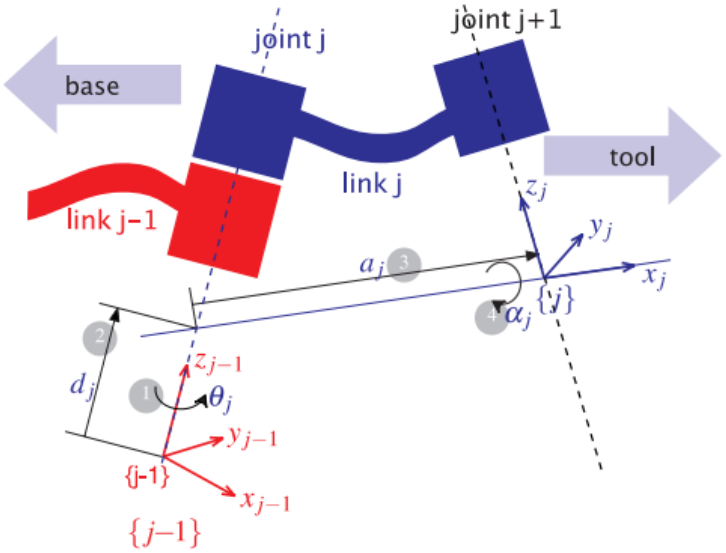


Figura 9: Definición de parámetros de enlace estándar de Denavit-Hartenberg

θ_j	Ángulo de giro del eje x_{j-1} hasta x_j, con respecto al eje z_{j-1}. Parámetro variable si la articulación es giratoria
d_j	Distancia medida a lo largo del eje z_{j-1} entre el punto de intersección del eje z_{j-1}con el eje x_j y el origen del sistema *j-1*. Parámetro variable si la articulación es prismática.
a_j	Para articulaciones giratorias, distancia medida a lo largo del eje x_j entre el punto de intersección del eje z_{j-1}con el eje x_j y el origen del sistema *j.* Para articulaciones prismáticas, distancia más corta entre los ejes z_{j-1} y z_j
α_j	Ángulo de giro del eje z_{j-1} hasta z_j, con respecto al eje x_j.

Tabla 1: Parámetros Denavit-Hartenberg

Mediante la obtención de los parámetros D-H se pueden obtener las matrices de transformación A_j^{j-1} que relacionan eslabones consecutivos, y, mediante el producto de un conjunto de matrices de transformación, se puede obtener la relación entre eslabones no consecutivos del robot. El método para determinar los 4 parámetros D-H viene descrito en [20].

La forma más comúnmente usada en robótica para la definición de un robot es a través de la especificación de sus ejes mediante los parámetros D-H, a partir de la definición de matrices homogéneas entre un eje y el siguiente. Sin embargo, no es la única.

2.2.2 ARCHIVO URDF

2.2.2.1 ¿Qué es URDF (United Robotics Description Format)?

Se trata de un archivo de descripción del robot que tiene un formato de lenguaje XML. Este archivo es ampliamente usado en el ya mencionado ROS; se modela el robot a través de URDF y se realiza la simulación y análisis mediante ROS. También se puede obtener un modelo Simscape Multibody a través de la importación de un archivo URDF, lo cual permite el análisis de simulación o diseño de controlador en SIMULINK. El formato URDF permite definir la descripción cinemática y dinámica del robot, la representación visual de este y su modelo de colisiones.

Los robots se describen mediante estructuras de árbol, a través de un conjunto de elementos de eslabones (*links*) y de un conjunto de elementos de unión (*joints*), que conectan los eslabones juntos y definen características cinemáticas y dinámicas de la articulación, además de limitar los posibles giros y movimientos. Los elementos del robot deben ser estructuras rígidas conectados por articulaciones, las estructuras flexibles no son compatibles.

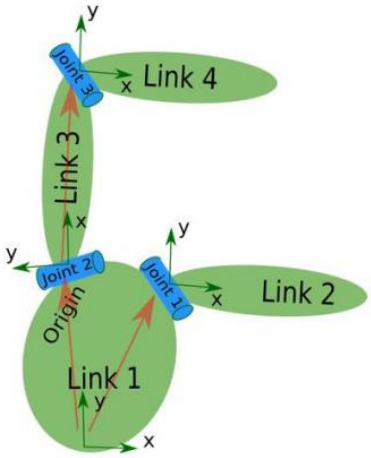


Figura 10: Ejemplo de estructura URDF

La descripción URDF típica de un robot tiene el siguiente formato:

```
<robot name= "ejemplo">
        <link> ... </link>
        <link> ... </link>
        <joint> ... </joint>
</robot>
```

El elemento raíz del formato URDF es el elemento *<robot>,* donde se nombre la estructura del robot diseñada. Seguido a este se definen los elementos *link* y *joint* junto a sus respectivas propiedades.

2.2.2.2 Elemento *Link*

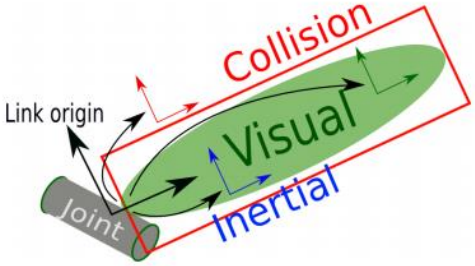


Figura 11: Estructura del elemento link

Cada eslabón del robot está definido por un nombre *<link name>.* Cuando se utiliza una determinada estructura como sistema de coordenadas base el nombre que recibe es *base-link.*

Puesto que describe un cuerpo rígido con una determinada inercia y características visuales, está compuesto por los siguientes bloques con algunos de sus atributos:

- **<visual>:** Encierra la parte visual del eslabón.
    - ***<origin>:*** especifica el origen de coordenadas del elemento visual con respecto al origen del *link*. El desplazamiento viene representado por x,y,z y los ángulos de giro por r,p,y.
    - ***<geometry>:*** especifica la forma visual de la estructura. Puede ser una forma determinada como un cubo o una figura tridimensional definida al cargar un archivo.
    - ***<material>:*** define un nombre del material del elemento visual.
    - ***<color rgba>:*** define el valor del color del material mediante cuatro números que representan el rojo, verde, azul y opaco, en un rango entre 0 y 1

- **<inertial>:** Define las propiedades inerciales del eslabón.
    - ***<origin>:*** también se define el origen del elemento inercial con respecto al origen del *link.*
    - ***<mass>:*** determina la masa del eslabón.
    - ***<inertia>:*** es la matriz de rotación inercial 3x3.

- **<collision>**

  Especifica las propiedades de colisión del eslabón. Generalmente es más grande que el elemento de visión de tal manera que detecta que el *link* ha colisionado antes de que suceda realmente.
  En el elemento de colisión también se pude definir, al igual que hemos visto en los casos anteriores, propiedades como *<origin>, <geometry>, <material>* o *<color>.*

Ejemplo:

```
<link name="link_1">

  <inertial>
   <mass value="3.067"/>
   <origin rpy="0 0 0" xyz="9.77E-05 -0.00012 0.23841"/>
   <inertia ixx="0.0142175" ixy="-1.28579E-05" ixz="-2.31364E-05"   iyy="0.0144041"
        iyz="1.93404E-05" izz="0.0104533"/>
  </inertial>

  <visual>
   <origin rpy="0 0 0" xyz="0 0 0"/>
   <geometry>
    <mesh filename="package://Visual/link_1.stl"/>
   </geometry>
   <material name="">
    <color rgba="0.7372549 0.3490196 0.1607843 1"/>
   </material>
  </visual>

  <collision>
   <origin rpy="0 0 0" xyz="0 0 0"/>
   <geometry>
    <mesh
filename="package://abb_irb120_support/meshes/irb120_3_58/collision/link_1.stl"/>
   </geometry>
   <material name="">
    <color rgba="1 1 0 1"/>
   </material>
  </collision>
 </link>
```

2.2.2.3 Elemento *Joint*

Se presenta como un elemento de unión entre eslabones que define las propiedades dinámicas y cinemáticas.

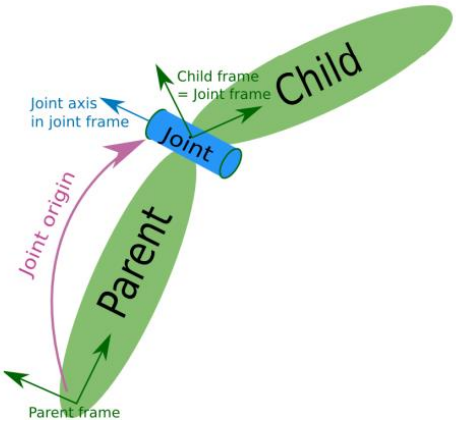


Figura 12: Estructura elemento Joint

Cada elemento *joint* viene definido por un nombre y el tipo de articulación que va a representar (prismática, continua, de revolución, fija…). Dentro de cada *joint* pueden definirse varios elementos, entre ellos:

- ***<parent link>:*** especifica quién es el eslabón padre, es decir, la estructura principal conectada por esta unión.

- ***<child link>:*** especifica quién es el eslabón hijo, es decir, la estructura secundaria conectada por esta unión.

- ***<origin>:*** define el origen de coordenadas del elemento joint con respecto al origen de coordenadas del *link padre*. Como se muestra en la *figura 12*, la articulación se encuentra en el origen del *link hijo*.

- ***<axis>:*** establece la dirección del movimiento de la articulación. Por defecto, el valor predeterminado es el eje x *(1 0 0).*

- ***<dynamics>:*** especifica las propiedades físicas de la articulación: *damping* indica el valor de amortiguación física de la articulación y *friction* el valor de fricción estática física de la articulación.

- ***<limit>:*** define los límites de movimiento de la articulación. Se puede definir únicamente para articulaciones prismáticas y de revolución.

Ejemplo:

```
<joint name="joint_1" type="revolute">
   <origin rpy="0 0 0" xyz="0 0 0"/>
   <parent link="base_link"/>
   <child link="link_1"/>
   <limit effort="0" lower="-2.87979" upper="2.87979" velocity="4.36332"/>
   <axis xyz="0 0 1"/>
   <dynamics damping="0.0" friction="0.0"/>
 </joint>
```

Cuando se plantea la creación de un modelo URDF, con gran número de *links* y *joints*, el código puede volverse muy extenso. Debido a ello, se utiliza XACRO. XACRO es un lenguaje de macros XML que permite crear archivos URDF más cortos y legibles mediante el uso de macros.

3. DESARROLLO DEL PROYECTO

3.1 CREACIÓN LIBRERÍA DE ICONOS

Se ha desarrollado una librería de iconos para el entorno SIMULINK que permite el diseño de una estación robótica, de tal manera que el alumno pueda crear su propia estación, de forma rápida y sencilla, mediante la incorporación de bloques de iconos que representen todos los elementos requeridos para trabajar con esta: robots manipuladores, objetos de trabajo, herramientas … Cada icono cuenta con una máscara donde se definen sus parámetros.

La creación de las estaciones mediante el uso de la librería Simscape Multibody tiene como fin su importación en MATLAB, de tal manera que se pueda interactuar con ella en un entorno de simulación.

3.1.1 ICONOS DE ROBOTS MANIPULADORES

El apartado 2.2 muestra dos procedimientos por los cuales se pueden obtener la definición de un robot: a través del algoritmo de Denavit-Hartenberg y, a través de un archivo de descripción URDF. La creación de los iconos de los robots de los que queremos disponer en la librería se ha basado en estos dos procedimientos para su realización: a través de un formato URDF específico y, mediante la universalización de los ejes D-H, compatible con el fichero URDF de formulación libre.

3.1.1.1 Importación archivo URDF a SIMULINK.

La librería Simscape Multibody de SIMULINK permite la importación y traducción de ficheros URDF en un fichero SIMULINK. Como se ha visto, estos modelos muestran los eslabones y elementos de unión del robot, así como las propiedades cinemáticas y dinámicas de cada articulación y sus límites de movimientos. Por lo tanto, a través de la obtención e importación del archivo de descripción URDF específico de un determinado robot, se puede obtener el modelo en SIMULINK que lo define y, con este, crear el subsistema que específica su icono en la librería.

Para su importación, se elige como carpeta de trabajo aquella donde se ha guardado el fichero URDF y, en la ventana de Comandos de MATLAB, se introduce el comando *Smimport:*

>> smimport('Nombre.urdf')

Una vez ejecutado, se abre una ventana SIMULINK y empieza la importación, que concluye con la creación de un sistema donde vienen reflejados todos los aspectos de la estructura del robot.

En el fichero SIMULINK creado aparecen los siguientes bloques de la librería Simscape Multibody:

- **World Frame**

  Este bloque representa el marco de referencia global en un modelo. Establece un sistema de referencia inercial y en reposo absoluto. Sus ejes de coordenadas son ortogonales y solo puede existir uno en cada sistema, el resto de los puntos de referencia se definen con respecto a él.

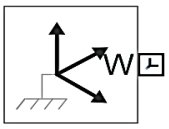


Figura 13: Bloque World Frame de la librería Simscape Multibody

- **Mechanism Configuration**

  Este bloque proporciona parámetros mecánicos y de simulación al mecanismo al que se conecta. Principalmente configura la fuerza de la gravedad, pudiendo establecerla como cero, constante o variable en el tiempo.

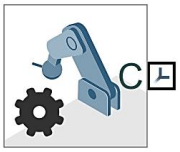


Figura 14: Bloque Mechanism Configuration de la librería Simscape Multibody

- **Solver Configuration**

  Este bloque es necesario para todos los modelos de Simscape. Especifica los parámetros de cálculo de la simulación, como el tipo de solucionador a utilizar, las opciones de inicialización o el tiempo de muestra para la simulación.

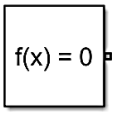


Figura 15: Bloque Solver Configuration de la librería Simscape Multibody

Estos tres primeros bloques aparecen siempre como punto de partida cuando se desea iniciar un nuevo modelo, dando lugar a la base de este.

- **Reference Frame**

Permite definir sólidos fijos en el espacio estableciendo un sistema de referencia en el punto en que se conecte.

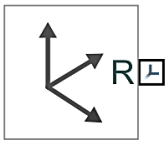


Figura 16: Bloque Reference Frame de la librería Simscape Multibody

- **Rigid Transform**

Este bloque implanta las relaciones de posición, traslación y rotación entre los elementos que se conectan a sus extremos (B/F). Gracias a él se disponen los componentes físicos del sistema de la forma deseada.

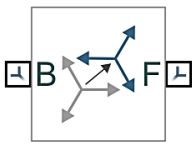


Figura 17: Bloque Rigid Transform de la librería Simscape Multibody

- **Joint**

Este bloque de unión esblece las restricciones que determinan cómo pueden moverse entre sí los cuerpos conectados a sus extremos (B/F). Pueden ser de distintos tipos: de revolución, planar, de soldadura… También pueden ser usados como sensores o actuadores.

Figura 18: Bloque Revolute Joint

Figura 19: Bloque Weld Joint

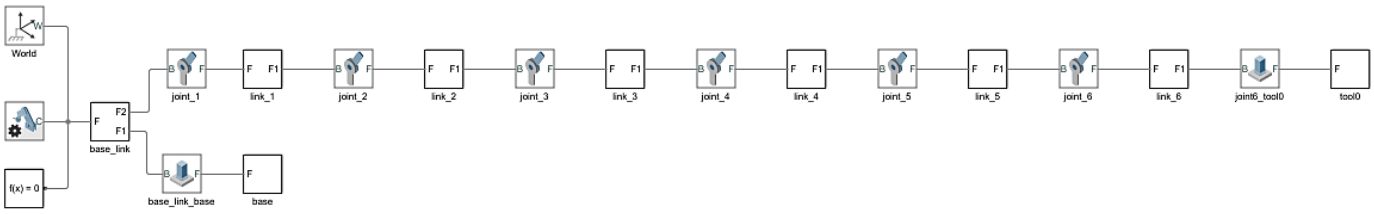


Figura 20: Modelo del brazo robótico irb120 de ABB

Como ejemplo de sistema, *la figura 20* muestra el archivo que se obtendría en SIMULINK tras la importación del fichero URDF del robot *irb120* de ABB. En este modelo vienen definidos cada uno de los eslabones (*links*) que componen el robot, así como cada uno de los *joints* que unen estos eslabones.

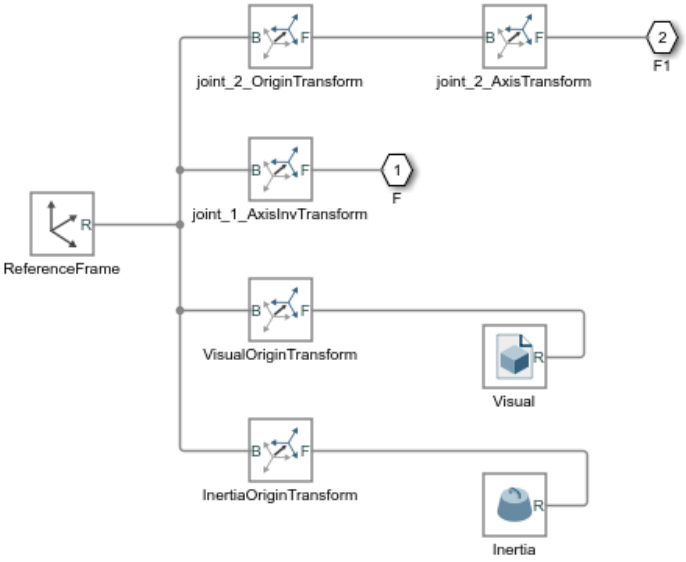


Figura 21: Definición de un cuerpo (link) con Simscape Multibody

Como vemos en la *figura 21*, para la definición de estos cuerpos (*links*), además del uso de bloques *Rigid Transform* y del establecimiento de su sistema de referencia, también se debe definir su masa, centro de masa, momentos y productos de inercia (Bloque *Inertia*), así como la geometría del cuerpo (Bloque *File Solid*).

Figura 22: Bloques Inertia y File Solid de la librería Simscape Multibody

Con estos modelos importados, se generan los subsistemas que dan lugar a los iconos de los robots manipuladores.

3.1.1.2 Universalización con Denavit-Hartenberg

Con el fin de conseguir un método más universal de obtención de modelos de robots, que no requiera el URDF específico de cada uno de ellos, este proyecto ha buscado la creación de iconos partiendo de un modelo base en SIMULINK que permite, a partir de unas simples modificaciones de sus parámetros, adaptar dicho modelo a cada robot en particular.

El patrón seguido a la hora de construir los iconos de cada robot está basado en dos aspectos:

-Sistema de SIMULINK obtenido a partir de un archivo URDF estándar.
- Parámetros D-H que definen la matriz homogénea entre un eje del robot y el siguiente.

Como se ha comentado, se puede obtener un modelo en SIMULINK partiendo de un archivo URDF. Para conseguir nuestro objetivo, se ha partido de un archivo de definición de formato universal convertido en un modelo de SIMULINK. Este cuenta con un número determinado de eslabones *(links)* y uniones (*joints)* que se irá modificando en función de la estructura concreta del robot que se quiera representar.

Este modelo se convertirá en el subsistema que componga el icono de cada robot. Para poder especificar cada uno de ellos, se han definido una serie de variables internas dentro del modelo que podrán ser modificadas a través de una máscara creada. Se irá distinguiendo entre variables que son inicializadas dentro de la máscara y variables que pueden ser modificadas por el alumno a través de MATLAB. A continuación, se mostrará la colocación y definición de estas variables, así como los pasos seguidos para obtener este icono universal.

En primer lugar, al principio del modelo importado se le ha incorporado el bloque *Rigid Transform*, que permite la colocación de la base del robot en una posición concreta con respecto al sistema de coordenadas global de la estación.

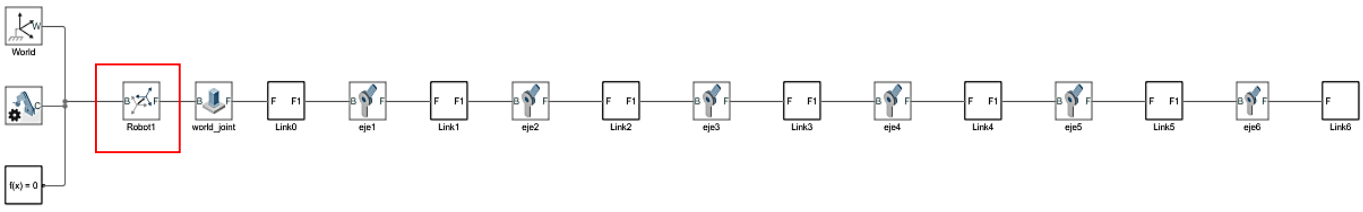


Figura 23: Modelo estándar de un robot Simscape Multibody

Esta posición se define dentro del bloque como una variable local, *base*, para que pueda ser modificada por el alumno a través de la máscara:

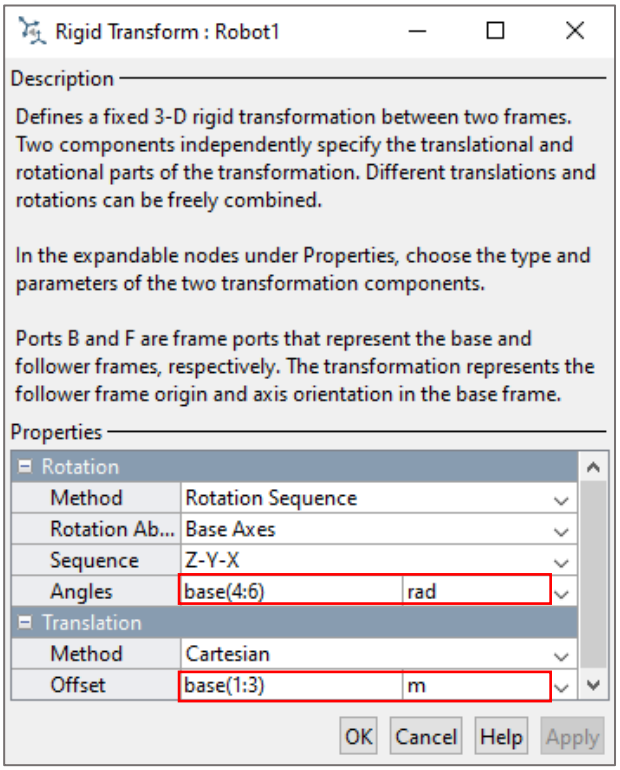


Figura 24: Bloque Rigid Transform

Para el funcionamiento de una estructura robótica móvil se precisa de articulaciones de revolución. Los bloques de la librería Simscape Multibody que las definen ofrecen un gran abanico de posibilidades, siendo posible definir los puntos por los que se quiere que pase durante la simulación, sus límites de giro, su mecánica interna y, además, pueden actuar como sensores y actuadores.

Puesto que los límites máximos y mínimos de giro de una articulación dependen de la estructura de esta, estos se han definido como variables internas para ser inicializadas dentro de la máscara en función de las especificaciones del robot que se quiera configurar. Además, también se ha definido como variable local la posición inicial de cada articulación, *q0*, que puede ser introducida por el alumno. Cabe mencionar que, al simular el robot, esta variable permitirá que los sucesivos movimientos partan de una posición inicial, la cual se irá modificando con respecto a la finalización del movimiento anterior.

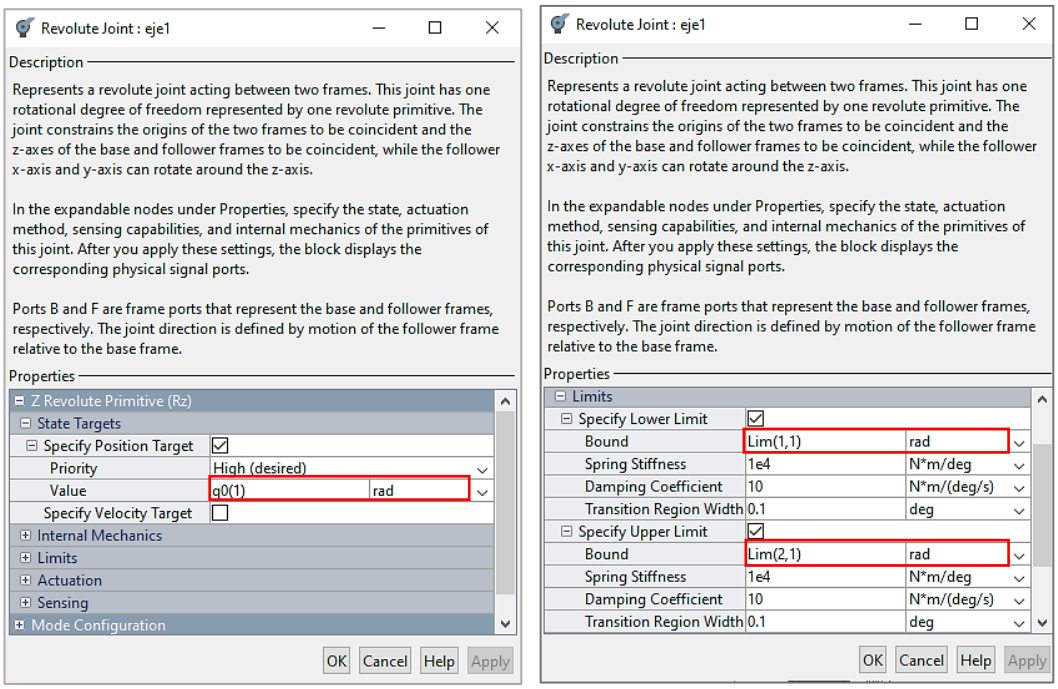


Figura 25: Bloque Revolute Joint del primer eje de un robot

El modelo también presenta cada uno de los eslabones que conforman al robot. Cada estructura de un cuerpo (*link*) presenta un bloque *File Solid* donde se establecen sus propiedades geométricas; se define la figura tridimensional de este a través de un fichero *.stl, así como sus valores de inercia, color y opacidad, pudiendo ver el aspecto que mostrará en el espacio de simulación. El color y opacidad también han sido definidas como variables para poder ser inicializadas con un valor determinado dentro de la máscara que engloba el icono.

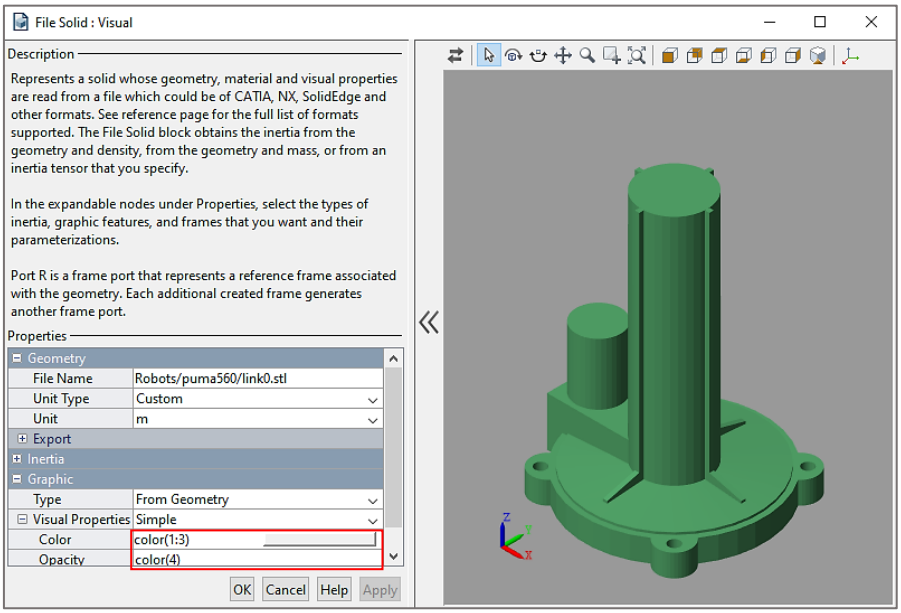


Figura 26: Bloque File Solid del eslabón base del robot Puma560

También se debe definir, en el bloque *Inertia*, la masa, centro de masa, momentos y productos de inercia de cada eslabón. Todos ellos establecidos como variables locales inicializadas dentro de la máscara.

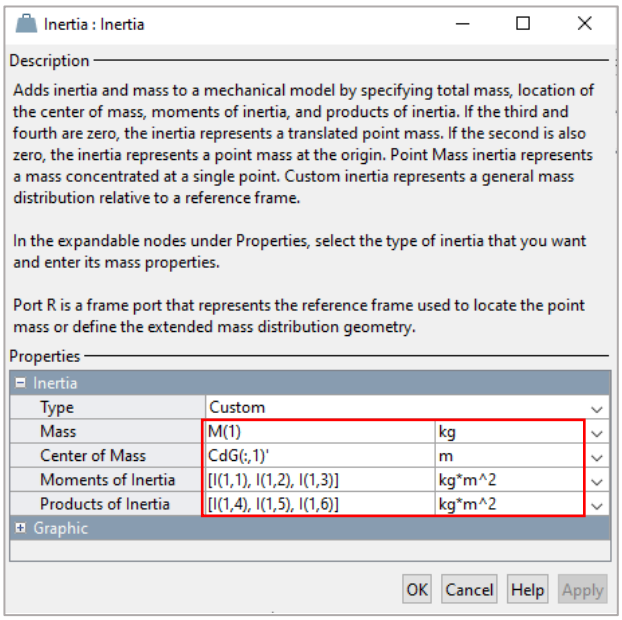


Figura 27: Bloque Inertia del primer eslabón del robot Puma560

Con este modelo, para obtener una universalización de los ejes del robot, se han definido las transformaciones que permiten relacionar el sistema de referencia de un eslabón con respecto a su anterior. El algoritmo de Denavit-Hartenberg establece que estas transformaciones se basan, con respecto a los ejes del eslabón anterior, en una rotación de un ángulo θ alrededor del eje z, una traslación d en el eje z, otra traslación a a lo largo del eje x, y, una rotación α alrededor del eje x.

A cada *link* del modelo, unido por una articulación giratoria, se le ha añadido los bloques tipo *Rigid Tranform* que permiten establecer estas relaciones con respecto a su precedente:

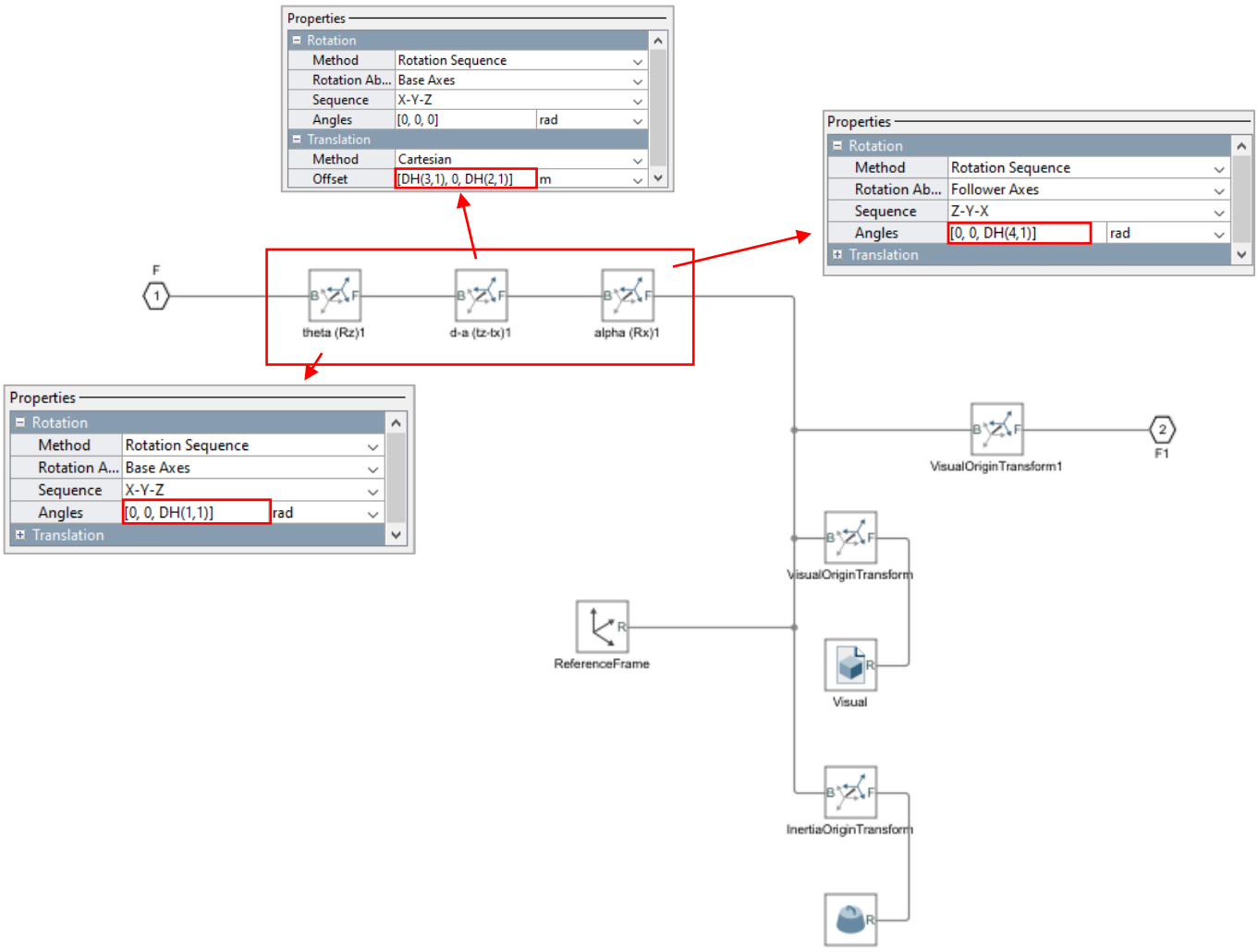


Figura 28: Estructura de un eslabón (link) del modelo en SIMULINK

Como se puede observar en la *figura 28*, la primera relación se basa en la rotación alrededor del eje z, seguido por las traslaciones en el eje x y z, para lo cual se ha usado un único bloque puesto que estas no se interponen entre sí. Finalmente, el último bloque define la rotación alrededor del eje x.

$\theta_j, d_j, a_j, \alpha_j$ son los parámetros D-H del eslabón j. Estos serán variables locales inicializadas dentro de la máscara.

El *link 0*, que representa la base de la estructura del robot, solo presenta el bloque que define la rotación alrededor del eje z con respecto al sistema de coordenadas global.

En resumen, conociendo los parámetros D-H de los eslabones de un determinado robot, sus límites de movimiento, valores de inercia y geometría, se puede modelar el modelo universal para obtener el icono de un determinado robot manipulador.

3.1.1.3 Modelado del icono

Como ya se ha mencionado, a partir del modelo universal determinado en el apartado anterior se ha generado el subsistema que da lugar al icono del robot. Se ha creado una máscara que engloba este subsistema para especificar las variables que definen cada robot manipulador y así obtener la representación de un conjunto de ellos con los que disponer en la librería.

A la hora de modelar el icono de cada robot se van a seguir dos pasos:

1) En función de la estructura robótica, se debe ajustar el número de *joints* y *links* del modelo, añadiendo a cada uno de los eslabones los ficheros *.stl que van a permitir su visualización.
2) Se inicializan dentro de la máscara como variables:
   -Color de los ficheros de visualización.
   -Parámetros D-H de cada eslabón.
   -Límites de giro de articulaciones de revolución.
   -Parámetros dinámicos: masa, centro de gravedad, momentos y productos de inercia.

La ya mencionada librería ARTE proporciona todos estos parámetros de un gran número de robots, incluidos los ficheros de visualización de los *links* de cada uno de ellos. Debido a ello, se ha recurrido a esta para poder construir los iconos de cada robot manipulador.

Como ejemplo de parametrización de un manipulador determinado, se va a mostrar la definición del robot Puma560.

1) Este presenta 6 grados de libertad, coincidiendo el número de articulaciones y uniones con el del modelo base del que hemos partido, por lo que no hace falta realizar ninguna incorporación. Se han introducido en el bloque *File Solid* de cada *link* los ficheros *.stl obtenidos a través de la librería ARTE.

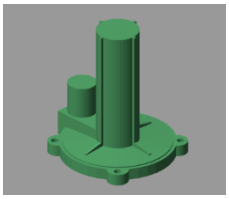
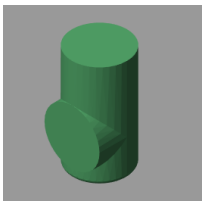
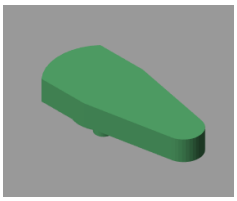
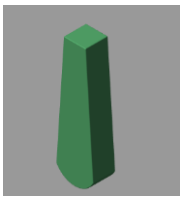
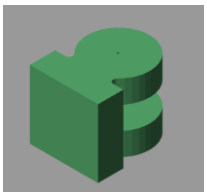
				
Link 0	*Link 1*	*Link 2*	*Link 3*	*Link 4*

*Tabla 2: Archivos *.stl de algunos links del robot Puma560*

2) En la máscara del icono se especifica, en el apartado de inicialización, los siguientes datos del robot:

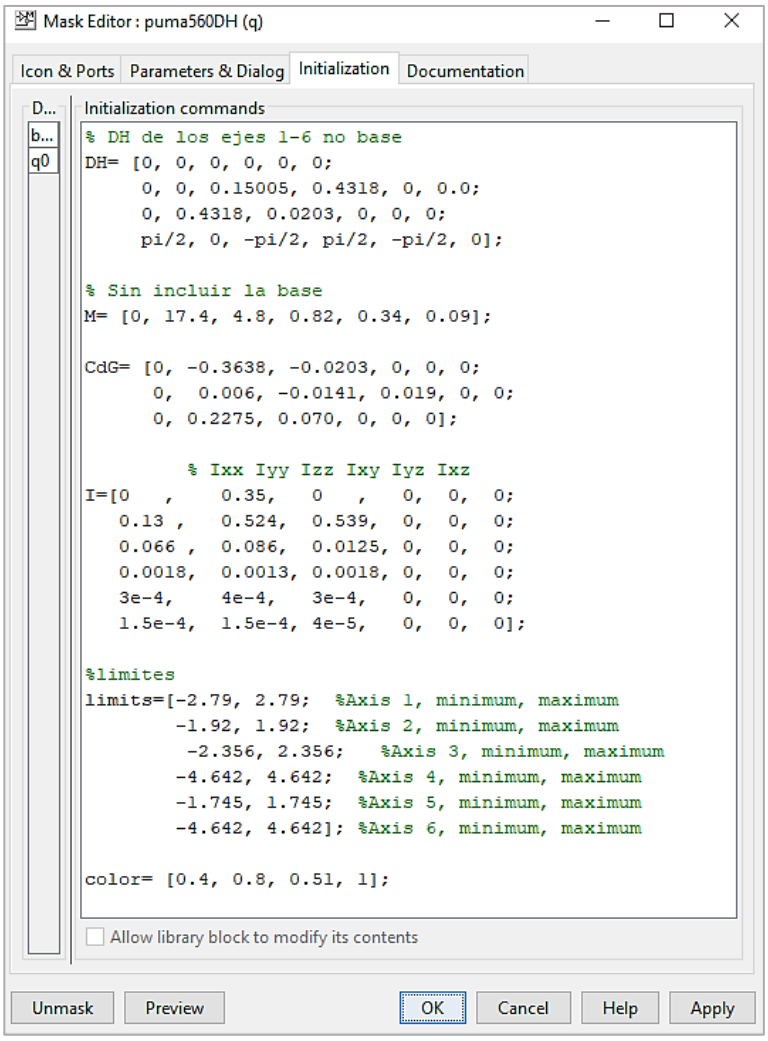


Figura 29: Cuadro de inicialización de la máscara del icono Puma560

Cabe recordar que, cuando se vaya a usar un icono, el alumno debe introducir los parámetros *base* y *q0*; variables locales del modelo que no han sido inicializadas dentro de la máscara.

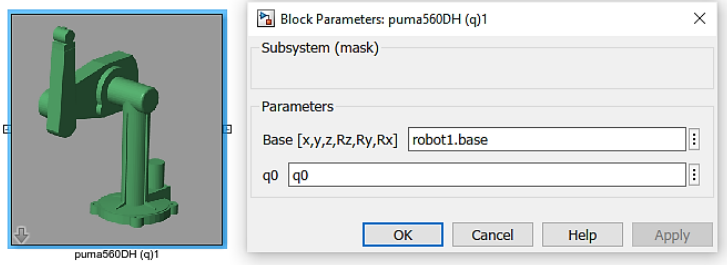


Figura 30: Parámetros base y q0 del icono del robot Puma560

Las mismas modificaciones han sido realizadas con una serie de robots manipuladores y, además, se ha modificado el aspecto de cada icono para que muestre la imagen del robot que representa.

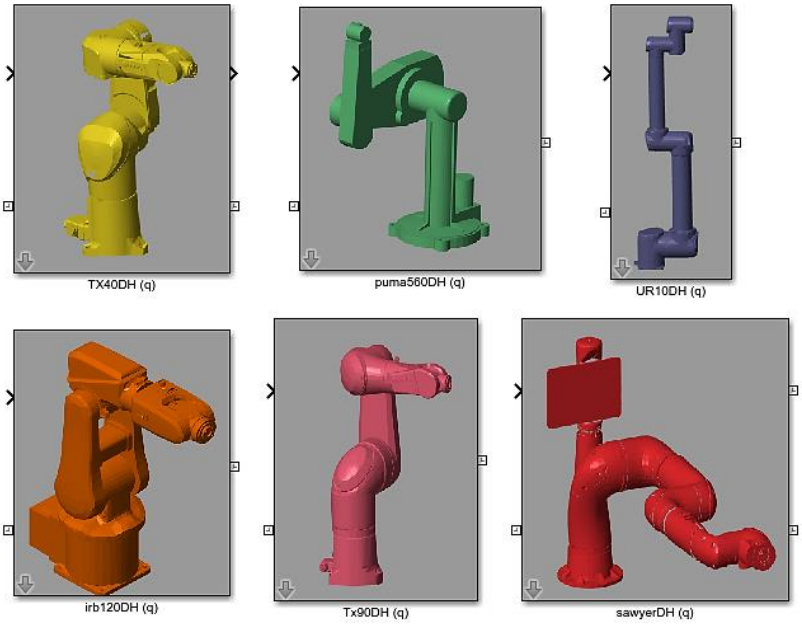


Figura 31: Iconos de robots manipuladores disponibles en la librería

3.1.1.3 Comportamiento

Finalmente, para concluir con el diseño de estos iconos, se ha realizado un análisis del comportamiento para verificar que se asemeja al del sistema real. Se puede obtener la visualización tridimensional de cada uno de estos modelos mediante Mechanics Explorer, gracias a los ficheros *.stl.

En primer lugar, se probó qué sucede cuando no hay ninguna actuación sobre el sistema, cuando únicamente está sometido a la acción de la gravedad, que actúa sobre el eje z. En este caso, los robots no tienen ningún tipo de control y caen por su propio peso, produciendo su propia oscilación.

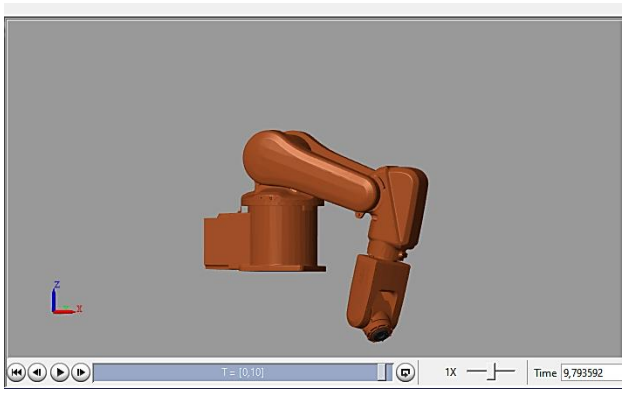


Figura 32: Ejemplo comportamiento del robot irb120 sin entradas al modelo

Para limitar esa caída, cada modelo debe ser modificado de tal manera que se le añada una actuación. Las entradas a cada uno de los ejes del robot pueden ser de dos tipos: ángulo o par motor. La librería dispone de iconos que están sometidos a una actuación cinemática, sin embargo, también se va a estudiar su comportamiento cuando la entrada es un par motor en el apartado de dinámica *(apartado 3.4.1.2)*.

Se han modificado las articulaciones (bloques *joint)* de cada modelo de robot para realizar una actuación sobre el movimiento, permitiendo obtener una determinada posición inicial mediante la introducción del ángulo. El torque será calculado de manera automática.

Esta actuación requiere de un ángulo, velocidad y aceleración angular. Se ha generado un subsistema que permita obtener, a partir del ángulo deseado, la velocidad y aceleración mediante una sucesión de derivadas.

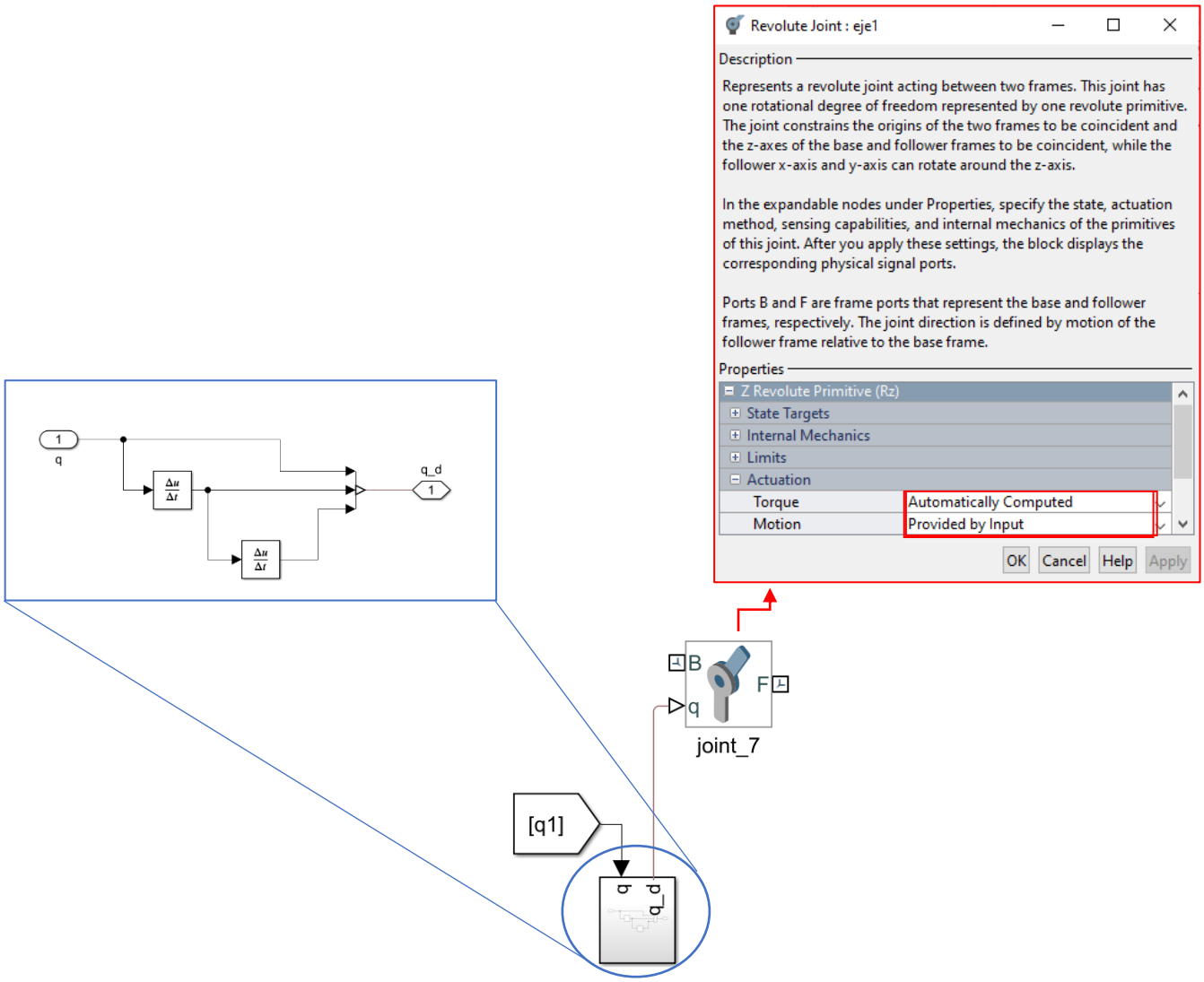


Figura 33: Bloque Joint con ángulo de entrada

A modo de ejemplo, se muestra el comportamiento resultante del robot *irb120* de ABB al introducirle distintos ángulos, uno por cada *joint* que compone la estructura de este robot industrial:

- ángulos [0,0,0,0,0,0] rad (*figura 34*).

- [0,1,0,1,0,0] rad (*figura 35*).

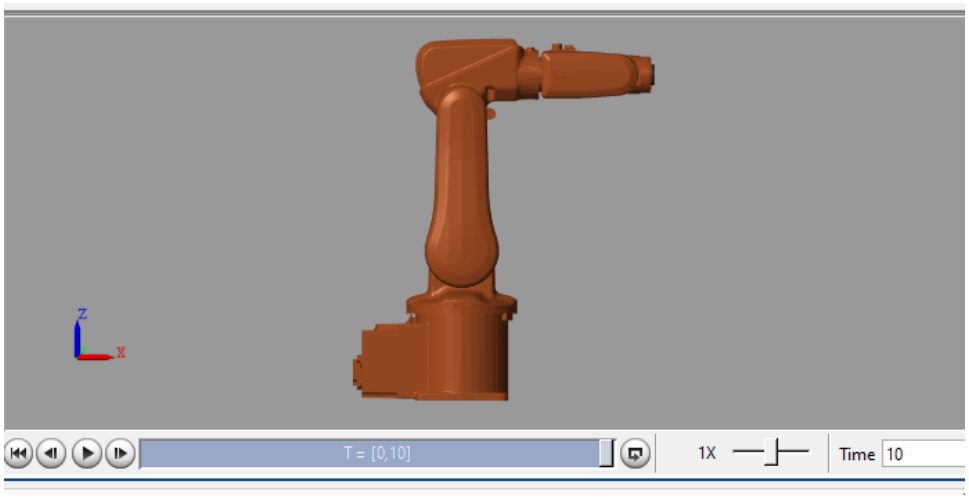


Figura 34: Ejemplo robot irb120 con ángulos de entrada (0,0,0,0,0,0) rad

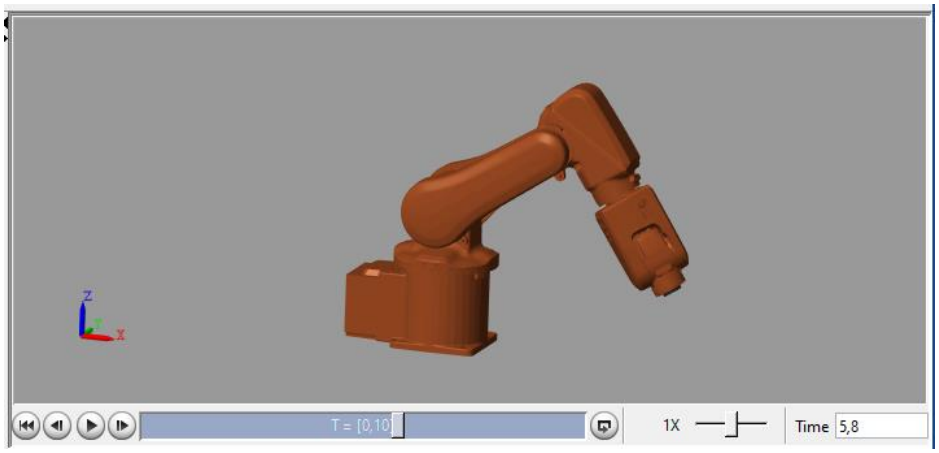


Figura 35: Ejemplo robot irb120 con ángulos de entrada (0,1,0,1,0,0) rad

Todos los iconos de robots manipuladores, creados y almacenados en la librería presentan una actuación sobre su modelo con el fin de obtener un comportamiento lo más semejante a la realidad posible a la hora de obtener su simulación. De esta manera, se concluye con el diseño de los iconos de robots manipuladores.

3.1.2 ICONOS DE TRABAJO

3.1.2.1 Iconos de objetos de trabajo (*Wobj*)

La creación de iconos que representen el objeto de trabajo (*Wobj)* que se quiera incorporar a la estación también se ha llevado a cabo mediante la creación de un subsistema, el cual cuenta con una máscara a través de la que se define una variable de tipo estructura. De esta forma, solo con modificar la variable de la máscara se modifica el icono.

En el interior de dicho subsistema se ha creado un modelo que permite la definición de un cuerpo en el entorno SIMULINK mediante herramientas de la librería Simscape Multibody:

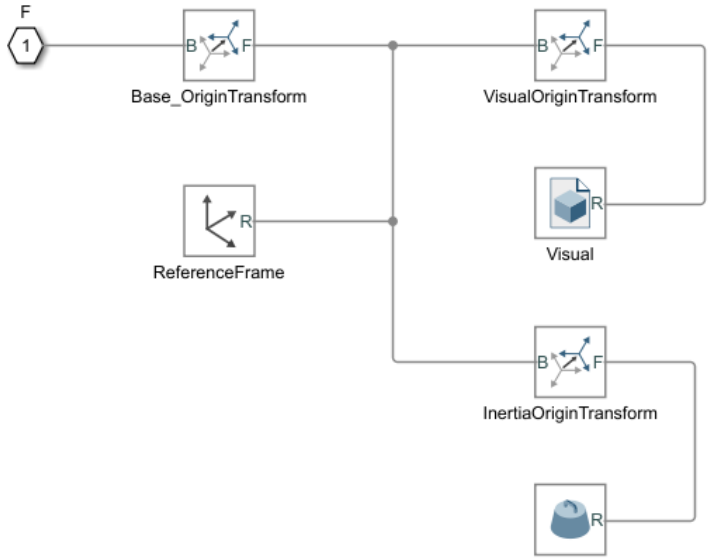


Figura 36: Modelo del icono de un objeto de trabajo

Como vemos en la *figura 36*, cada objeto de trabajo está definido por su sistema de referencia, sus características geométricas; se define su figura tridimensional a través de un fichero *.stl, así como sus valores de inercia y color y, por sus características inerciales; masa, centro de masa, momentos y productos de inercia. Se establecen las relaciones de posición, traslación y rotación entre los elementos y su sistema de referencia, así como con el marco de referencia global del modelo (bloques *Rigid Transform*).

Los bloques que componen el modelo han sido explicados en el apartado de los iconos de robots manipuladores.

Con el propósito de disponer de una serie de objetos de trabajo con los que trabajar, se ha personalizado cada subsistema para referirse a un elemento concreto: esfera, cilindro, tablero… Para ello, se han obtenido los archivos *.stl que los representan, definiéndose cada uno de estos dentro del bloque *File Solid* del icono de trabajo correspondiente. La mayoría de los elementos seleccionados presentan su equivalente en RobotStudio.

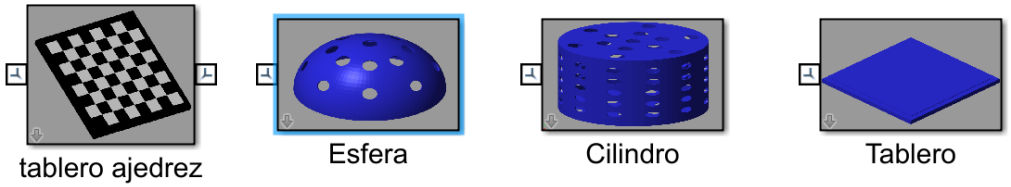


Figura 37: Ejemplos de iconos de trabajo presentes en la librería

Cada icono presenta una máscara en la que se ha definido la variable estructura de nombre *Wobj*. El nombre de esta variable debe ser modificada en función del objeto de trabajo específico a representar, ejemplo *figura 39.*

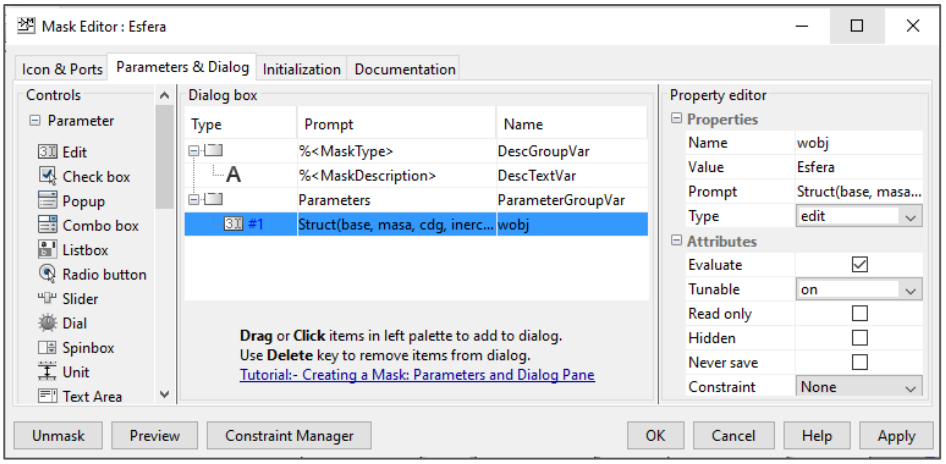


Figura 38: Máscara de un icono de objeto de trabajo

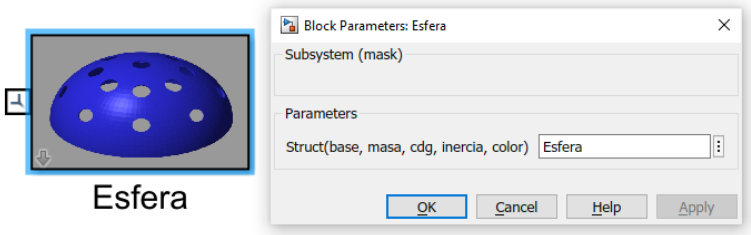


Figura 39: Parámetro de la máscara del icono Esfera

Wobj es una variable estructura que permite al usuario modificar los siguientes campos: base, masa, centro de gravedad, inercia y color. Se han definido estos campos dentro de cada bloque correspondiente:

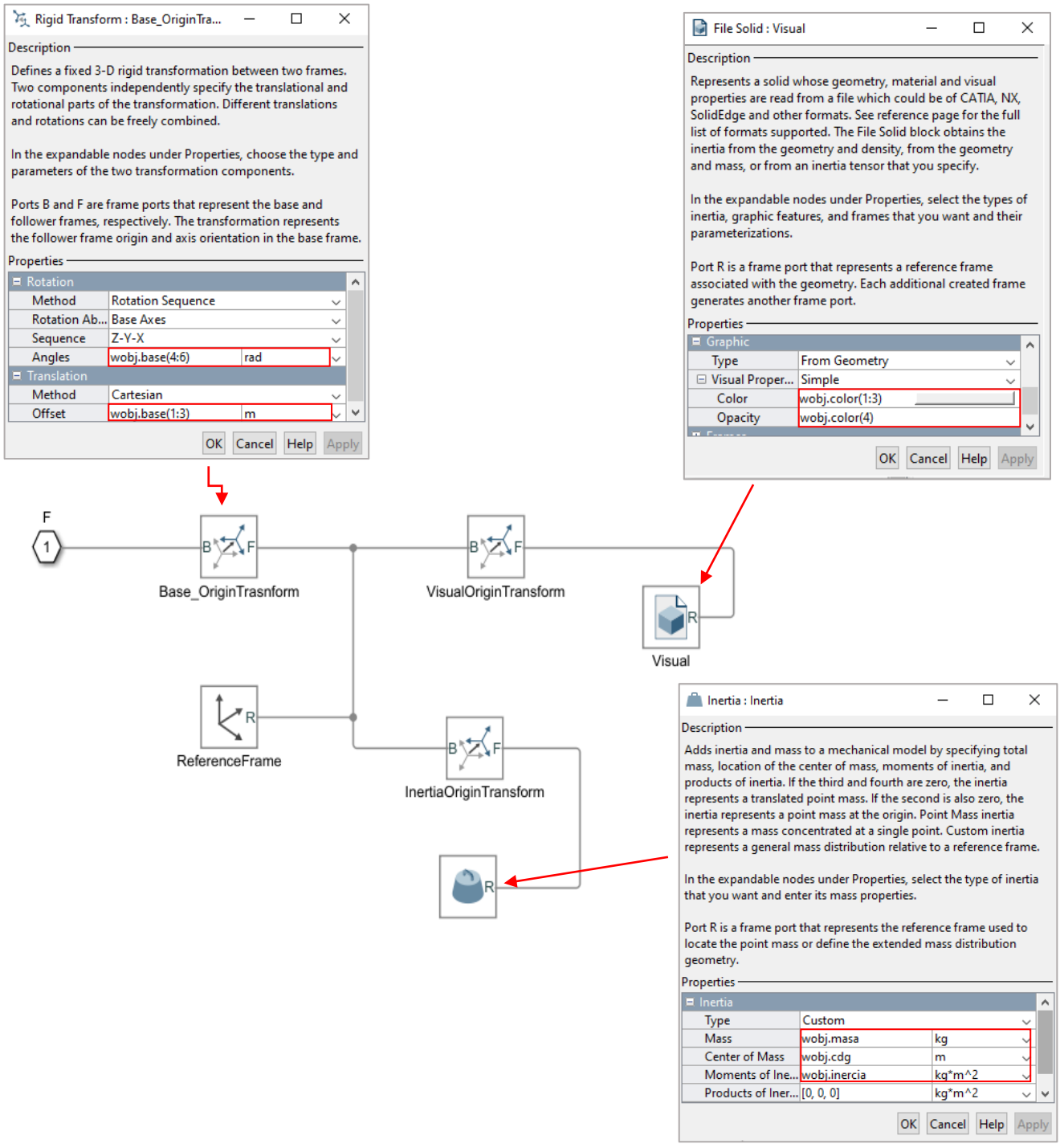


Figura 40: Parámetros de los bloques de iconos de objeto de trabajo

De esta manera, se puede insertar en la estación tantos objetos de trabajo como se quieran, permitiendo al alumno especificar los datos de cada uno de ellos mediante su variable estructura asociada, a través de la ventana de comandos de Matlab.

3.1.2.2 Iconos de cargas/herramientas (*Tool/Load*)

La librería también dispone de un conjunto de iconos de herramientas y cargas (*tool/load*). Estas pueden enlazarse entre sí y con los robots que componen la estación, permitiendo realizar distintos juegos de manipulación de cuerpos.

De manera análoga a los iconos de los objetos de trabajo, estos se basan en un subsistema al que se le han establecido parámetros, englobados por una variable estructura, modificable a través de una máscara. Una herramienta o carga también vendrá definida por su propio sistema de referencia, su geometría, definida a través de un archivo *.stl, de color e inercia específica, y sus propiedades inerciales: masa, centro de masa, momentos y productos de inercia.

También se establece la posición de la base de la herramienta o carga, referida respecto al marco de referencia que mejor convenga. Por ejemplo, la posición de una pinza se desea que este basada respecto al sistema de referencia del último eslabón del robot al que se desea acoplar. Sin embargo, a mayores, también se específica el punto de trabajo exacto de estas (TCP), lo cual va a permitir definir el extremo del robot con el que se trabaja en la simulación.

Para ello, con respecto al subsistema que compone los iconos de objetos de trabajo visto, se han realizado unas pequeñas modificaciones: se ha introducido el bloque que permite definir la relación de posición, traslación y rotación entre el punto extremo y el sistema de referencia de la propia carga/herramienta, así como la unión de tipo soldadura, que va a permitir acoplar una pieza a la siguiente y generar movimientos relativos entre ejes.

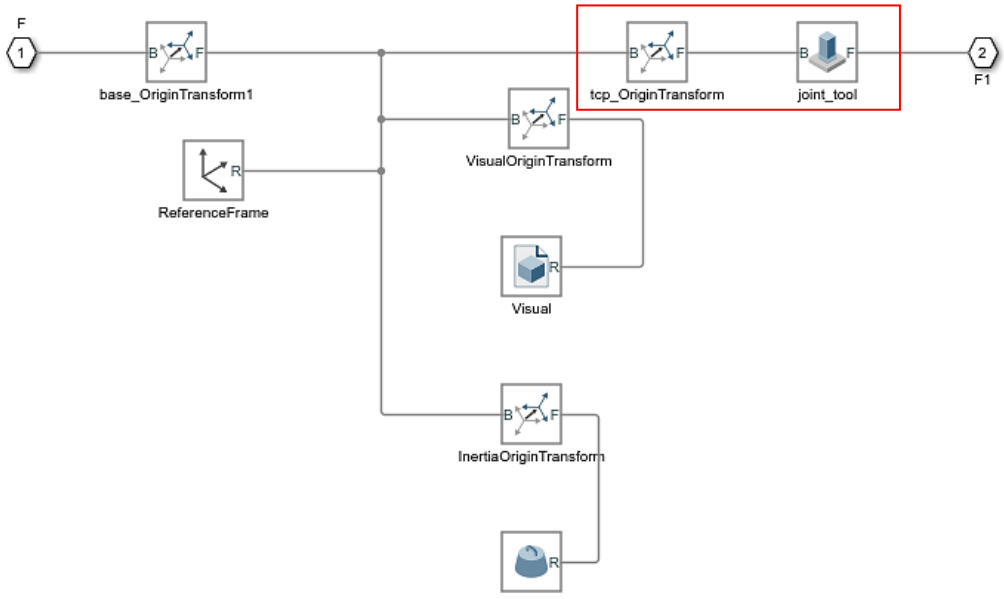


Figura 41: Modelo de los iconos de carga/herramienta

Se han realizado distintos iconos que representen cargas o herramientas específicas: lápiz, pinza, piezas de ajedrez… Los archivos *.stl empleados también permiten su compatibilidad con RobotStudio.
Todos los iconos presentan una máscara en la que se ha definido la variable estructura de nombre *tool*. El nombre de esta variable debe ser modificado en función de la carga o herramienta con la que se trabaje, ejemplo *figura 42*.

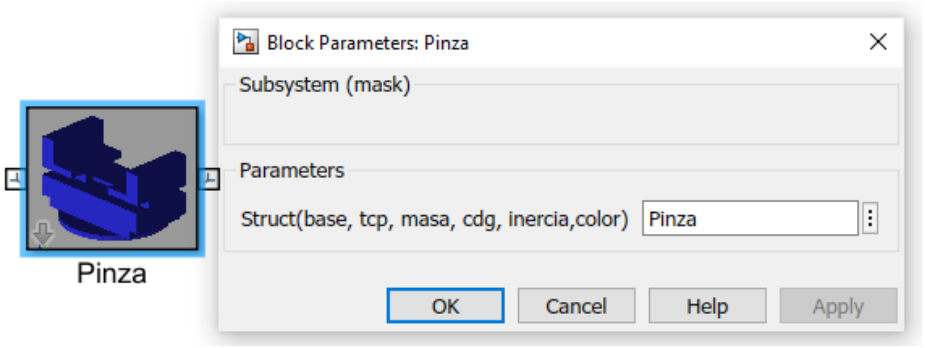


Figura 42: Parámetro tool de la máscara del icono Pinza

Esta variable presenta los siguientes campos modificables: base, TCP, masa, centro de gravedad, inercia y color. Estos han sido definidos dentro de cada bloque correspondiente, al igual que se mostraba en el apartado anterior, a excepción del TCP que en este caso se define en el bloque añadido *tcp_OriginTransform.*

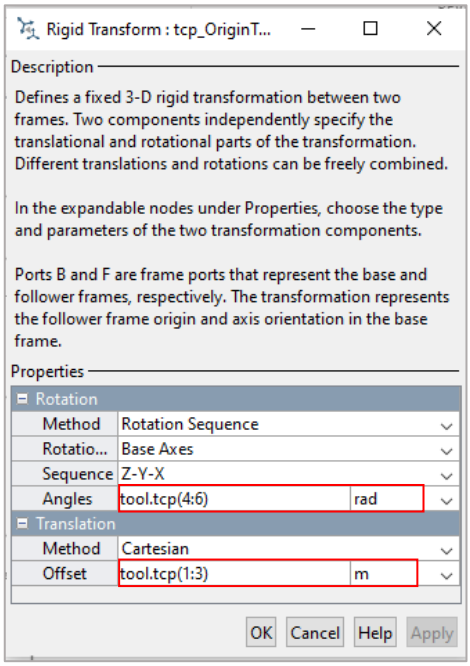


Figura 43: Bloque tcp_OriginTransform del icono de herramienta/carga

El alumno define los datos de la herramienta o carga a través de la ventana de comandos de MATLAB. Además, al igual que sucede con los objetos de trabajo, se ha personalizado a través de la máscara el aspecto de cada icono para que muestre el elemento que representa, una vez ha sido definido.

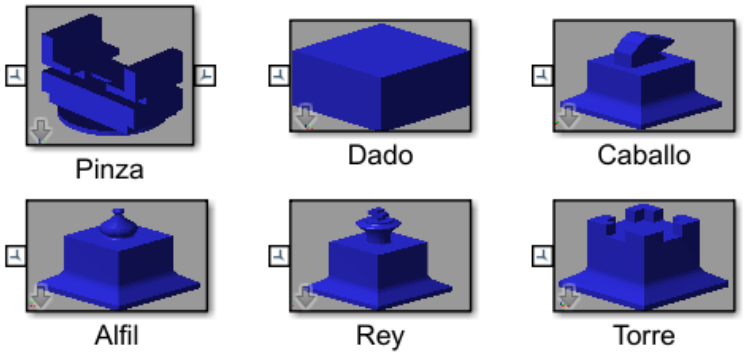


Figura 44: Iconos de trabajo disponibles en la librería

3.1.2.3 Parametrización

Los iconos de objetos de trabajo, así como los que representan cargas o herramientas, presentan una máscara que permite al alumno establecer los parámetros específicos de cada uno de ellos dentro de una variable estructura. Estos parámetros, establecidos dentro del modelo, son:

- base
- color
- masa
- centro de gravedad
- inercia
- tcp, en el caso de cargas o herramientas

Una vez el alumno haya diseñado su propia estación en SIMULINK, se debe definir estos parámetros de cada uno de los iconos, así como los ficheros *.stl que los representan, antes de simular la planta. Esto puede realizarse de manera manual a través de la ventana de comandos de MATLAB. Sin embargo, para facilitar al alumno una rápida inserción y modificación de los parámetros de cada icono *Wobj* y *Tool/Load*, se ha usado la función *Piezas()*.

Aunque a diferencia de las cargas y herramientas, los objetos de trabajo no presenten TCP, las variables usadas van a ser las misma, para usar una función general que sirva para ambas. Se han usado dos directorios del sistema: *Biblioteca* (con los ficheros *.stl de los *Tool/Load* y *Wobj* originales) y un directorio auxiliar *auxStl* para guardar los ficheros *.stl de la estación.

Esto nos va a permitir que, al llamar a la función, a la variable asociada al *.stl de un icono, *figIcono*, se le asocie uno de los ficheros *.stl originales, nombrado como *figPieza,* perteneciente a la Biblioteca. Esto va a ser especialmente útil cuando a esa variable se le quiera cambiar de fichero *.stl, por ejemplo, al cambiar de carga.

```
function Pieza(figIcono, figPieza)
% Devuelve una variable de nombre el texto de figIcono a la base.
    if nargin<1
        error('Pieza(figIcono, figPieza)')
    end
    pieza.base= zeros(1,6);
    pieza.tcp= zeros(1,6);
    pieza.masa= 0;
    pieza.cdg= [0,0,0];
    pieza.inercia= [1,1,1]*1e-3;
    pieza.body= '';
    pieza.color= [rand(1,3), 0.7];
    if nargin==1
       figPieza= 'Null';
    end
    pieza.figIcono= figIcono;
    eval(['! copy .\Biblioteca\', figPieza,'.stl  ','.\auxStl\', figIcono,'.stl'])
    eval(['! copy .\Biblioteca\', figPieza,'.png  ','.\auxStl\', figIcono,'.png'])
    % Devuelve el valor en la variable de la base figIcono
    assignin('base', figIcono, pieza);
end
```

Figura 45: Función Pieza()

La función crea y devuelve, almacenando en el *Workspace* de MATLAB, todos los campos de la variable estructura definidos por defecto. A partir de aquí, el alumno puede modificar los campos que él considere.

Además, como se ha comentado, esta función crea el campo *FigIcono* que representa el *.stl asociado al icono. Gracias a este se modifica el aspecto de los iconos de cargas y herramientas, así como el de los objetos de trabajo.

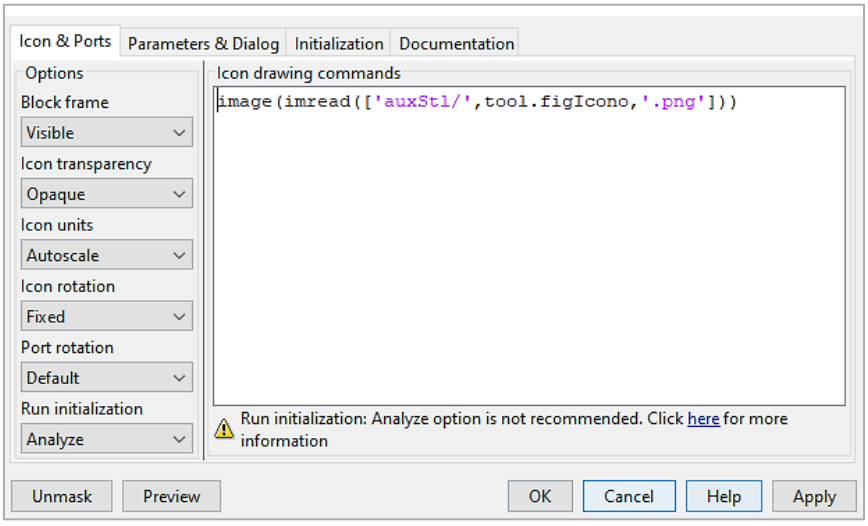


Figura 46: Ventana Icon & Ports de la máscara de iconos de trabajo

3.1.3 OTROS ELEMENTOS

A parte de los iconos creados, se han añadido a la librería un conjunto de bloques que contribuyen a la formación de una estación robótica:

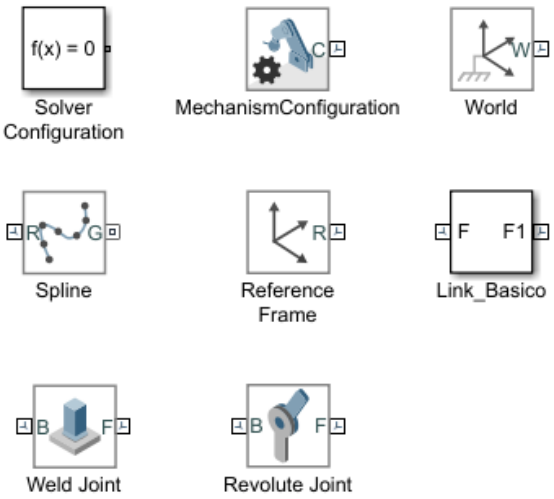


Figura 47: Bloques incluidos en la librería

Estos bloques pertenecen a la librería Simscape Multibody, o han sido creados a través de ellos, caso del bloque *Link_Basico*.

Todos los bloques de la *figura 47* se han ido detallando a lo largo de este capítulo, a excepción del bloque *Spline*, que va a permiter establecer la pose inicial del rastreador. También se han añadido los bloques que representan las articulaciones, *Weld Joint y Revolute Joint*, y la estructura básica de un *Link* para aquellos modelos que requieran de su incorporación.

3.1.4 IMPORTACIÓN A MATLAB (*Robotic System Toolbox*)

Una vez diseñada la estación a través de los iconos, se procede a la importación de esta a MATLAB, a través de la Robotic System Toolbox, de manera que podamos obtener su representación mediante un modelo de árbol de cuerpo rígido (RigidBodyTree).

La obtención en Matlab del modelo importado desde SIMULINK se lleva a cabo a través del siguiente comando:

>>robot= importrobot('NombreEstacion.slx')

Una vez importada la estructura robótica, a través del comando *show* podemos obtener su representación gráfica, de manera simple y genérica, puesto que, como hemos visto,

esta librería no destaca en cuanto al entorno de simulación, solo permitiendo la representación de ejes. Sin embargo, permitirá analizar las cinemáticas inversas y directas del robot, objetivo principal de su obtención.

A través del comando *showdetails* se muestran los detalles de los cuerpos (*bodies*) que forman el modelo. Esto va a permitir ver en detalles qué cuerpos son las herramientas y cargas. Aparecen numerados por orden de aparición en SIMULINK.

```
--------------------
Robot: (8 bodies)

 Idx     Body Name           Joint Name           Joint Type     Parent Name(Idx)   Children Name(s)
 ---     ---------           ----------           ----------     ----------------   ----------------
   1          base       base_link-base                fixed       base_link(0)
   2        link_1              joint_1             revolute       base_link(0)   link_2(3)
   3        link_2              joint_2             revolute          link_1(2)   link_3(4)
   4        link_3              joint_3             revolute          link_2(3)   link_4(5)
   5        link_4              joint_4             revolute          link_3(4)   link_5(6)
   6        link_5              joint_5             revolute          link_4(5)   link_6(7)
   7        link_6              joint_6             revolute          link_5(6)   tool0(8)
   8         tool0         joint6-tool0                fixed          link_6(7)
--------------------
```

Figura 48: Ejemplo de estructura RigidBody de la estación del robot irb120

Es preciso señalar que la librería *Robotic Toolbox* de Matlab también permite la representación de una estructura robótica a partir de un fichero URDF, obteniendo el modelo importado a través del mismo comando, *importrobot.* De hecho, esta librería dispone de ficheros URDF de robots industriales ya guardados.

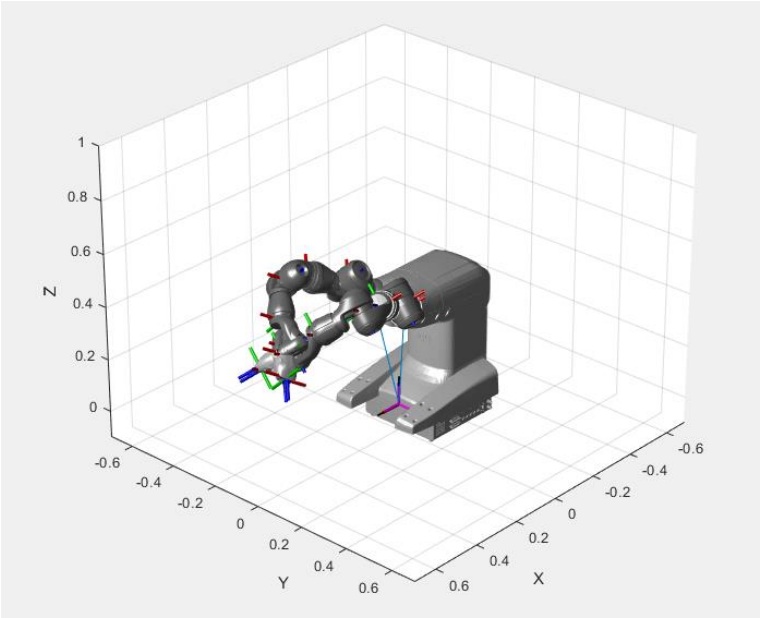


Figura 49: Robot YuMi de la librería Robotic Toolbox de MATLAB

3.1.5 EJEMPLOS DE ESTACIÓN

Con los iconos disponibles en la librería se ha creado un conjunto de ejemplos de estaciones que pueden iniciarse directamente a través de una llamada desde la ventana de comandos de Matlab a un fichero *.mat, donde se han determinado todos los parámetros de las máscaras de los iconos que las componen.

Ejemplos:
>> *Estacion_Ejemplo_dobleirb120DH_Init*

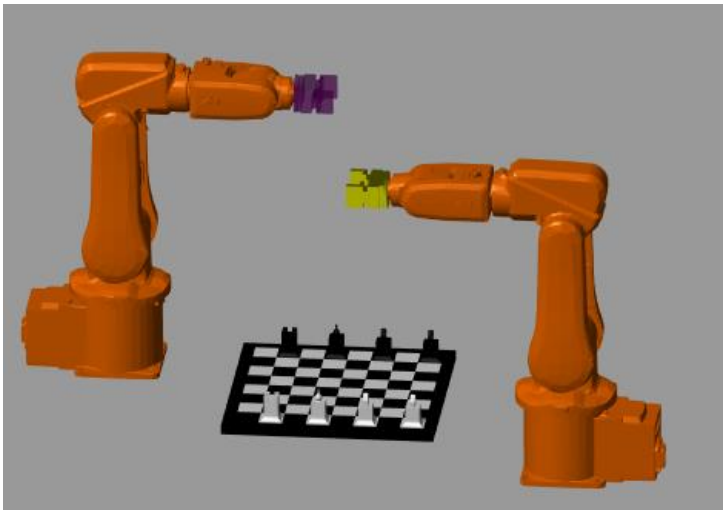


Figura 50: Visualización de la estación del primer ejemplo

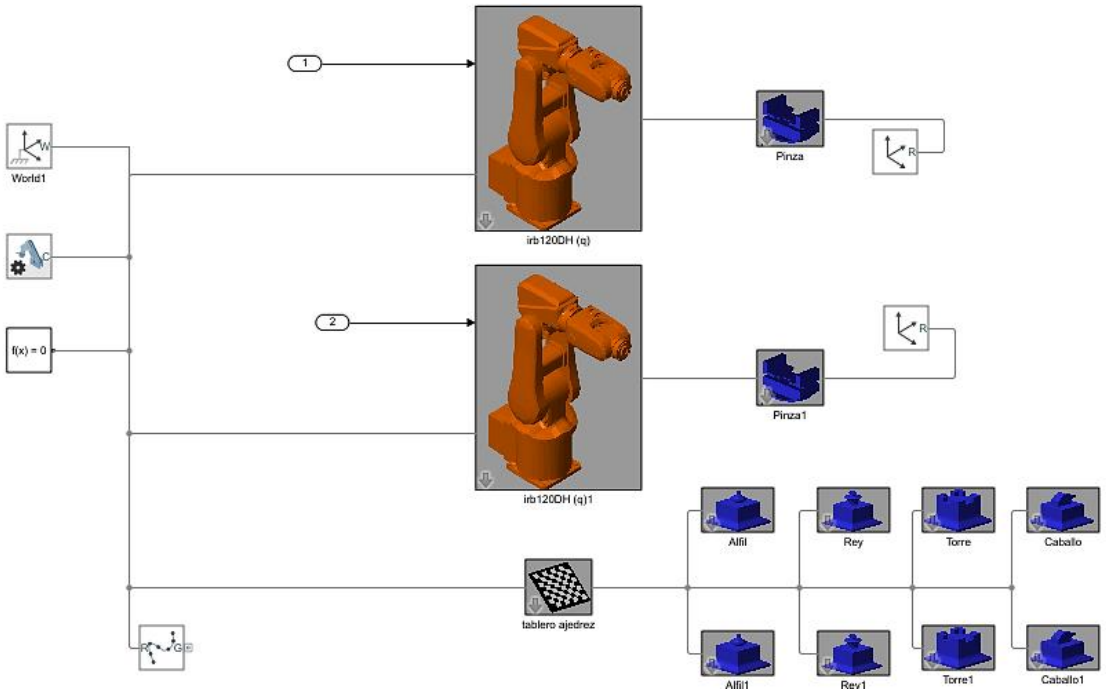


Figura 51: Modelo en SIMULINK de la estación del primer ejemplo

>>Estacion_Ejemplo_UniversalU3_Init

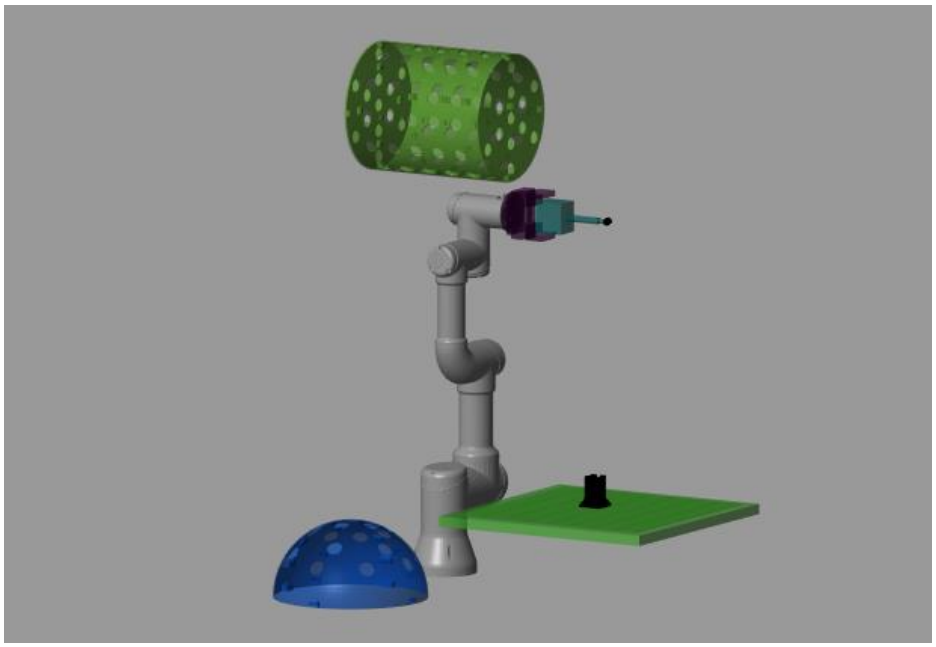


Figura 52: Visualización de la estación del segundo ejemplo

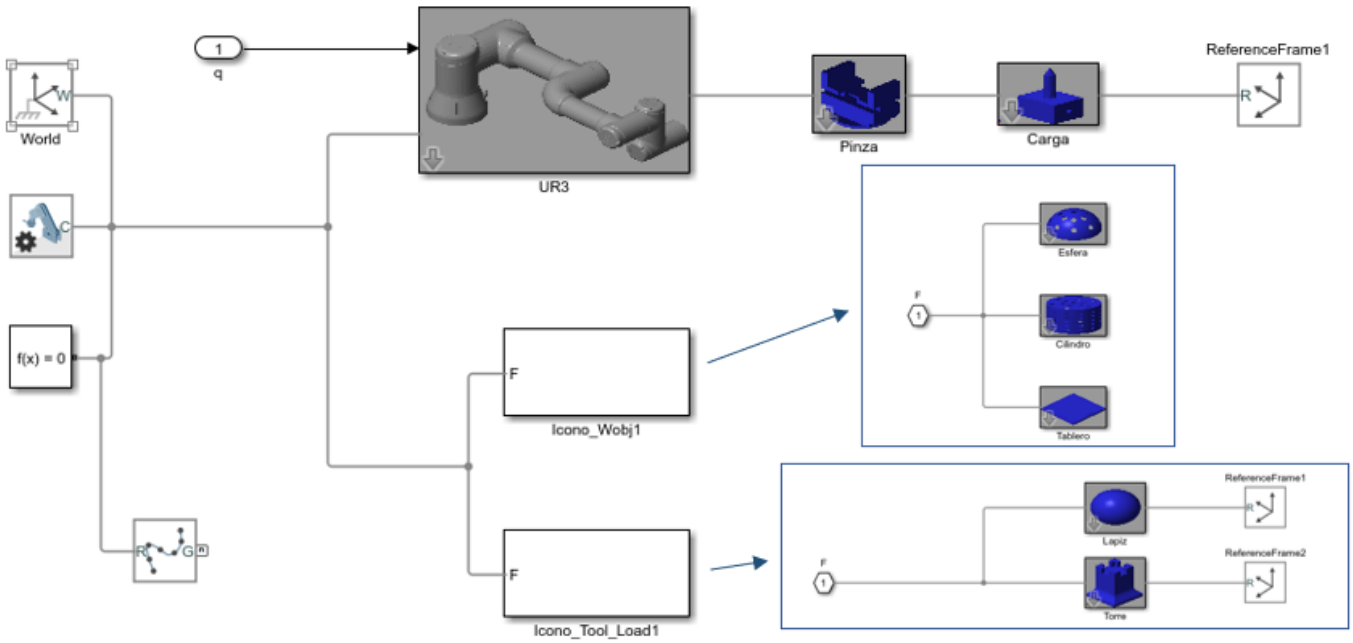


Figura 53: Modelo en SIMULINK de la estación del segundo ejemplo

3.2 HERRAMIENTAS DE LOCALIZACIÓN ESPACIAL

Un robot manipulador está compuesto por una cadena de eslabones conectados en serie a través de articulaciones. Uno de los extremos de esta cadena cinemática presentará una unión con el eslabón fijo que establecerá la base del robot, mientras que, en el otro extremo, se puede llevar la herramienta o efector final que realiza la función asignada al robot. Esta morfología provoca que, debido al movimiento relativo entre los distintos nudos que forman el robot, el efector final adquiera una determinada posición y orientación en el espacio. El estudio de ambas permitirá entender y desarrollar los modelos cinemáticos y dinámicos de los robots móviles.

En robótica es fundamental representar la posición y orientación de sólidos rígidos en el entorno respecto a una referencia fija, ya sea tanto el efector final de un robot como obstáculos o piezas de trabajo.

Para encontrar la localización espacial de un sólido rígido, se le asigna un sistema de referencia cartesiano, trirrectangular y dextrógiro, refiriendo su posición y orientación con respecto a un sistema de referencia fijo previamente establecido, pudiéndose utilizar distintos modos o herramientas para especificar la relación entre la posición y orientación del cuerpo y los sistemas de referencia.

3.2.1 MATRICES DE ROTACIÓN

Las matrices de rotación son el método más utilizado para describir las orientaciones de un sólido rígido. Se obtienen expresando los vectores de un sistema deseado {B} con respecto al marco de referencia {A}.

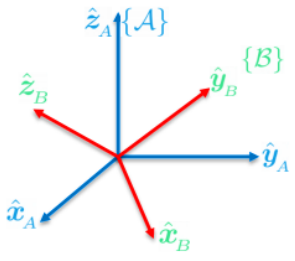


Figura 54: Orientación de un sistema {B} con respecto a otro {A}

Cada vector cuenta con 3 elementos, formando las columnas de la matriz de dimensión 3x3, R_B^A. El sistema rota un vector definido en el sistema {B} a otro descrito respecto al sistema {A}.

$$\begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} = R_B^A \cdot \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} \tag{3.1}$$

La matriz de rotación cumple con una serie de condiciones: todas sus columnas cumplen la condición de ortogonalidad dos a dos y tienen módulo unidad, cumpliéndose que $R^{-1} = R^T$ y *det(R)*=1.

Las matrices de rotación ortonormales para un ángulo q alrededor de los ejes coordenados son:

$$R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{pmatrix} \tag{3.2}$$

$$R_y(\theta) = \begin{pmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{pmatrix} \tag{3.3}$$

$$R_z(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

3.2.1 COMPOSICIÓN DE ROTACIONES

Un conjunto de matrices de rotación puede multiplicarse en un orden determinado obteniendo como resultado una secuencia de rotaciones alrededor de ciertas direcciones. Debe de tenerse en cuenta que el producto de rotaciones no es conmutativo, por lo que el orden en el que se realicen las rotaciones es relevante en el resultado final obtenido.

El teorema de Euler dice que se puede transformar un sistema de coordenadas en otro mediante una secuencia de máximo tres rotaciones alrededor de sus ejes, no produciéndose sucesivamente dos rotaciones sobre el mismo eje. Es necesario conocer tanto los valores de los ángulos de rotación como los ejes sobre los que se realizan los giros. Existen diversas configuraciones, uno de los más utilizados se muestra a continuación:

Ángulos de Euler ZYZ

Se compone de:

1. Rotación de un ángulo ϕ_1 alrededor del eje z.

2. Rotación del sistema obtenido un ángulo ϕ_2 alrededor del eje y'.

3. Rotación del sistema obtenido un ángulo ϕ_3 alrededor del eje z''.

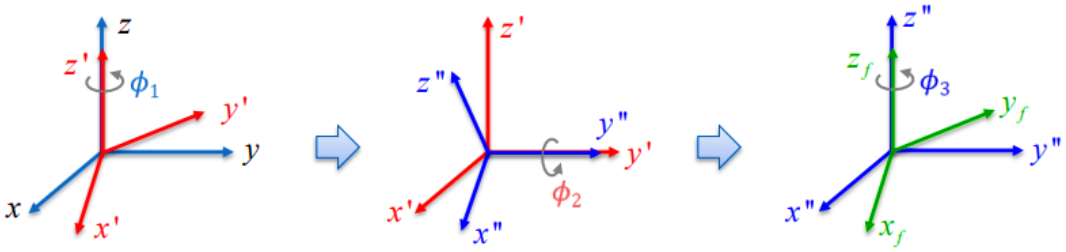


Figura 55: Ángulos de Euler ZYZ

Con la combinación de las tres rotaciones se obtiene:

$$R(\phi) = R_z(\phi_1)R_{y'}(\phi_2)R_{z''}(\phi_3) =$$

$$= \begin{bmatrix} C_{\phi_1}C_{\phi_2}C_{\phi_3} - S_{\phi_1}S_{\phi_3} & -S_{\phi_1}C_{\phi_3} - C_{\phi_1}C_{\phi_2}S_{\phi_3} & C_{\phi_1}S_{\phi_2} \\ S_{\phi_1}C_{\phi_2}C_{\phi_3} + C_{\phi_1}S_{\phi_3} & C_{\phi_1}C_{\phi_3} - S_{\phi_1}C_{\phi_2}S_{\phi_3} & S_{\phi_1}S_{\phi_2} \\ -S_{\phi_2}C_{\phi_3} & S_{\phi_2}S_{\phi_3} & C_{\phi_2} \end{bmatrix} \quad (3.5)$$

3.2.2 CUATERNIOS

Los cuaternios pueden ser utilizados para trabajar con giros y orientaciones. Un cuaternio Q está formado por cuatro componentes ($q0$, $q1$, $q2$, $q3$) que representan las coordenadas del cuaternio en la base {*e, i, j, k*}. Se distingue entre el componente escalar del cuaternio, *q0*, y el componente vectorial, *q1, q2, q3*. De esta manera, un cuaternio puede ser representado como:

$$Q = q0e + q1i + q2j + q3k = (s, v) \quad (3.6)$$

Donde s representa la parte escalar y v la parte vectorial.

Para la aplicación de cuaternios como representación de orientaciones, se asocia "el giro de un ángulo θ sobre el vector K" al cuaternio, definiéndolo como:

$$Q = Rot(K, \theta) = (cos\frac{\theta}{2}, ksin\frac{\theta}{2}) \quad (3.7)$$

3.2.2 MATRICES DE TRANSFORMACIÓN HOMÓGENEA

Para expresar la posición relativa en el espacio de un cuerpo hace falta, a parte de la rotación, emplear otra transformación, la traslación.

Las matrices de transformación homogénea permiten la representación conjunta de posición y orientación. Se trata de una matriz de dimensión 4x4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro.

Puede considerarse que está compuesta por 4 submatrices: una correspondiente a la matriz de rotación, otra que corresponde al vector de traslación, una submatriz que representa una transformación de perspectiva a (*(0,0,0)* en el caso de robótica), y una submatriz que representa un escalado global (*1* en el caso de robótica):

$$T = \begin{bmatrix} R_{3X3} & P_{3X1} \\ f_{1X3} & w_{1X1} \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix} = \begin{bmatrix} Rotación & Traslación \\ 0 & 1 \end{bmatrix} \quad (3.8)$$

Con esta matriz se pude cambiar el sistema de referencia respecto al cual se expresa la posición de un punto en el espacio, permitiendo describir la relación entre dos sistemas de referencia mediante rotaciones y translaciones. Se debe recordar que, debido a que la multiplicación de matrices no es conmutativa, el orden de las sucesivas rotaciones y traslaciones es importante.

3.2.2 APLICACIÓN EN MATLAB

Durante este estudio se va a ver la clase *Hmat()*. Se trata de una herramienta basada en las funciones de la librería *Robotic Toolbox* de Peter Corke [3]. Esta es utilizada en Matlab para llevar a la práctica el tema tratado, permitiendo establecer lar coordenadas, giros y movimientos relativos entre ejes de sólidos.

Esta clase presenta funciones que permiten definir y modificar la localización espacial de un cuerpo. Las más relevantes en este trabajo se han recogido en la *tabla 3*.

Hmat	Crea una matriz homogénea objeto.
H	Muestra la matriz homogénea del sólido. También permite introducir esta como argumento.
mtimes	Multiplica la posición de dos objetos, introducidos como argumentos, devolviendo la matriz homogénea resultante.
Txyz	Transforma los vectores de puntos, *x,y,z* , introducidos como argumentos, en la matriz homogénea correspondiente.
t	Obtiene la matriz homogénea a partir de los puntos *x,y,z* introducidos como argumento. En caso de no introducir argumentos, muestra el vector de traslación.
R	Obtiene la matriz de rotación.
Rx, Ry, Rz	Cada una de estas funciones permite aplicar una rotación sobre la matriz homogénea en el eje x, y o z, respectivamente.
tRzyz, tRxyz, tRzyx,tRyxz,	Estas funciones permiten obtener la matriz homogénea a partir de la matriz de rotación equivalente a los ángulos de Euler correspondientes: ZYZ, XYZ, ZYX, YXZ. También proporciona los vectores de traslación y ángulos de rotación de una determinada matriz.
tQ	Obtiene el vector traslación y los cuatro componentes de un cuaternio.
T3pts	Genera un eje a partir de la definición de tres puntos: dos puntos del eje x y un punto del eje y.
Plot	Muestra la posición de los ejes en MATLAB. Permite introducir la longitud y ancho de estos, así como el nombre.
DH	Concierte una matriz DH en una matriz de transformación homógenea.

Tabla 3: Funciones de la clase Hmat()

A continuación, se verán una serie de ejemplos de aplicación de esta clase.

DEFINICIÓN, ROTACIÓN Y TRASLACIÓN DE EJES

```
figure; T= Hmat(); % Matriz objeto comodín
T0= Hmat(); % Eje en origen absoluto
mag= [2,2] % [log, ancho] de los ejes
T0.Plot(mag,'A')
%traslación en x,y
T1= Hmat(T.t(5,5,0))
T1.Plot(mag,'B');
%rotación en z
T2=Hmat(T.Rz(45))
T2.Plot(mag,'   ,C');
% Traslación y rotación
T3= Hmat(T.t(0,2,4)*T.Rz(30));
T3.Plot(mag,'D');
% rotación y traslación
T4= Hmat(T.Rz(-30)*T.t(3,0,0));
T4.Plot(mag,'E');
% matriz homogénea resultante de rot. y trasl.
T4 =
      R                 | t
-------------------------------
 0.866,   0.500,   0.000 |   2.598
-0.500,   0.866,   0.000 | -1.500
 0.000,   0.000,   1.000 |   0.000
```

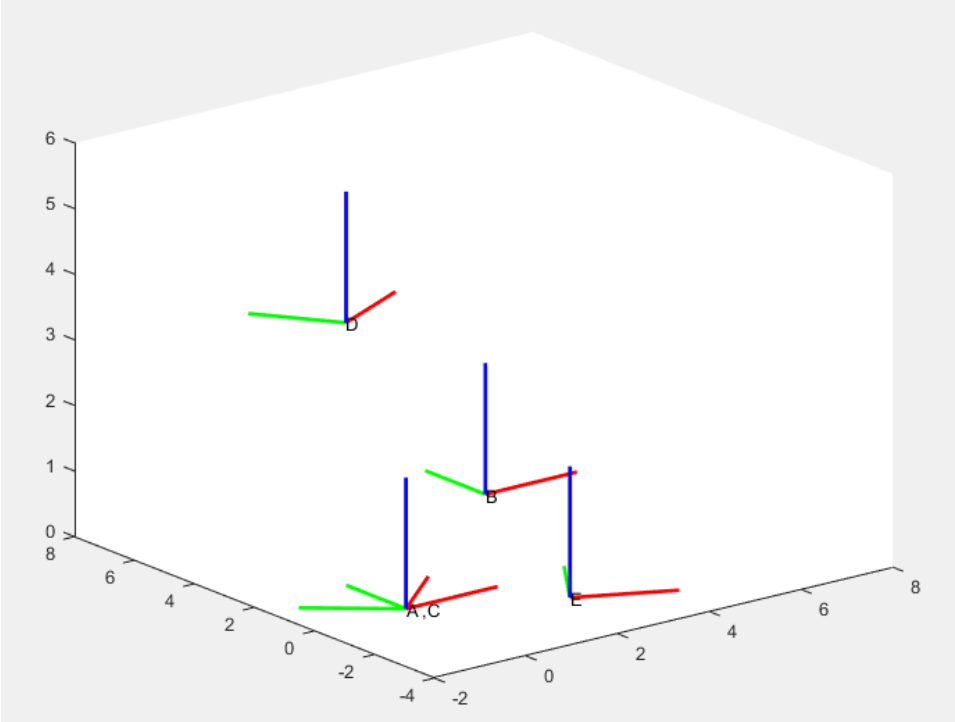


Figura 56: Ejes definidos

- Nube de puntos definida respecto a ejes A,B y E

```
pts0= [2*rand(1e3,2), zeros(1e3,1)]; %nube de puntos
%puntos referidos respecto a B
pts1= T1.Txyz(pts0);
%puntos referidos respecto a E
pts2= T4.Txyz(pts0);
plot3(pts0(:,1), pts0(:,2), pts0(:,3),'.b')
plot3(pts1(:,1), pts1(:,2), pts1(:,3),'.g')
plot3(pts2(:,1), pts2(:,2), pts2(:,3),'.r')
```

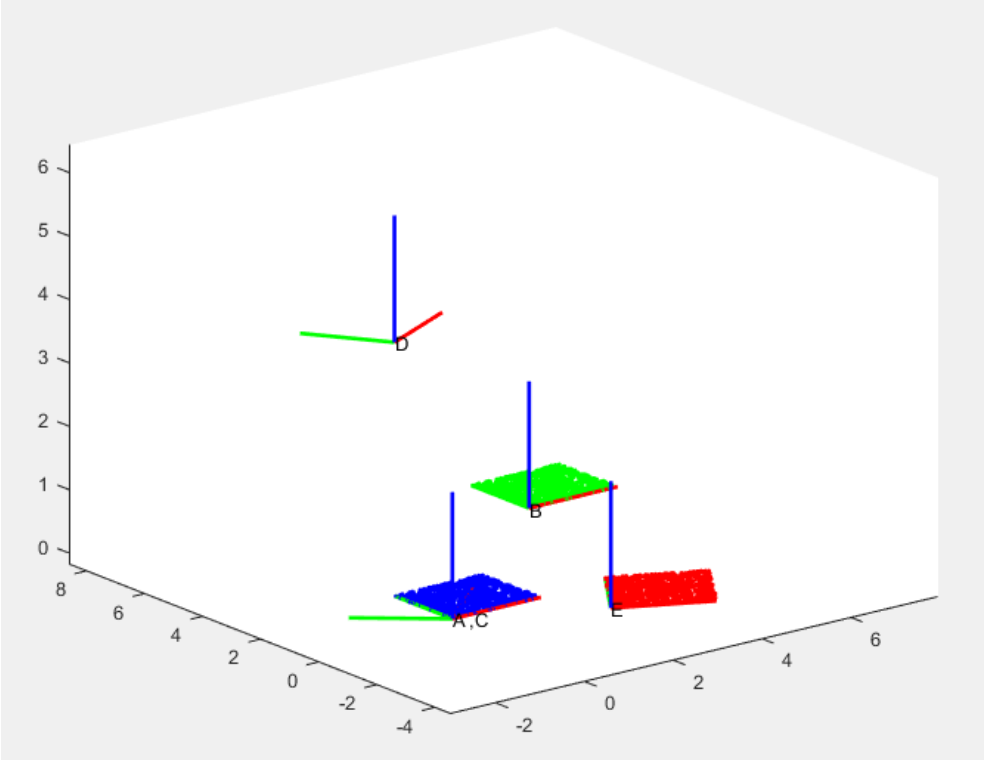


Figura 57: Nubes de puntos

EJE DEFINIDO A PARTIR DE 3 PUNTOS

```
x1= [0,.2,0];
x2= [-.2,.3,0];
y= [-.1,0,0.1];
pts= [x1; x2; y];
% Generación de los ejes

figure
T= Hmat(); T.T3pts(pts); T.Plot(0.3);
plot3(pts(:,1), pts(:,2), pts(:,3), ...
                'bo', 'LineWidth', 2)
```

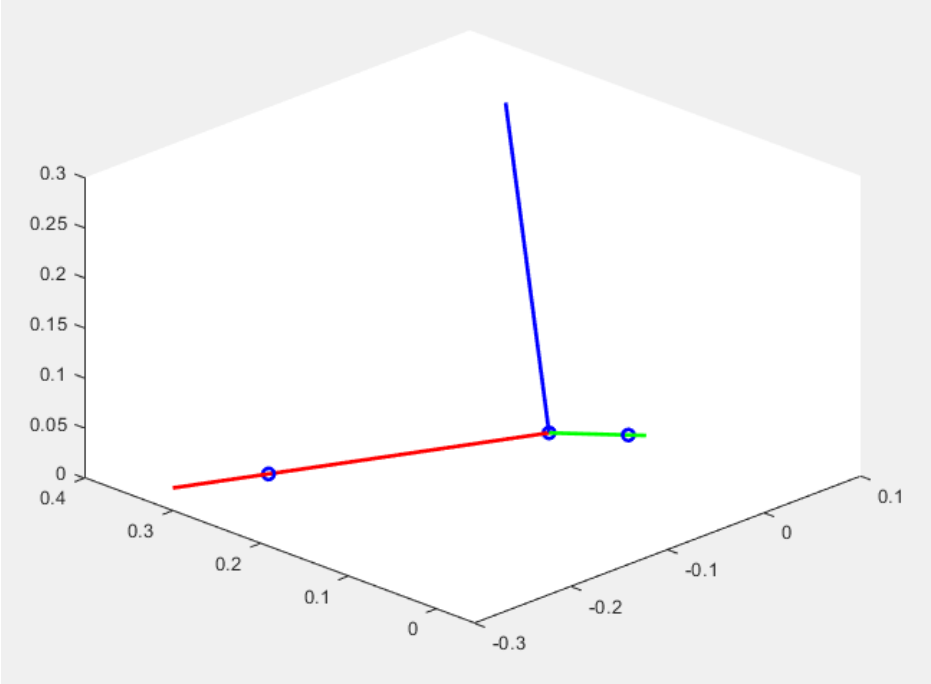


Figura 58: Eje definido a partir de 3 puntos

REPRESENTACIONES

```
>> punto= [[150,-200, 70]*1e-3, [0,0,180]*pi/180];
>> h=Hmat;          %matriz objeto
>> h.tRzyx(punto)    %matriz equivalente ángulos de Euler ZYX
ans =
    1.0000         0         0    0.1500
         0    0.9985   -0.0548   -0.2000
         0    0.0548    0.9985    0.0700
         0         0         0    1.0000


>> h.R             %matriz de rotación
ans =
    1.0000         0         0
         0    0.9985   -0.0548
         0    0.0548    0.9985

>> h.t              %vector traslación
ans =
    0.1500   -0.2000    0.0700

>> h.tRzyz(punto)    %matriz equivalente ángulos Euler ZYZ
ans =
    0.9985   -0.0548         0    0.1500
    0.0548    0.9985         0   -0.2000
         0         0    1.0000    0.0700
         0         0         0    1.0000
```

```
>> h.tQ;          %vector traslación y cuaternios
[[0.150, -0.200,0.070], [1.00,0.00,0.00,0.03]]

>> h.Rz(pi/2)      %rotación en z de 90 grados
ans =
    0.9996   -0.0274         0         0
    0.0274    0.9996         0         0
         0         0    1.0000         0
         0         0         0    1.0000
```

TRAYECTORIAS

Para definir la trayectoria de la herramienta de un robot se debe definir los puntos y orientación de dicha herramienta. La librería *Hmat* puede interpolar entre dos ejes sus posiciones y rotaciones.
Esta incluye funciones que permiten la realización de trayectorias

- ***JTraj***: Realiza una interpolación lineal entre dos posiciones del tipo [x,y,z], introduciendo como argumento el número de puntos interpolados. Permite restringir la velocidad a un valor inicial y final.

```
pos1= 0; pos2=15; %posiciones
v1=0; v2=3; % Velocidades iniciales
nPts= 50; % Puntos interpolados
h= Hmat; h.Rad;
[p,v,a]= h.jTraj(pos1,pos2,nPts,v1,v2);
figure
plot([p,v,a])
```

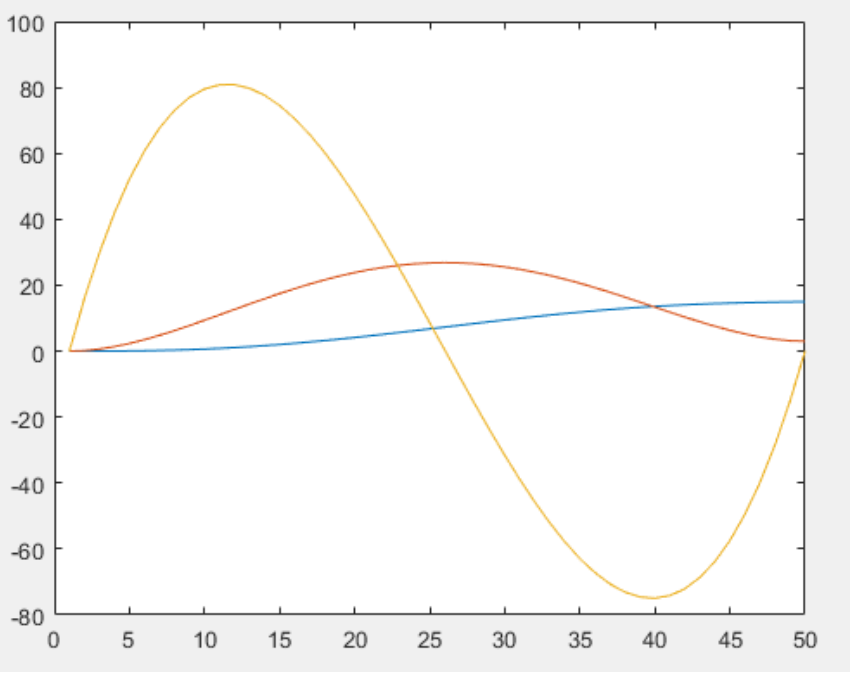


Figura 59: Trayectoria definida con función jTraj

- ***tRzyxTraj*:** Normaliza los ángulos para hallar la interpolación por el lugar más cercano. Se interpola un número determinado de puntos desde la posición 1 a la posición 2 en traslación y rotación, del tipo [x, y, z Rz, Ry, Rx].

```
%posiciones
pose1= [[0,0,0]*1e-3,[0,0,0]*pi/180];
pose2= [[300,200,0]*1e-3,[0,0,180]*pi/180];
h= Hmat;h.Rad;
nPts= 15; % Puntos interpolados
pose= h.tRzyxTraj(pose1,pose2,nPts);
figure
hold on
for i=1:nPts
    h.tRzyx(pose(i,:));
    h.Plot
end
eje= 200*[-1,1,-1,1,-1,1]*1e-3;
axis(eje)
```

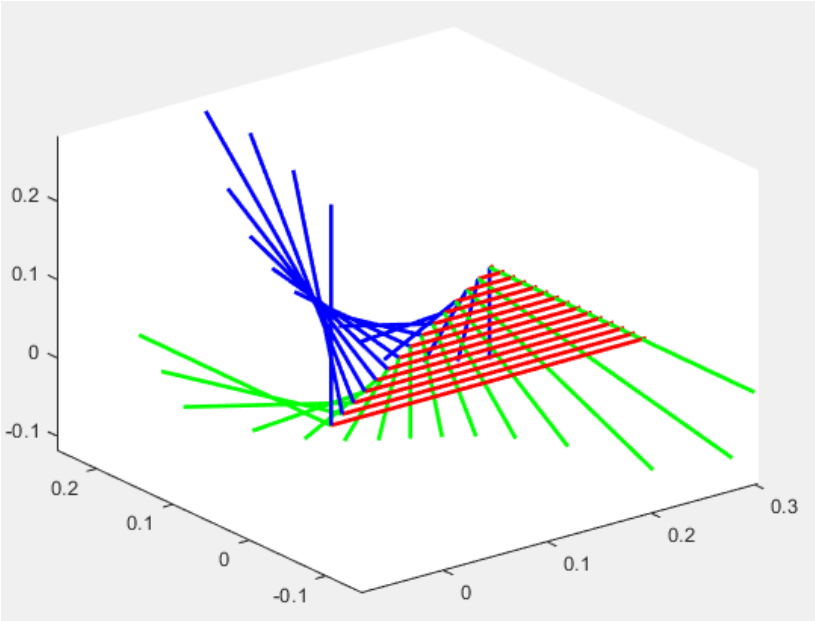


Figura 60: Trayectoria definida con tRzyxTraj

3.2.3 IMPLEMENTACIÓN CON MODELO EN SIMULINK

La generación de una estación formada por piezas estáticas y su posterior simulación permite al alumno analizar, de una manera más visual, la colocación y movimientos relativos de cada pieza con respecto a un determinado sistema de referencia. Esto le va a permitir comprender la relación de movimientos que van a tener lugar a la hora de desarrollar los modelos cinemáticos de los robots manipuladores.

Para su análisis, se ha creado en SIMULINK una estación formada por un tablero y unas piezas de ajedrez. Como se muestra en la *figura 61*, el rey y el caballo están acoplados al tablero, lo que va a provocar que sus movimientos sean relativos respecto al eje de este, mientras que el resto de los objetos; tablero, torre y alfil, se posicionarán en cada movimiento con respecto al sistema global de referencia.

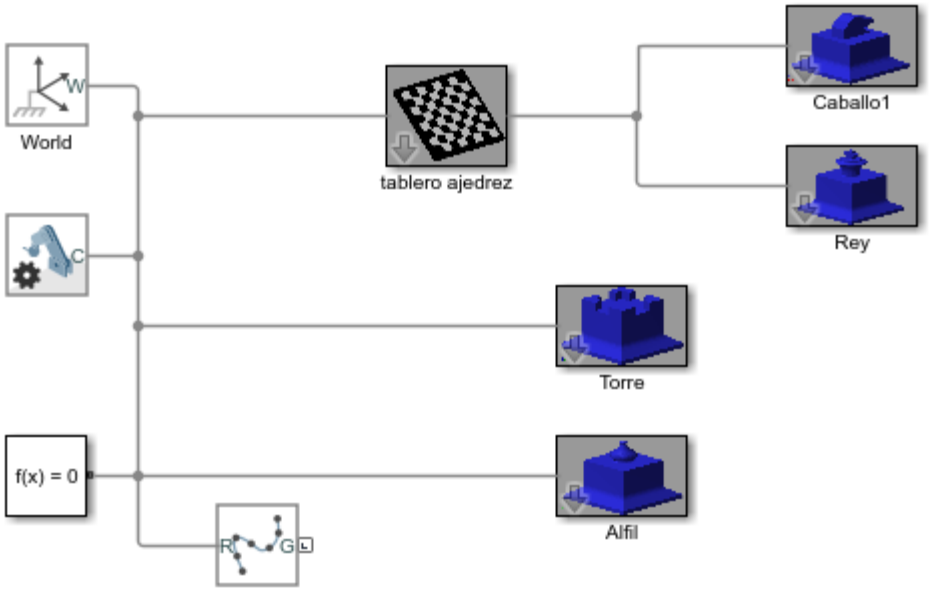


Figura 61: Esquema en SIMULINK de Estacion_ajedrez

Se establece la base de cada pieza en una posición particular:

- Tablero, torre y alfil:
  La posición se establece con respecto al *World Frame.*

```
tab_ajedrez.base= [[0,200,0]*1e-3, [0,0,0]*deg];
Torre.base= [[225,75, 0]*1e-3, [0,0,0]*deg];
Alfil.base= [[300,-200, 0]*1e-3, [0,0,0]*deg];
```

- Caballo y rey:
  La posición se establece con respecto al eje del tablero.

```
Rey.base= [[225,75,0]*1e-3, [0,0,0]*deg];
Caballo.base=([[125,75,0]*1e-3, [0,0,0]*deg]);
```

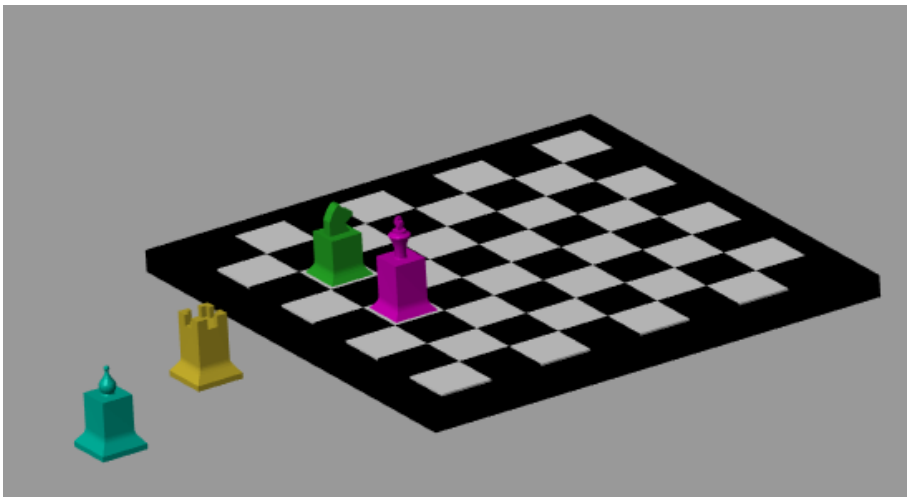


Figura 62: Primera simulación de Estacion_ajedrez

Las coordenadas de la base de la figura de la torre y el rey presentan los mismos valores, sin embargo, vemos en la *figura 62* que estas no coinciden debido a los ejes a los que están referenciados.

Se modifica la posición del tablero; giro de 30 grados sobre el eje z y traslación sobre eje y,z.

```
tab_ajedrez.base= [[0,100,100]*1e-3, [0,0,30]*deg];
```

En la *figura 63* se observa que, al modificar la posición del tablero, se modifica la posición del caballo y el rey de manera instantánea. La matriz que define la base del tablero cambia, pero la de estas piezas no, puesto que se mantienen invariables respecto al sistema de coordenadas del tablero; no hace falta aplicar sobre ellas la misma rotación y traslación que ha sufrido este.

```
h= Hmat;
h.tRzyx(tab_ajedrez.base)
 ans =
    1.0000         0         0         0
         0    1.0000   -0.0091    0.1000
         0    0.0091    1.0000    0.1000
         0         0         0    1.0000
```

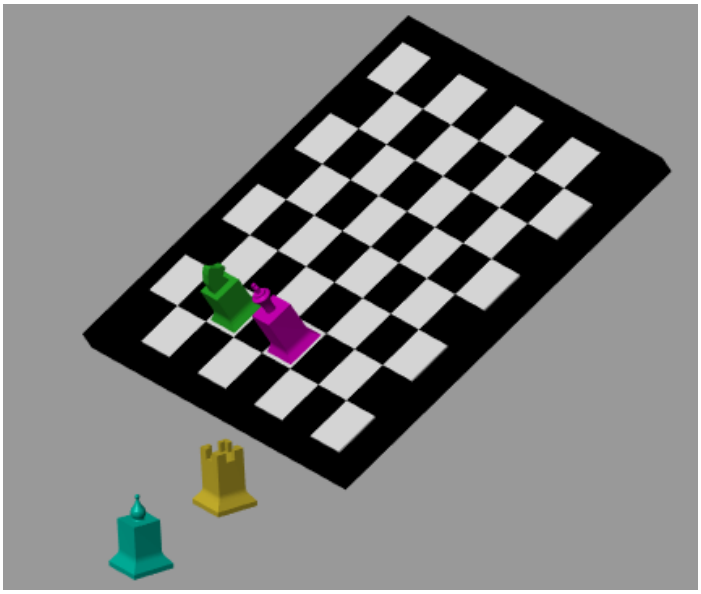


Figura 63: Segunda simulación de Estacion_ajedrez

En este caso, vamos a cambiar de posición a las piezas.

```
Caballo.base= [[175,175,0]*1e-3, [0,0,0]*deg];
Rey.base= [[325,375,0]*1e-3, [0,0,0]*deg];
Alfil.base= [[325,375, 70]*1e-3, [0,0,0]*deg]
```

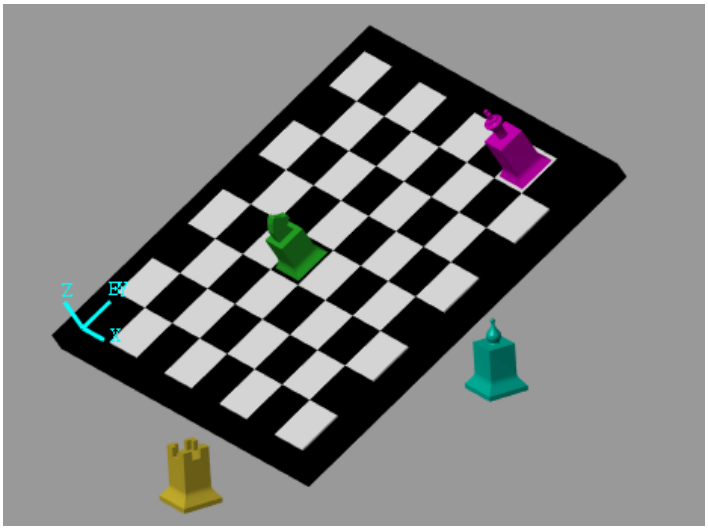


Figura 64: Tercera simulación de Estacion_ajedrez

El caballo y el rey se mueven por el tablero con la misma inclinación que tiene este. Este movimiento es relativo al eje del tablero, mostrado en la *figura 64*. El alfil ha adoptado otra posición dentro del espacio del sistema de coordenadas global.

Finalmente, se pretende posicionar la torre y el alfil sobre el tablero, realizando los movimientos con especto a su eje, al igual que la figura del caballo y el rey. Para ello, se debe multiplicar la matriz equivalente a los ángulos de Euler ZYX del tablero de ajedrez, y la matriz obtenida a partir de la posición que quiere adoptar la figura dentro de este.

La matriz resultante permitirá obtener las coordenadas que posicionen la figura con respecto al sistema de referencia del tablero:

```
% Pone la torre en función del tablero
h= Hmat; h.Rad;
Px= 3; Py= 4;
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[25+50*Px,25+50*Py,7]*1e-
3,[0,0,0]*deg]));
Torre.base= h.tRzyx;

% Pone el alfil en función del tablero
h= Hmat; h.Rad;
Px= 1; Py= 2;
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[25+50*Px,25+50*Py,0]*1e-
3,[0,0,0]*deg]));
Alfil.base= h.tRzyx;
```

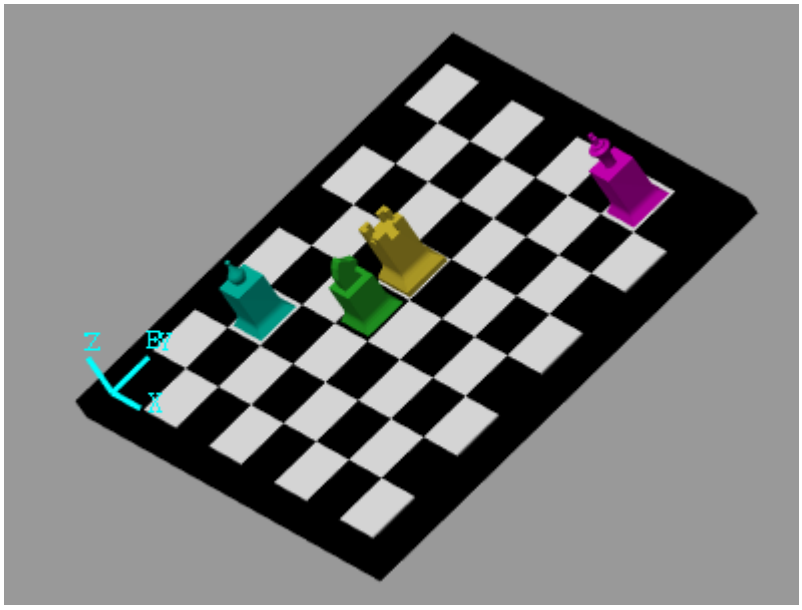


Figura 65: Cuarta simulación de Estacion_ajedrez

3.3 CINEMÁTICA DEL ROBOT

La cinemática de robots estudia los movimientos en el espacio de estos con respecto a un sistema de referencia fijo previamente establecido. Analiza los valores tomados por las coordenadas articulares del robot, relacionados con la posición y orientación del extremo final de este.

Dentro de la cinemática de robots se pude distinguir entre cinemática directa e inversa. La primera trata de determinar la posición y orientación del efector final del robot con respecto a un sistema de coordenadas tomado como referencia, todo ello conociendo los valores de las articulaciones y los parámetros geométricos del robot. La cinemática directa es fácil de resolver, existiendo siempre una solución para ella.
Por otra parte, la cinemática inversa determina la configuración que debe ser adoptada por el robot para alcanzar la posición y orientación del extremo, ambas conocidas. La cinemática inversa supone un problema más complejo, presenta singularidades y no linealidades, no existiendo siempre una solución en forma cerrada.

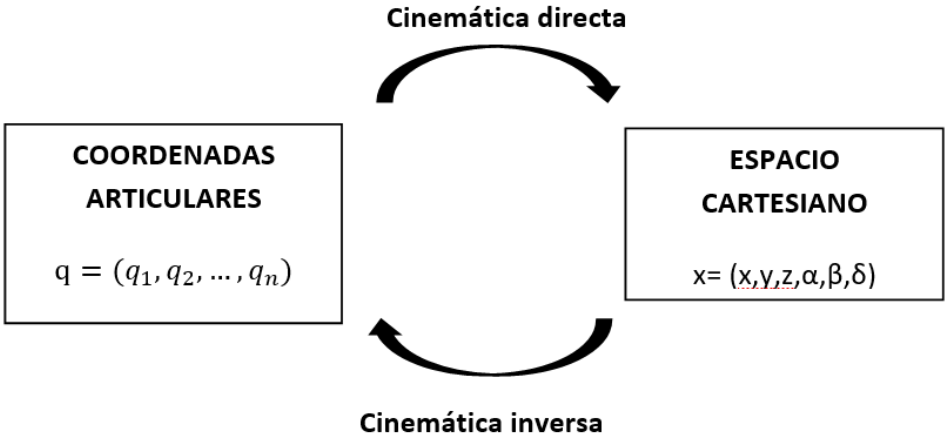


Figura 66: Cinemática directa e inversa

Puesto que se trata de un estudio cinemático, no se tienen en cuenta las fuerzas dinámicas presentes en el movimiento.
La cinemática del robot también busca encontrar la relación entre las velocidades del movimiento de las articulaciones y las del efector final, este es lo que se llama método diferencial [22].

3.3.1 PROBLEMA CINEMÁTICO DIRECTO

Como se ha mostrado en apartados anteriores, la notación Denavit-Hartenberg permite expresar la posición y orientación del extremo final de un robot manipulador mediante una matriz de transformación homogénea. Se trata de un método sistemático para describir la cinemática directa de robots manipuladores.

Un robot manipulador está formado por una serie de eslabones enumerados desde 0 (base del robot) hasta el eslabón i (extremo del robot). Las articulaciones se enumeran desde 1 hasta i.
Cada una de ellas posee un grado de libertad, siendo la articulación i la que permita el movimiento relativo entre el eslabón $i-1$ y el eslabón i.
La localización espacial de los eslabones está expresados a través de matrices de transformaciones homogéneas, definidas respecto a los distintos sistemas de referencia locales. Considerando las transformaciones consecutivas que tienen lugar sobre la cadena de eslabones adyacentes, se puede determinar la posición y orientación del elemento terminal de la siguiente manera:

$$T = A_1^0(q_1) \cdot A_2^1(q_2) \ldots\ldots A_i^{i-1}(q_i) \qquad (3.9)$$

Donde T es la matriz de transformación homogénea que determina la localización del efector terminal con respecto al sistema de referencia de la base del robot. Esta es la llamada ecuación cinemática del robot. La solución es única para la mayor parte de los robots seriales.

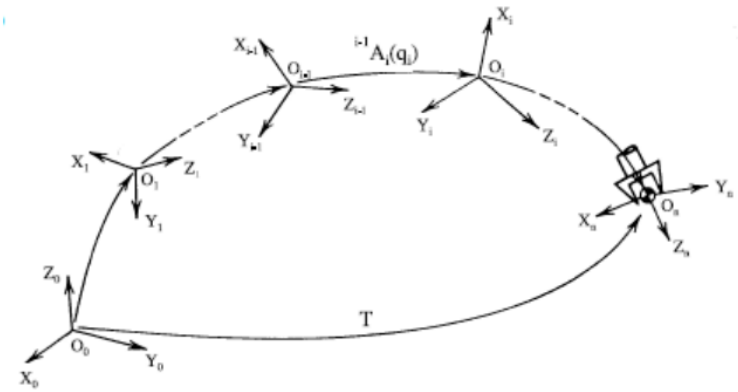


Figura 67: Localización espacial del extremo del robot

Otro procedimiento para la resolución del modelo cinemático directo es el llamado método geométrico. Se trata de un método no sistemático, válido únicamente para robots de pocos grados de libertad o configuraciones muy concretas como los robots planares. Se basa en el análisis trigonométrico del robot para una posición determinada. Asigna de manera arbitraria los sistemas de referencia de cada eslabón, describiendo cada sistema con respecto del anterior.

3.3.1.1 Resolución mediante modelo de simulación en SIMULINK

La resolución de la cinemática directa puede abordarse a través de la simulación de un sistema dinámico representado por un modelo de SIMULINK. Esta simulación muestra el comportamiento del sistema al mover el robot a una posición articular determinada, en función de un tiempo determinado, adquiriendo el efector final, como resultado, una posición y orientación concreta en el espacio cartesiano.

Para su estudio y comprobación, se ha creado en SIMULINK, usando los iconos de la librería, un modelo de estación compuesta por el robot industrial *Staubli TX90*, al que se ha acoplado como herramienta una pinza:

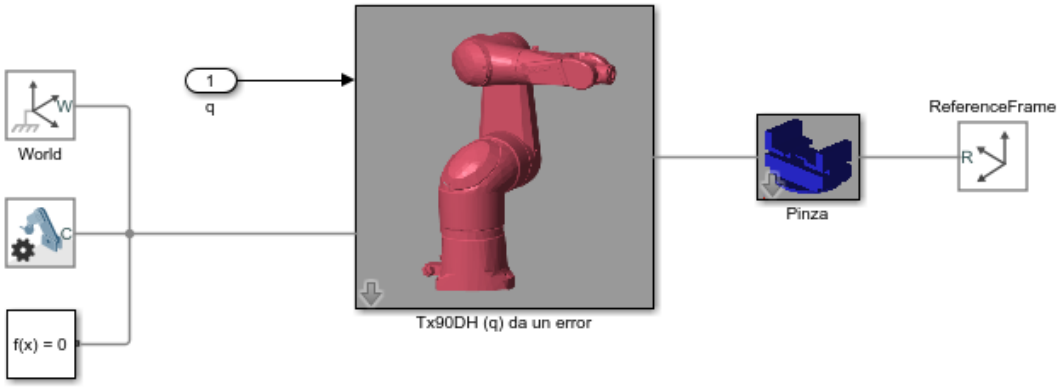


Figura 68: Estación en SIMULINK del robot Staubli TX90

Tras definir las variables asociadas al modelo, se procede a su importación en Matlab.
Para su simulación, se ha crea un vector de 1000 puntos, espaciados uniformemente en el intervalo [0-7] como representación del tiempo en segundos (comando *linspace*).
Para la representación de cada articulación, se ha generado un conjunto de vectores, también de 1000 puntos cada uno de ellos, espaciados de manera uniforme entre la posición inicial, 0, y la posición final que se quiere adoptar, en radianes.
Mediante el comando *sim* se procede a la simulación del modelo, a través de la interpolación de los puntos definidos durante el tiempo establecido.

Ejemplos:

1) Partiendo de una posición inicial, se buscan los siguientes movimientos de los ejes del robot:

   -movimiento de 0 a 30 grados alrededor del eje 1
   -movimiento de 0 a 30 grados alrededor del eje 2
   -movimiento de 0 a -30 grados alrededor del eje 3

-movimiento de 0 a 50 grados alrededor del eje 4
-movimiento de 0 a 30 grados alrededor del eje 5
-movimiento de 0 a -40 grados alrededor del eje 6

```
nPts= 1e3;
t= linspace(0,7,nPts)';
% movimiento de ejes
q= zeros(nPts,6);
q(:,1)= linspace(0,30,nPts)'* pi/180;
q(:,2)= linspace(0,30,nPts)'* pi/180;
q(:,3)= linspace(0,-30,nPts)'* pi/180;
q(:,4)= linspace(0,50,nPts)'* pi/180;
q(:,5)= linspace(0,30,nPts)'* pi/180;
q(:,6)= linspace(0,-40,nPts)'* pi/180;
opt= []; % parámetros de simulación por defecto
sim(Estacion, t, [], [t,q])
```

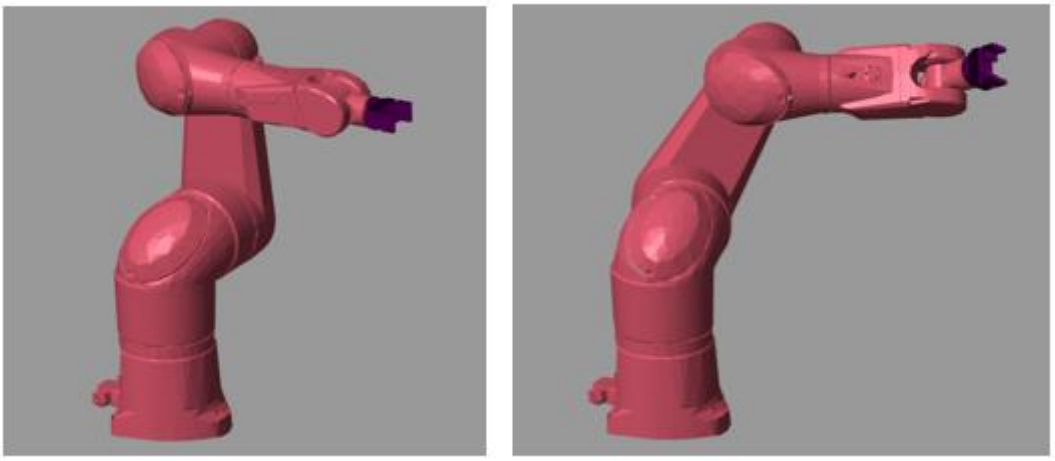


Figura 69: Posición inicial y final del robot Staubli TX90 (CD)

2) Partiendo de una posición inicial, se buscan los siguientes movimientos de los ejes del robot:
-movimiento de 0 a 50 grados alrededor del eje 2
-movimiento de 0 a 30 grados alrededor del eje 3
-movimiento de 0 a 50 grados alrededor del eje 6

```
nPts= 1e3;
t= linspace(0,7,nPts)';
% movimiento de ejes
q= zeros(nPts,6);
q(:,2)= linspace(0,50,nPts)'* pi/180;
q(:,3)= linspace(0,30,nPts)'* pi/180;
q(:,6)= linspace(0,50,nPts)'* pi/180;
opt= []; % parámetros de simulación por defecto
sim(Estacion, t, [], [t,q])
```

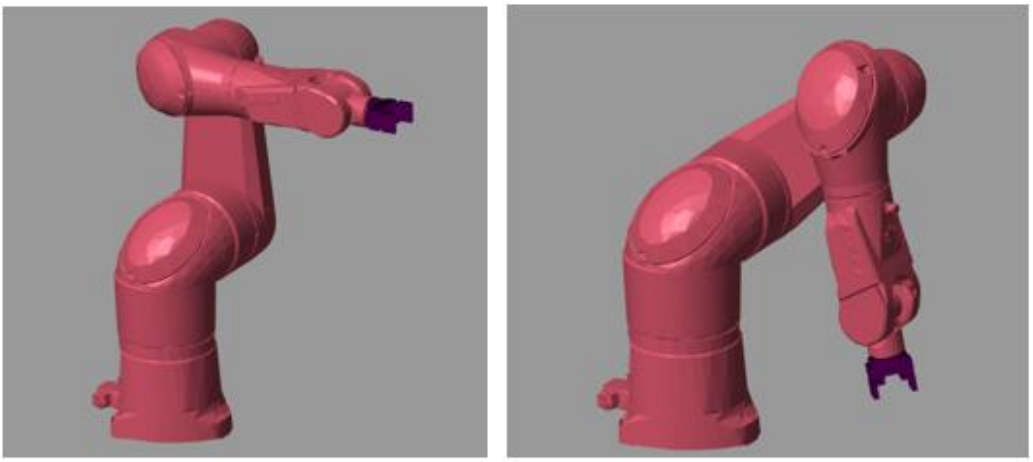


Figura 70: Posición inicial y final (2) del robot Staubli TX90 (CD)

3.3.2 PROBLEMA CINEMÁTICO INVERSO

El problema cinemático inverso halla los valores articulares que deben ser adoptados por el robot para que su extremo se encuentre posicionado y orientado según una determinada localización espacial. En este caso, la obtención de las ecuaciones cinemáticas no es sistemática y depende estrechamente de la configuración específica de cada robot. Resueltas estas ecuaciones, se realiza el movimiento previsto del efector final asignando a cada una de las articulaciones los valores obtenidos.

Mientras que en el problema Directo se obtiene una única solución, en el problema inverso pueden existir varias que sean compatibles con una misma localización del efector final. También puede ocurrir que, para una determinada estructura robótica, no exista solución explícita de las variables de las articulaciones. Debido a ello, y a la presencia de ecuaciones no lineales con numerosas funciones trigonométricas, la resolución de este problema tiene un alto coste en lo que al tiempo computacional se refiere.

Existen varios métodos para calcularla. Para algunas configuraciones de pocos grados de libertad o para el caso en el que se consideren solo los primeros grados de libertad, esta se puede hallar a partir del procedimiento geométrico mediante relaciones geométricas, típicamente trigonométricas.
Para un sistema de mayor número de grados de libertad los métodos geométricos pierden eficacia a medida que la adquieren otros métodos tales como el uso de matrices homogéneas. Es posible buscar el modelo cinemático inverso de un robot a partir de su modelo directo; conocida la matriz de transformación homogénea T en función de las coordenadas articulares (q1, q2, q3), se podría intentar manipular las ecuaciones resultantes de T para despejar estas.

Debido a su consistencia, el estudio de la cinemática se ha centrado principalmente en el método inverso ya que, con la librería desarrollada, se ha buscado proporcionar una herramienta que permita su resolución y pueda ser usada de manera universal.

3.3.2.1 Resolución mediante modelo de simulación en SIMULINK

Es posible obtener la resolución del problema cinemático inverso a través de un modelo generado en SIMULINK cuya simulación permite que, a través de la entrada del vector de coordenadas [x,y,z], se obtenga el valor de los ángulos articulares que van a permitir al robot realizar el movimiento adquiriendo la localización deseada de su efector final.

Los procesos de optimización utilizan una función de coste que penaliza desviaciones de la referencia y otros efectos indeseados. La librería Robotic System Toolbox de MATLAB da acceso a algoritmos de optimización local de la cinemática inversa que permite obtener una configuración de robot que alcance unos determinados objetivos y restricciones. Se tratan de métodos iterativos que parten de una conjetura inicial en la solución y buscan minimizar la función de coste, de tal manera que, si alguno converge a una configuración en la que el coste es cercano a cero dentro de una tolerancia especificada, ha encontrado una solución al problema de la cinemática inversa. En caso de no encontrarla, se produce un reinicio aleatorio que reinicia la búsqueda iterativa hasta encontrar la solución, o hasta que transcurra un tiempo máximo o límite de iteración [21].

Partiendo de esto, para el desarrollo del modelo, se usan dos bloques de la librería Robotics System Toolbox:

- ***Inverse Kinematics***

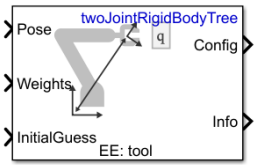


Figura 71: Bloque Inverse Kinematics de la librería Robotics System Toolbox

Utiliza un solucionador de la cinemática inversa para realizar el cálculo de las configuraciones articulares que logran una posición deseada del efector final.
El bloque presenta como entradas la matriz de transformación homogénea que representa la posición del efector final buscada, un vector de pesos que permite indicar qué objetivos optimizar y, un vector que representa un supuesto inicial de la configuración del robot. Los valores articulares correspondientes se obtienen a la salida del bloque.

- ***Coordinate Transformation Conversion***

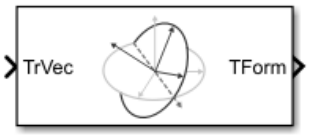


Figura 72: Bloque Coordinate Transformation Conversion de la librería Robotics System Toolbox

Transforma una representación de coordenadas de entrada en otra específica de salida. En el modelo será usado para transformar el vector de posición deseado en la matriz de transformación que se introducirá como entrada en el bloque *Inverse Kinematics*.

Con ambos bloques, se ha modelado el sistema compuesto por el robot industrial *Stabuli TX90*, al que se ha acoplado como herramienta una pinza:

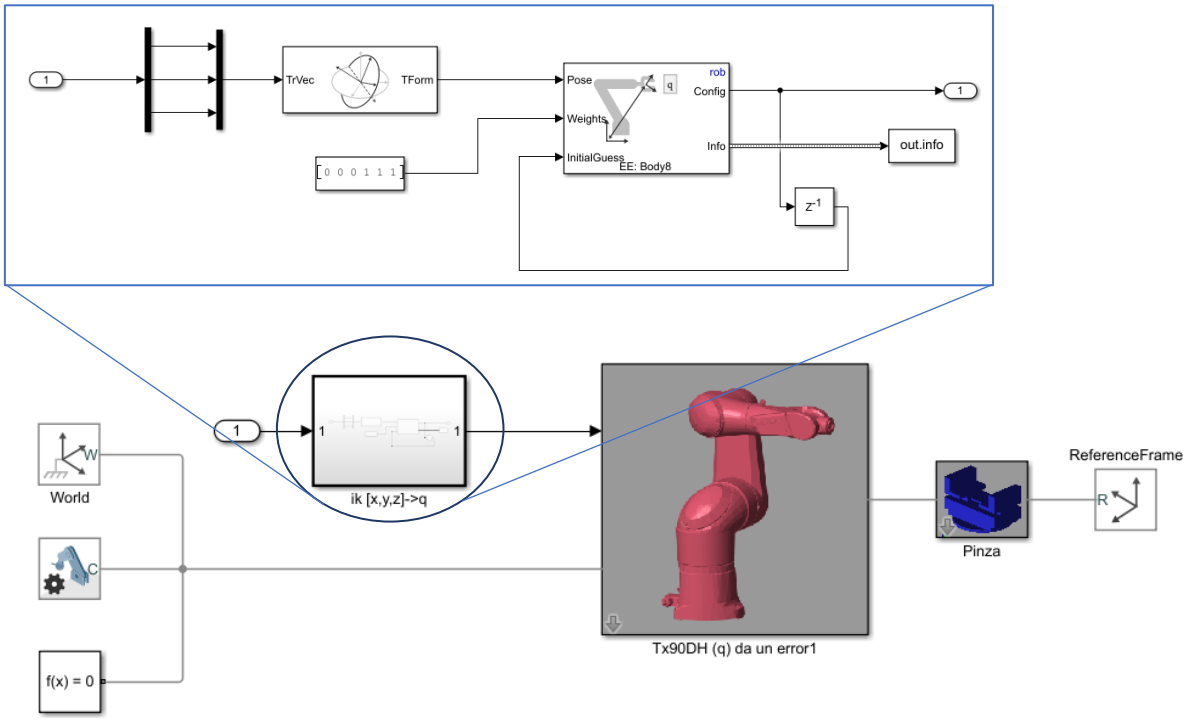


Figura 73: Modelo en SIMULINK con cinemática inversa

El modelo presenta un vector de pesos [0 0 0 1 1 1] puesto que en este caso se optimiza solo la posición (corresponde a [Rx, Ry, Rz, x, y, z]). En caso de optimizar posición y ángulos a la vez el vector de pesos correspondiente sería [1 1 1 1 1 1].
Además, este modelo presenta una singularidad; al bloque *Inverse Kinematics* se le necesita asociar como variable el propio objeto RigidBody. Este no puede definirse debido a la impedancia que supone él mismo a la hora de importar el modelo. Por ello, en primer lugar, se ha establecido dicho bloque como comentario, de tal manera que, tras la definición de las variables del modelo, se permita su importación y se pueda obtener el objeto RigidBody correspondiente.

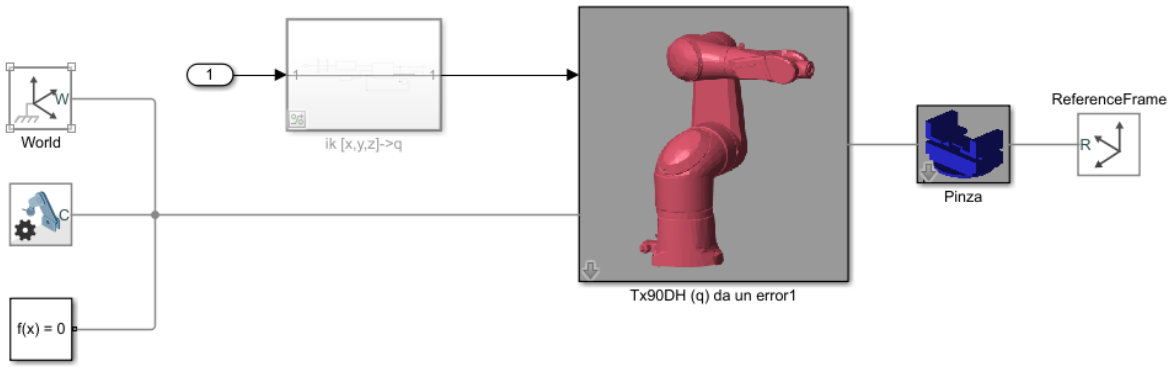


Figura 74: Modelo con bloque comentado

Obtenido el objeto, nombrado *rob*, se restaura el bloque comentado y se introduce este dentro del bloque *Inverse Kinematics,* seleccionando el efector final sobre el que se desea realizar la cinemática:

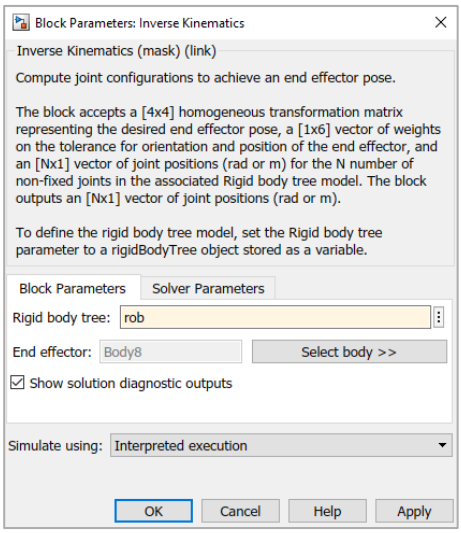


Figura 75: Parámetros del bloque Inverse Kinematics

De manera análoga al modelo cinemático directo, se simula el modelo durante un tiempo determinado a través del comando *sim,* mediante la interpolación de los puntos definidos; los vectores de coordenadas [x,y,z] y el tiempo de simulación.

Ejemplo:

1) Partiendo de una posición inicial, el robot se mueve por los siguientes valores cartesianos:
   -valor de x de 0.2 a 0.5.
   -valor de y de 0.2 a 0.3.
   -valor de z de 0.1 a 0.6.

```
nPts= 1e3;
tend= 4;
t= linspace(0,tend,nPts)';
% valores de entrada [x,y,z]
x= linspace(200*1e-3,500*1e-3,nPts)';
y= linspace(200*1e-3,300*1e-3,nPts)';
z= linspace(100*1e-3,600*1e-3,nPts)';
sim('Estacion_CI_ik.slx',t,[],[t,x,y,z]);
```

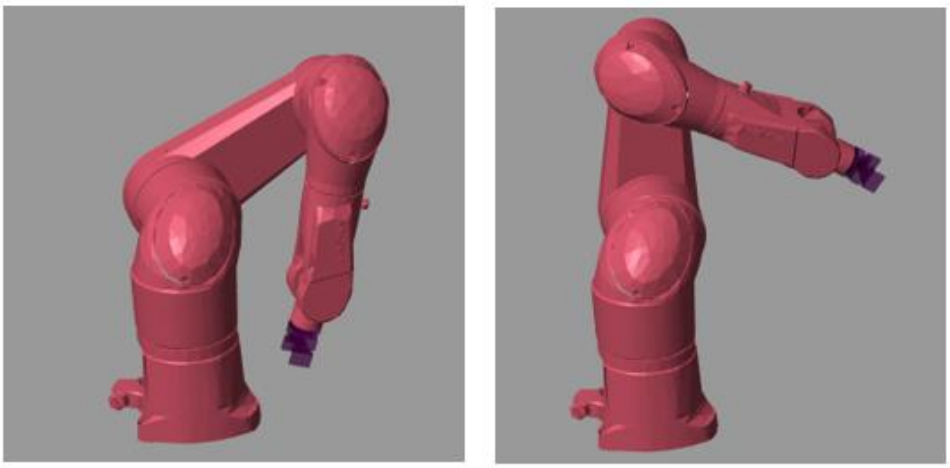


Figura 76: Posición inicial y final del robot Staubli TX90 (CI)

2) El robot describe una trayectoria circular, que empieza y concluye en el mismo punto.

```
nPts= 1e3;
tend= 2;
t= linspace(0,tend,nPts)';
radio= 200*1e-3;
% valores de entrada [x,y,z]
x= 400*1e-3+ radio*sin(2*pi*t/tend);
y= 200*1e-3+radio*cos(2*pi*t/tend);
z= 100*1e-3*ones(nPts,1);
sim('Estacion_CI_ik.slx',t,[],[t,x,y,z]);
```

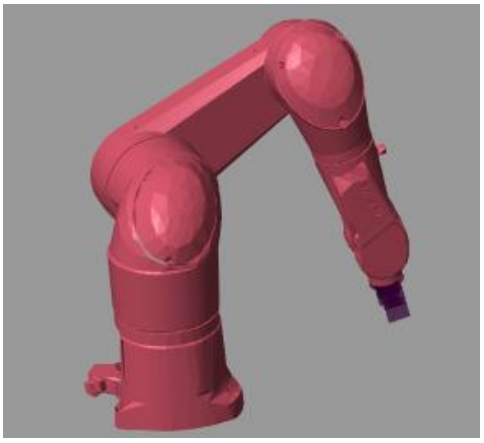


Figura 77: Posición inicial y final (2) del robot Staubli TX90 (CI)

3.3.3 OBJETO *KIN*

Para la resolución de la cinemática, además de los métodos llevados a la práctica mediante simulación vistos anteriormente, este proyecto se ha basado principalmente en su estudio mediante la denominada clase *Kin*. El profesor y tutor de este proyecto, Alberto Herreros López, creó esta herramienta con el objetivo de sincronizar las herramientas *Multi-Body* de SIMULINK y *Rigid-Body* de MATLAB para poder obtener un simulador de robot.

Los métodos principales de la clase *Kin* son los movimientos típicos de todo lenguaje para programación de robot: movimientos circulares, lineales y, por coordenadas articulares. Sin embargo, aunque su principal interés se encuentre en la cinemática del robot, principalmente inversa debido a su mayor complejidad, también presenta otras funciones que se verán a continuación y que, todo ello, proporcionará a la librería una herramienta esencial para la manipulación de la estación.

Se parte de un modelo robótico creado y definido en SIMULINK mediante *Multi-Body* y se obtiene su equivalente en *Rigid-Body,* definido a partir del primero. Se usa *MultiBody* para la simulación del modelo debido a que la simulación de movimientos es más continua. En este, se pueden añadir las herramientas que se deseen y los objetos de trabajo, ya que, aunque al modelo *Rigid-Body* también se le puedan añadir, no se va a visualizar y puede constar únicamente del robot, ya que solo se va a usar para la obtención de puntos para la cinemática. Como alternativa para la obtención de estos puntos, también se puede usar la *app Teacher rastreador* que veremos en el apartado siguiente.

La clase *Kin* está definida en un script de Matlab. Los métodos y funciones que presenta se apoyan para realizar cálculos en la herramienta *Hmat()*, basada en las funciones de la librería Robotic Toolbox de Peter Corke [3].

3.3.3.1 Funciones

Métodos de movimientos básicos

- **MoveAbsJ (Joint, [nPts]):** Mueve al robot a una posición articular dada con un movimiento no lineal del TCP.
  -Joint es la posición articular a la cual se desea mover el robot.
  -nPts es el número de puntos interpolados entre la posición actual y la deseada.

  Se puede definir una sola posición los puntos interpolados entre la posición actual y esta, o una matriz, donde en cada fila hay una posición articular. La diferencia es el tiempo de simulación. La posición es dada en radianes.

- **MoveJ (pose, [Hwobj], [nPts]):** Mueve el TCP del robot a un punto dado usando un movimiento no lineal. Algoritmo de optimización: el robot sigue el camino más sencillo.
  -pose es la posición en coordenadas cartesianas a la cual se desea mover el robot.
  -Hwobj es la matriz homogénea que representa el eje de referencia al que se relaciona los puntos dados.

  Se puede definir una sola posición y los puntos interpolados entre la posición actual y esta, o, una matriz donde en cada fila hay una posición en la forma $[x, y, z, r_z, r_y, r_x]$. Si la entrada *pose* es únicamente $[x, y, z]$ se mantiene la orientación y solo se optimiza la posición $[x, y, z]$. La posición es dada en mm y la orientación en radianes.

- **MoveL (pose, [Hwobj], [nPts]):** Mueve el TCP del robot de la posición actual a la dada siguiendo una trayectoria lineal. Presenta los mismos posibles argumentos de entrada que *MoveJ.*

  Interpola *nPts* desde la posición actual a la posición *pose* tipo $[x, y, z, r_z, r_y, r_x]$ en línea recta. Si solo presenta los elementos $[x, y, z]$ se mantiene la orientación. Pueden incorporarse los ejes del objeto de trabajo. La posición es dada en mm y la orientación en radianes.

- **MoveC (pos1, pos2, [Hwobj], [nPts]):** Mueve el TCP del robot de la posición actual pasando por un primer punto hasta un segundo describiendo un arco.
  -pos1 es la primera posición en coordenadas cartesianas a la cual se desea mover el robot.
  -pos2 es la segunda posición en coordenadas cartesianas a la cual se desea mover el robot.

  Ambas posiciones son de la forma $[x, y, z]$, siendo el ángulo perpendicular al plano definido por los tres puntos.

Métodos Internos

- **Pose([Joint]):** a partir de las posiciones articulares (*Nx6*) introducidas como argumento, la función devuelve las posiciones cartesianas (*Nx6*) correspondientes. Sin argumento de entrada devuelve la posición cartesiana actual del robot. *(Cinemática directa)*

- **Joint([Pose]):** a partir de las posiciones cartesianas (*Nx6*) introducidas como argumento, la función devuelve las posiciones articulares (*Nx6*) correspondientes. Sin argumento de entrada devuelve la posición articular actual del robot. *(Cinemática inversa)*
- **Robot ():** Devuelve el objeto *RigidBody* asociado al objeto *Kin*.
- **Sim (mdl, type, value):** Manda a SIMULINK los parámetros de simulación.
  -*mdl* es el nombre del fichero de SIMULINK.
  -*type* es el tipo de medida: intervalo de tiempo o velocidad fijo.
  -*value* es el valor de la velocidad o tiempo.

Métodos de visualización

- **Show ([Joint], [pose]):** Con el argumento *Joint* muestra el robot en la posición articular dada, mientras que con el argumento *pose* genera un rastreo de la posición del TCP, principalmente usado en los métodos de movimiento. Sin argumentos, muestra el robot en la posición actual.
- **Rec(value):** Controla la grabación de la simulación para una posible reproducción.
  -Sin argumentos resetea la información grabada y empieza a grabar.
  -Si value=1 empieza a grabar respetando los movimientos en memoria
  -Si value=0 deja de grabar.
- **Rep ():** Simula las posiciones grabadas.

Métodos de Rigid-Body

- **Tool ():** Permite realizar cambios de herramienta, o entre cargas y herramientas. Modifica el nombre del cuerpo del efector final del robot *Rigid-Body.*
- **Wobj, Body, DelBody:** Permite incluir y quitar cuerpos en el robot *Rigid-Body.* Sin embargo, no serán usados junto a *Multi-Body* ya que todos los cuerpos están definidos en el fichero de SIMULINK usado.

3.3.3.2 Aplicación

Se ha diseñado un modelo de estación para reflejar la resolución de la cinemática mediante la aplicación de las funciones de la clase *Kin* descritas.

Este modelo generado en SIMULINK, a través de los iconos de la librería, está compuesto por dos brazos robóticos *irb120* de ABB, a los que se han acoplado pinzas y, uno de ellos, presenta un lápiz como carga. Se ha incorporado al sistema un tablero y distintas piezas de ajedrez. El modelo es inicializado directamente a través del script *Estacion_Ejemplo_dobleirb120DH_Init.*

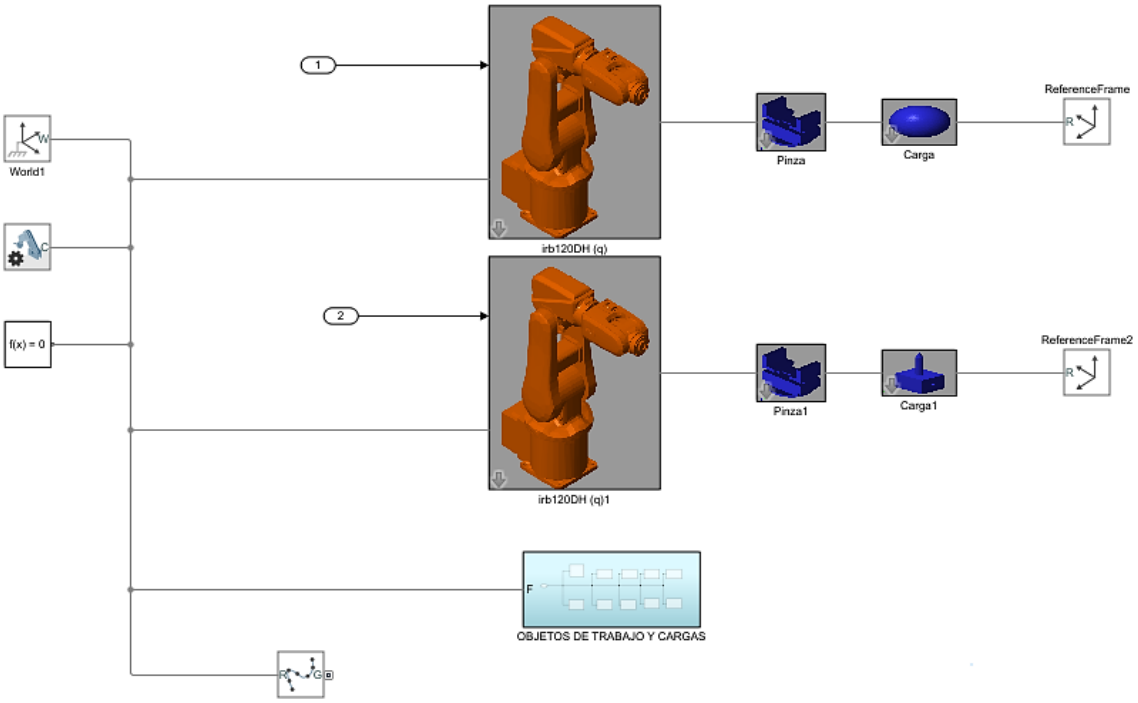


Figura 78: Modelo en SIMULINK de Estacion_doble_irb120Cin

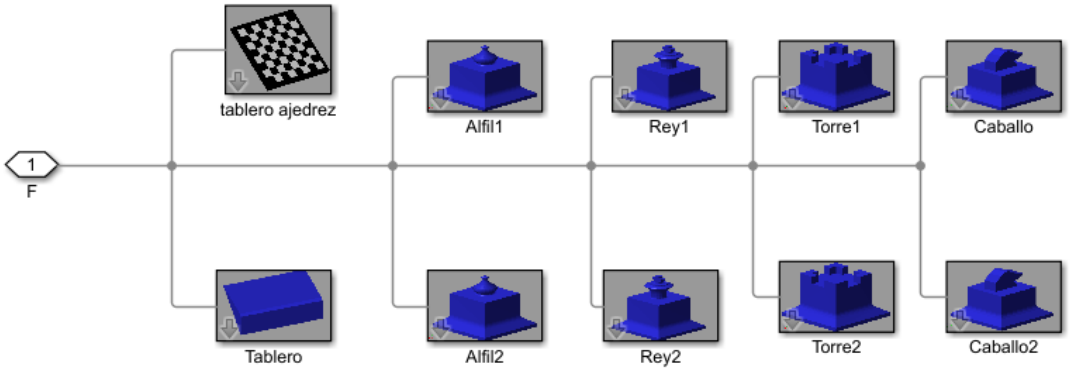


Figura 79: Interior del subsistema 'objetos de trabajo y cargas'

Tras la definición de las variables del modelo, y su posterior importación en MATLAB con *RigidBody*, se inicia la clase *Kin*. Se llama a la función introduciendo como argumentos el modelo *RigidBody* importado, el modelo de SIMULINK de la estación generada, el cuerpo del efector final sobre el que se va a desarrollar la cinemática y el vector de las posiciones articulares iniciales *q0*.

```
rob= Kin(robot, Estacion, Pinza1.body,q);
```

Posteriormente, se han realizado con el modelo una serie de acciones.
El primer robot va a partir de una posición de reposo y se va a mover hasta una posición cercana a donde se encuentra la pieza que quiere trasladar, la toma y deposita en otra casilla del tablero, regresando de nuevo a la posición de inicio. Por tanto, para la trayectoria a realizar se especifica:

- Coordenadas articulares de la posición de reposo del robot. (CD)
- Coordenadas cartesianas de la pieza en el tablero. (CI)
- Coordenadas cartesianas de la posición de la casilla en la que dejar la carga. (CI)

Posteriormente, se produce un cambio de herramienta entre robots que va a permitir que el segundo dibuje en la mesa con el lápiz las iniciales de la figura y color que han sido movidas. Se grabará la simulación de cada letra para su posterior reproducción total. En este caso, para la trayectoria a realizar se especifica:

- Coordenadas articulares de la posición de reposo del robot. (CD)
- Coordenadas cartesianas de los puntos de las letras a dibujar en la mesa. (CI)

Se ha desarrollado una función, *PegarEn()*, que va a permitir el intercambio de cargas y herramientas acopladas al robot con las cargas y herramientas asociadas a la base de la estación, almacenadas en este caso dentro del subsistema *objetos de trabajo y cargas.* Esto va a permitir que el primer robot, que en principio presenta una pinza y una carga vacía, sea capaz de tomar una determinada pieza al intercambiar las variables asociadas a esta con la carga hasta ahora vacía. El intercambio contrario posibilitará dejar la pieza. Esta función permite definir la nueva base de la carga al ser introducida como argumento.

```
function PegarEn(figIcono, piezal, base)
    if nargin<2
       error('[pieza2, piezal]= Pegar(figIcono, piezal, base)')
    elseif nargin==2
       base= zeros(1,6);
    end
    pieza2= piezal;
    pieza2.base= base;
    pieza2.figIcono= figIcono;
    eval(['! copy .\auxStl\',piezal.figIcono,'.stl  ','.\auxStl\',pieza2.figIcono,'.stl']);
    eval(['! copy .\auxStl\',piezal.figIcono,'.png  ','.\auxStl\',pieza2.figIcono,'.png']);

    assignin('base', figIcono, pieza2);
    Pieza(piezal.figIcono);
end
```

Figura 80: Función PegarEn ()

EJEMPLO: **Movimiento del caballo negro.**

A continuación, se especifican los pasos seguidos que han sido programados para la descripción de la trayectoria con ayuda del objeto *Kin*:

-Se adopta la posición articular de reposo de los robots mediante *MoveAbsJ.*

-Se calculan las coordenadas del punto situado por encima de la pieza a tomar. (*Cálculos realizados mediante herramienta Hmat()*).

- El primer robot se desplaza hasta ese punto siguiendo el camino más sencillo mediante *MoveJ*.

-Toma la pieza mediante el uso de la función *PegarEn(). (Intercambio de carga vacía por el caballo negro)*

-Se calculan las coordenadas del punto donde se quiere depositar la pieza.

-El robot se desplaza hasta ese punto siguiendo el camino más sencillo mediante *MoveJ*.

-Deposita la pieza mediante el uso de la función *PegarEn(). (Intercambio del caballo por la carga vacía)*

-Se adopta la posición articular de reposo de los robots mediante *MoveAbsJ*.

-Se cambia de robot a través de *Tool()*, adoptando como TCP la punta del lápiz del segundo robot.

-Se define el eje de la mesa a través de tres puntos. (*Cálculos realizados mediante herramienta Hmat()*).

-El robot se desplaza hasta un punto dado con respecto al eje de la mesa mediante *MoveJ*.

-Se inicia la grabación con *Rec()* y, mediante sucesivos movimientos lineales, no lineales y circulares, se dibujan las letras C-N.

-Se concluye la grabación con *Rec(0)* y se reproduce el dibujo de todas las letras.

-Se adopta de nuevo la posición articular de reposo de los robots mediante *MoveAbsJ.*

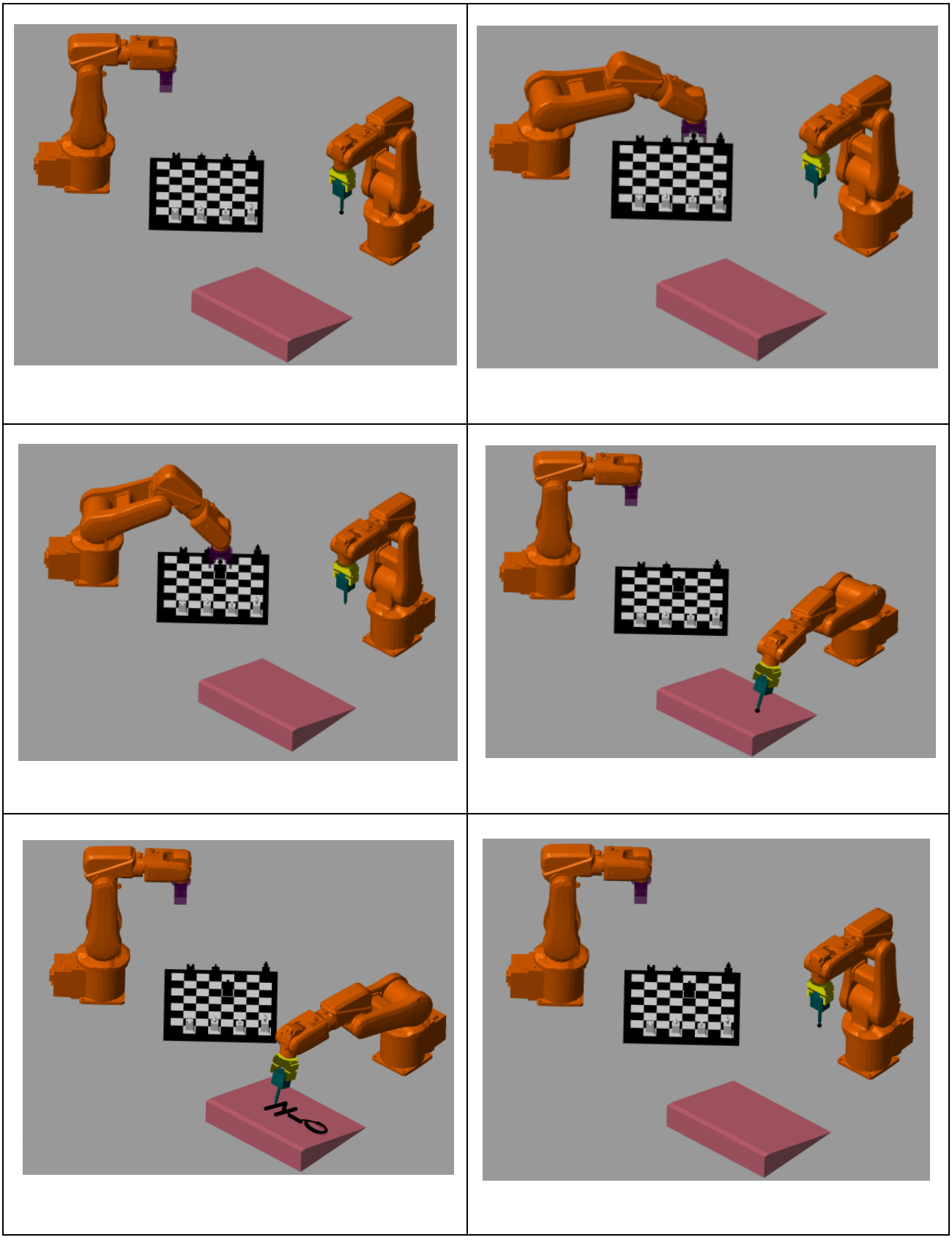


Tabla 4: Secuencia de la trayectoria simulada

Al cambiar de herramienta y que el segundo robot empiece a escribir respecto al eje de la tabla y no el global, los movimientos son más lentos y, como se puede observar en la *tabla 4*, entre las letras pueden aparecer interpolaciones no deseadas.

Debido al algoritmo de optimización, en ocasiones el robot prueba a describir una trayectoria aleatoria y si se da el caso de que encuentra un camino más favorable, cambia de recorrido. Esto también se ve en la simulación al usar uno de los robots, que, al intentar realizar un movimiento, el otro robot parece moverse hasta que se da cuenta de que cambiar sus posiciones articulares no va a influir en el movimiento buscado por el otro robot.

Se debe tener en cuenta que al realizar movimientos sucesivos las orientaciones del extremo del robot cambian, lo que provoca que en ocasiones la trayectoria realizada no sea la más sencilla puesto que, aunque el camino para alcanzar la traslación deseada sea simple, se necesita uno más complejo para alcanzar también la orientación buscada.

***EJEMPLO:* Movimiento de la torre blanca.**

En este caso se ha buscado el movimiento de la torre blanca. Este movimiento para realizar es horizontal, por lo que el robot toma la pieza y la posiciona en otra casilla siguiendo una trayectoria lineal mediante *MoveL*. En la *figura 81* se puede ver el rastreo del TCP al realizar el movimiento.

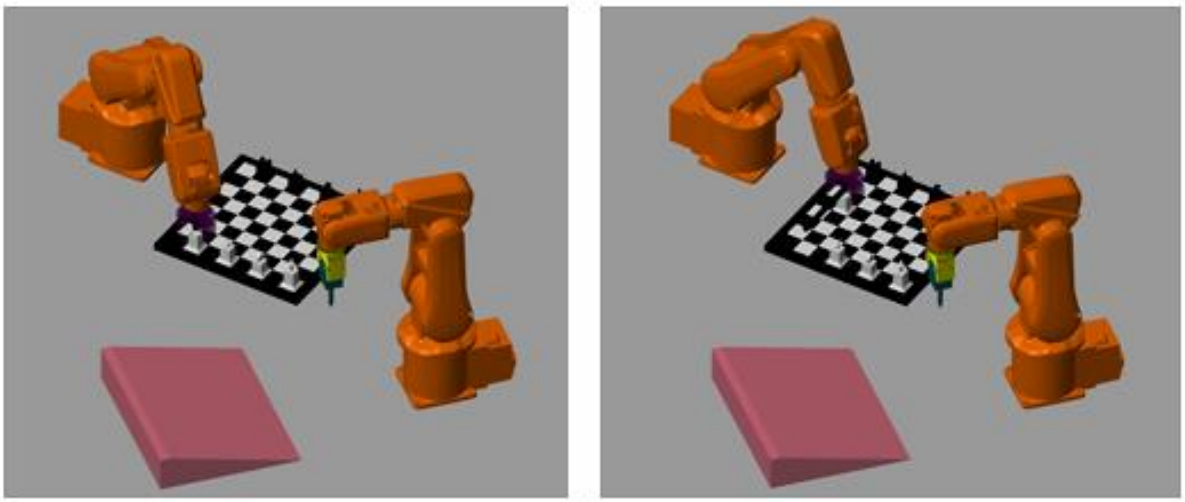


Figura 81: Trayectoria lineal de la torre

3.3.3.3 App Teacher Rastreador

La librería proporciona una aplicación creada a partir de las funciones del objeto *Kin* que facilita al usuario la búsqueda de puntos de interés para la cinemática inversa, así como la simulación de movimientos por ejes cartesianos.

Esta aplicación puede usarse de manera universal para cualquier robot. La interfaz se actualiza con los datos del objeto *Kin* cada vez que es llamada, cualquier cambio es aplicado en el robot *Multi-Body* y en las coordenadas del *Rastreador* para estar sincronizados, importando y exportando el objeto con los cambios realizados.

Figura 82: Visualización de la app Teacher Rastreador

La aplicación diseñada en el entorno app Designer se basa en 3 funciones internas que permiten realizar las siguientes acciones:

- Al pulsar el botón **Inicio,** la app lee la posición en la que se encuentra en ese momento el robot simulado, estableciendo ese punto como posición inicial y escribiendo sus valores [x,y,z] en la interfaz.
- Se pueden modificar las coordenadas cartesianas directamente de manera numérica o, a partir de incrementos y decrementos pulsando los símbolos **<** o **>**. Se puede introducir el valor a incrementar o decrementar, en la *figura 82*, este es de 50.
- Cada vez que estas coordenadas son modificadas, se muestra la posición del punto en el modelo simulado.

- Pulsando el botón **MoveJ** el robot se mueve hacia el punto mostrado siguiendo una trayectoria no lineal.
- Pulsando el botón **MoveL** el robot se mueve hacia el punto mostrado siguiendo una trayectoria lineal.
- Se puede **importar** el objeto *Kin*, *rob* en la *figura 82*, obteniendo sus datos desde el Workspace y actualizando con ellos los del Rastreador, o de manera contraria, se puede **exportar**, actualizando los datos del objeto almacenados en el Workspace con los valores modificados mediante la aplicación.
- De manera análoga, se pueden **importar** las coordenadas cartesianas, *punto* en la *figura 82*, desde el Workspace al Rastreador, **o exportar**, actualizando su valor en el Workspace.

3.4 DINÁMICA DEL ROBOT

La dinámica del robot se ocupa de establecer la relación entre las fuerzas aplicadas sobre el robot y el movimiento de este. Su modelo dinámico establece relaciones matemáticas entre las coordenadas del robot, ya sean articulares o las coordenadas cartesianas del TCP, sus derivadas (velocidad y aceleración), las fuerzas aplicadas y los parámetros del robot (masas, inercias…).

Se pueden considerar dos problemas dinámicos: el Problema Dinámico Inverso y el Problema Dinámico Directo.

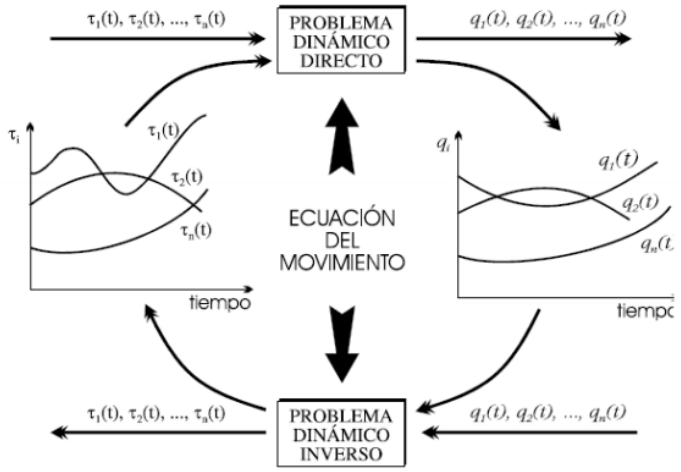


Figura 83: Dinámica de robots

El modelo dinámico directo fue el primero en ser abordado. Expresa la evolución en el tiempo de las coordenadas articulares del robot en función de las fuerzas y pares implicadas. Por el contrario, el modelo dinámico inverso expresa las fuerzas y pares que intervienen según las coordenadas articulares y sus derivadas. El uso principal de la dinámica directa es la simulación, mientras que el de la inversa se centra en el control.

La complejidad del modelo dinámico de un robot manipulador crece con el número de grados de libertad de este, no siendo siempre posible su obtención en forma cerrada. Existen relaciones no lineales e interrelación entre movimientos de las articulaciones, siendo frecuente la realización de simplificaciones. Por ello, en ocasiones se debe recurrir al uso de procedimientos iterativos para ser resuelto.

La obtención del modelo dinámico de un robot se basa en el balance de fuerzas o torques basado en la segunda ley de Newton, formulación de Newton-Euler, o en el balance energético, Formulación de Lagrange.

La formulación de Euler-Lagrange presenta un modelo más simple compuesto por ecuaciones diferenciales no lineales de 2º orden, donde aparecen los pares y fuerzas que intervienen en el movimiento. La formulación de Lagrange adopta la siguiente configuración:

$$\tau = D(q)\ddot{q} + H(q,\dot{q})q + g(q) \tag{3.10}$$

Donde *D* es la matriz de masa o inercia, *H* representa las fuerzas centrífugas y de Coriolis y *g* es el vector que incluye los términos gravitatorios.

Las ecuaciones de Lagrange-Euler son las siguientes:

$$\frac{d}{dt}\left(\frac{dL}{d\dot{q}_i}\right) - \frac{dL}{dq_i} = \tau_i \tag{3.11}$$

Siendo la función de Lagrange:

$$L(q,\dot{q}) = T(q,\dot{q}) - U(q) \tag{3.12}$$

Donde T representa la energía cinética y P la energía potencial del sistema.

Al modelar dinámicamente un robot se busca encontrar el modelo que permite el control y simulación del robot.

3.4.1 CONTROL DINÁMICO

Dentro del entorno de la robótica, el fin de realizar una determinada tarea requiere de la ejecución de un movimiento concreto por parte del robot. La ejecución correcta de dicho movimiento es responsabilidad del sistema de control, puesto que se encarga de proporcionar a los actuadores del robot las órdenes necesarias para conseguir el movimiento deseado.

El control dinámico procura que las trayectorias realmente seguidas por el robot $q(t)$, sean lo más parecidas posibles a las propuestas por el control cinemático $q_d(t)$. Para ello, se utiliza el modelo dinámico del robot y las herramientas de análisis y diseño aportadas por la teoría del servocontrol: representación interna, representación en el espacio de estado, estabilidad, control PID, control adaptativo, etc…

3.4.1.1 Acción de control

El regulador PID es uno de los más extendidos en la industria debido a su robustez y la diversidad de sus aplicaciones. Este presenta la estructura de la *figura 84.*

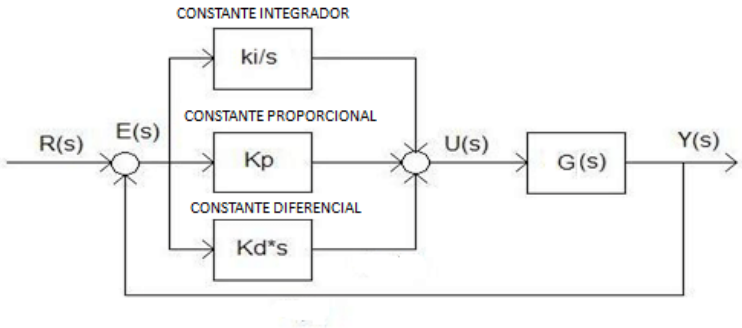


Figura 84: Estructura PID

Se pueden observar tres acciones: Proporcional (P), Integral (I) y Derivativa (D). Estas tres acciones pueden dar lugar a distintos tipos de reguladores: P, PI, PD y PID.

- **Acción proporcional (P):** Es el regulador más simple, cuya señal de salida es proporcional a la entrada, siendo su función de transferencia.

$$u(t) = k_P e(t) \qquad (3.13)$$

k_P es la constante de proporcionalidad ajustable que determinará el grado de amplificación del elemento de control.

- **Acción derivativa (D):** Su señal de salida es proporcional a la derivada de la señal de error:

$$u(t) = \frac{de(t)}{dt} k_d(s) \; = k_d s \qquad (3.14)$$

- **Acción Integral (I):** Su señal de salida es proporcional a la integral del error acumulado:

$$u(t) = k_i \int_0^t e(t)dt = \frac{k_i}{s} \qquad (3.15)$$

- **Acción proporcional-derivativo (PD):** En un control proporcional al aumentar k_P, los errores en régimen permanente disminuyen, sin embargo, también se modifica el régimen transitorio. Así, no se puede aumentar k_P de manera indefinida puesto que el sistema puede hacerse inestable. Con la acción derivativa, en régimen permanente la derivada se anula y el error es constante. De esta manera, la acción no influye en el régimen permanente y permite

mejorar únicamente el transitorio. Así, combinada con una acción proporcional se obtiene un regulador con una apreciable mejora de la velocidad de respuesta del sistema.

$$u(t) = k_P e(t) + \frac{de(t)}{dt} k_P T_d \qquad (3.16)$$

- **Acción proporcional-integral (PI):** Un control proporcional requiere de la existencia de error para tener acción de control distinta de 0. La acción integral se incorpora para mejorar el régimen permanente, un pequeño error positivo o negativo siempre dará una acción de control creciente o decreciente, aunque debe hacerse con cuidado para no modificar el transitorio. En realidad, no existen controladores que actúen únicamente con acción integral. Los controladores PI son ampliamente usados en sistemas químicos e hidráulicos.

- **Acción proporcional-integral-derivativa (PID):** Es un sistema de regulación que combina los tres controladores de acciones básicas con el fin de aprovechar las ventajas de cada uno de ellos, de manera que, si la señal de error varía lentamente en el tiempo, predomina la acción proporcional e integral y, mientras que, si la señal de error varía rápidamente, predomina la acción derivativa.

$$u(t) = k_P e(t) + \frac{de(t)}{dt} k_P T_d + \frac{k_p}{T_i} \int_0^t e(t)dt \qquad (3.17)$$

3.4.1.2 Implementación práctica

Se ha mostrado anteriormente el comportamiento de los modelos de robots al ser sometidos a una actuación sobre el movimiento (*apartado 3.1.1.3*).

En este apartado se va a realizar un análisis sobre el comportamiento dinámico, realizando un control sobre el modelo que permita encontrar los pares o fuerzas de los actuadores requeridos para generar una trayectoria deseada del robot. Para ello, se ha generado un modelo que permita realizar los distintos tipos de sistemas de control de realimentación lineal.

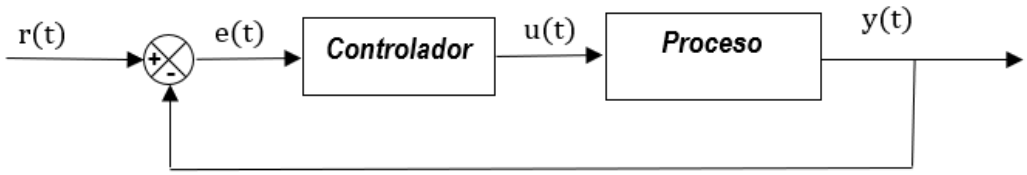


Figura 85: Sistema de control realimentado

En nuestro modelo la señal de entrada será el ángulo que indica la posición articular de cada eslabón. La señal de salida intentará seguir esta señal de referencia. La salida del controlador será el par motor aplicado a cada articulación. Se debe tener en cuenta que la señal de control no es el único efecto que actúa sobre el sistema, suelen estar presentes perturbaciones no controladas que tienen efecto sobre la salida.

Para este estudio, se ha creado un modelo que cuenta con esta realimentación en lazo cerrado. Se ha trabajado con el icono del robot *Staubli TX40*, al cual se le ha incorporado sus datos dinámicos: masas, centros de gravedad, momentos de inercia:

```
%masa
M= [10.8, 5.4, 4.05, 3.24, 2.16, 1.36];

%centro de gravedad
CdG= [0, 0, 0, 0, 0, 0;
      0,  0, 0, 0, 0.05, 0;
      0.15, 0.2, 0, 0.05, 0, 0.03];

% Ixx Iyy Izz Ixy Iyz Ixz
I=[0    ,    0.35,    0    ,   0,  0,  0;
   0.13 ,    0.524,   0.539,   0,  0,  0;
   0.066 ,   0.086,   0.0125,  0,  0,  0;
   0.0018,   0.0013,  0.0018,  0,  0,  0;
   3e-4,     4e-4,    3e-4,    0,  0,  0;
   1.5e-4,   1.5e-4,  4e-5,    0,  0,  0];
```

Figura 86: Parámetros dinámicos del robot Staubli TX40

Se han adaptado las articulaciones del robot para que la entrada sea el par de cada motor y la salida el ángulo generado por el par en el modelo obtenido por simulación. Esta salida será la realimentada a la entrada de referencia q_{ref}. El movimiento es calculado de manera automática.

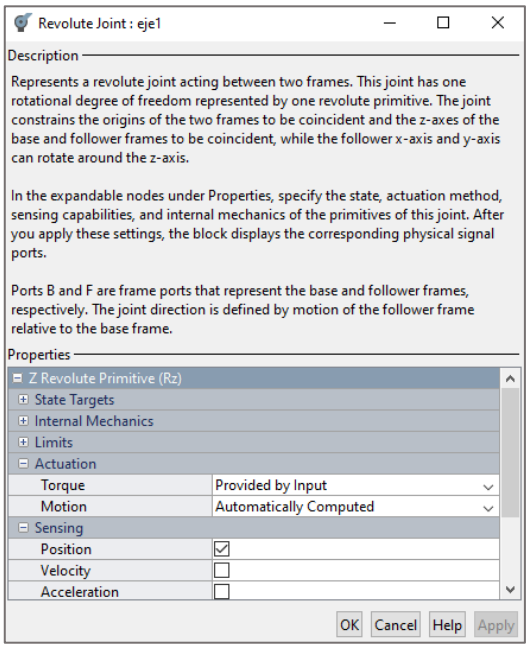


Figura 87: Bloque Joint con actuación dinámica

El modelo de la estación generada en SIMULINK es el siguiente:

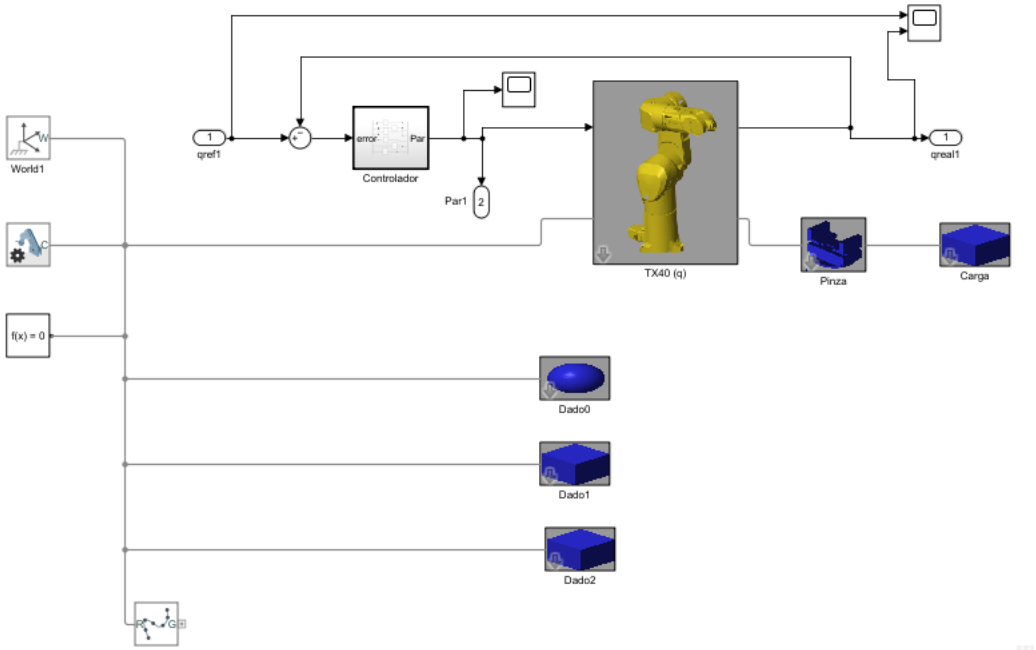


Figura 88: Esquema en SIMULINK de Estacion_Control

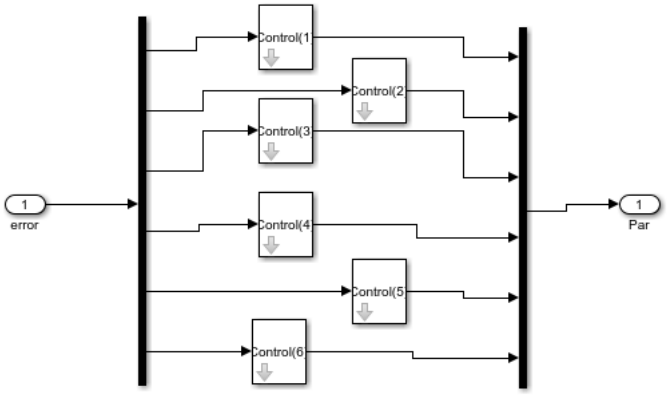


Figura 89: interior subsistema del bloque 'Controlador'

Cada uno de los bloques que presenta la estructura de la *figura 89* son bloques *LTI System*, uno por cada eje del robot a controlar. Este bloque permite importar cualquier tipo de modelo de sistema dinámico, pudiendo especificar su función de transferencia y parámetros.

Se modifica esta función de transferencia, determinada en el dominio de la frecuencia, en función del controlador que se quiera usar, permitiendo así incorporar distintos sistemas de control al modelo de la *figura 88*.

Para este estudio se han analizado tres supuestos: que el robot desee levantar una masa de 1 kg (*Dado0*), una masa de 15kg (*Dado1*) y otra de 40kg (*Dado2*). Se han incorporado en el modelo dos bloques *Scope* que permiten ver los pares motores aplicados en función del esfuerzo que necesite realizar el robot, así como comprobar el seguimiento de la señal de referencia por parte del ángulo de salida generado.
Puesto que se trata de un sistema mecánico, se analizarán estos tres supuestos en función de usar un controlador proporcional o un controlador proporcional-derivativo.

Función de transferencia usada en MATLAB del controlador proporcional:

$$Control(i) = k_P(\mathrm{i}) \tag{3.18}$$

Función de transferencia usada en MATLAB del controlador proporcional-derivativo:

$$Control(i) = k_P(\mathrm{i})\frac{T_d(\mathrm{i})\cdot(\mathrm{s}+1)}{T_d(\mathrm{i})/\mathrm{N}\cdot(\mathrm{s}+1)} \tag{3.19}$$

Debido a los parámetros dinámicos de este robot, se necesita una constante de proporcionalidad muy elevada, en torno a 10^5, para eliminar errores. Sin embargo, se ha trabajado con una constante del orden de 10^3 para que sean visibles las oscilaciones producidas al generar la trayectoria.
Se ha aplicado el objeto *Kin* para realizar los movimientos cinemáticos.

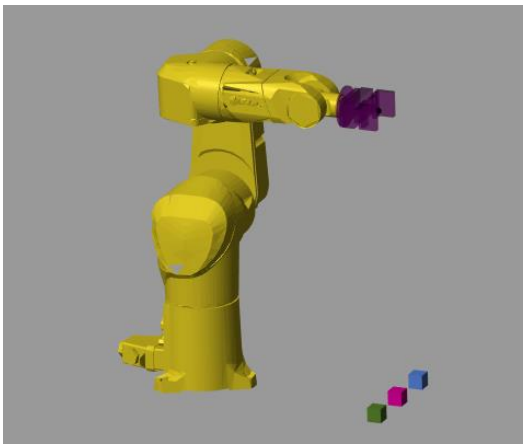


Figura 90: Posición inicial del robot en el modelo Estacion_Control

CONTROLADOR PROPORCIONAL (K)

- ***Primera trayectoria – masa de 1 kg***

Partiendo de una posición inicial, *figura 90,* se ha dirigido el robot hacia el dado cuya masa es 1 kg (dado de color verde). Una vez tomado este, se ha observado el esfuerzo que debe realizar el robot para moverlo hasta adquirir de nuevo su posición inicial.

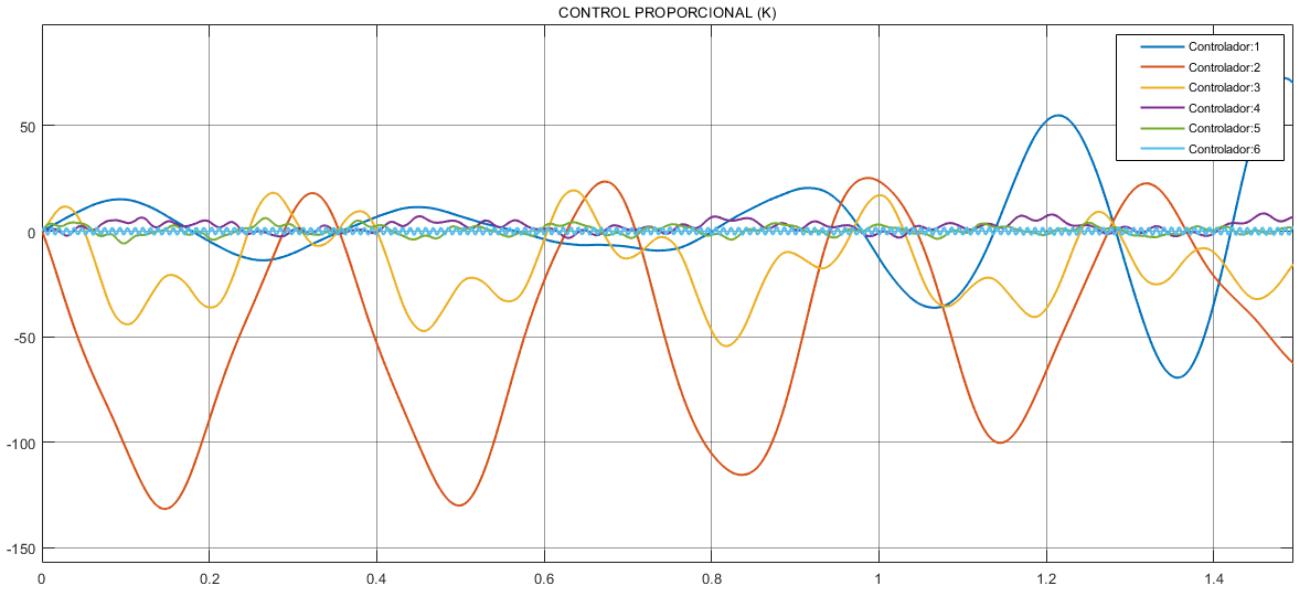


Figura 91: Pares aplicados en la primera trayectoria (K)

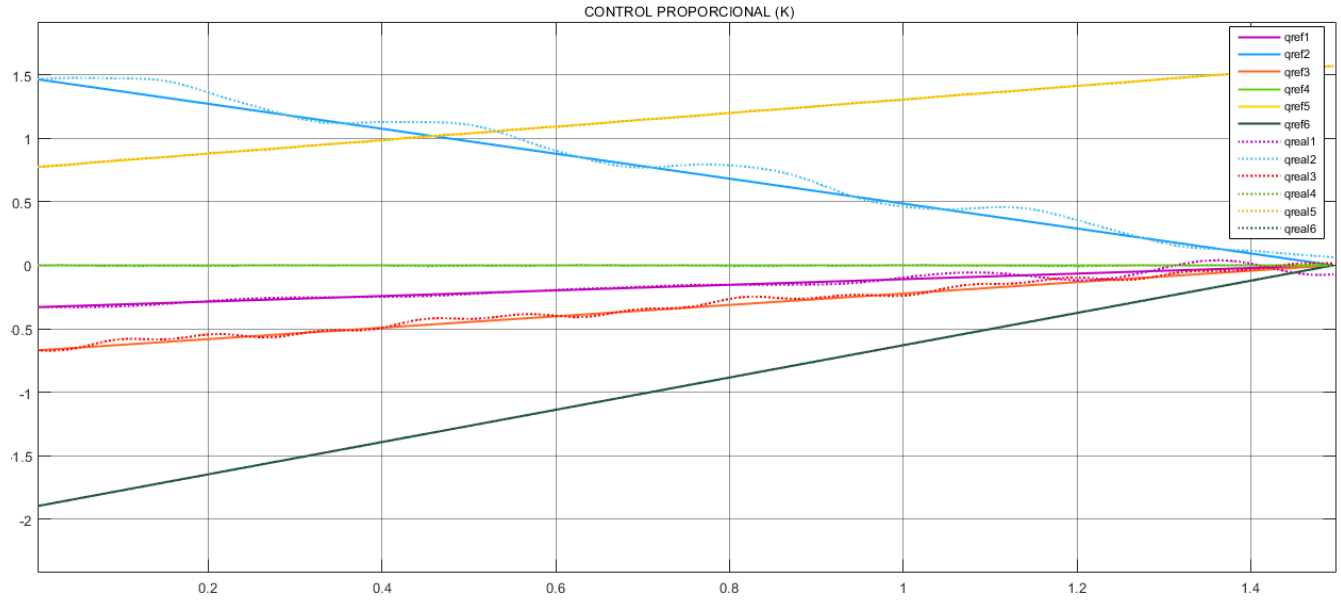


Figura 92: Seguimiento de la referencia en la primera trayectoria (K)

- ***Segunda trayectoria – masa de 15 kg***

De manera análoga al caso anterior, se ha medido la fuerza aplicada para mover el dado de masa 15 kg (dado de color rosa) desde su posición actual hasta la posición de partida del robot.

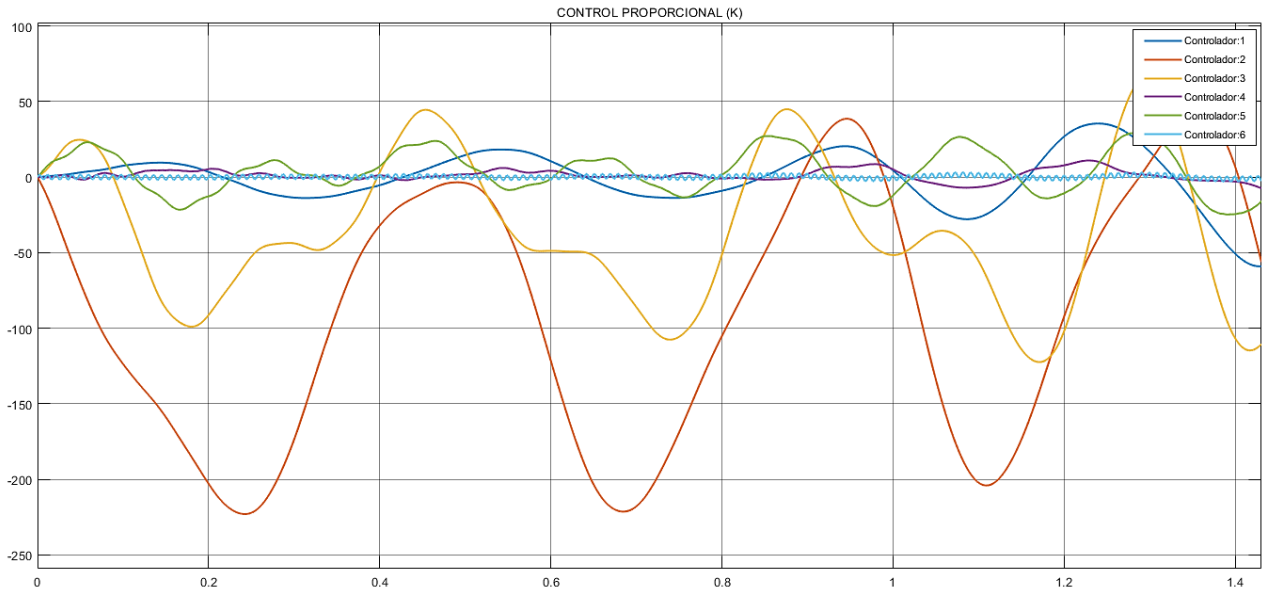


Figura 93: Pares motores aplicados en la segunda trayectoria (K)

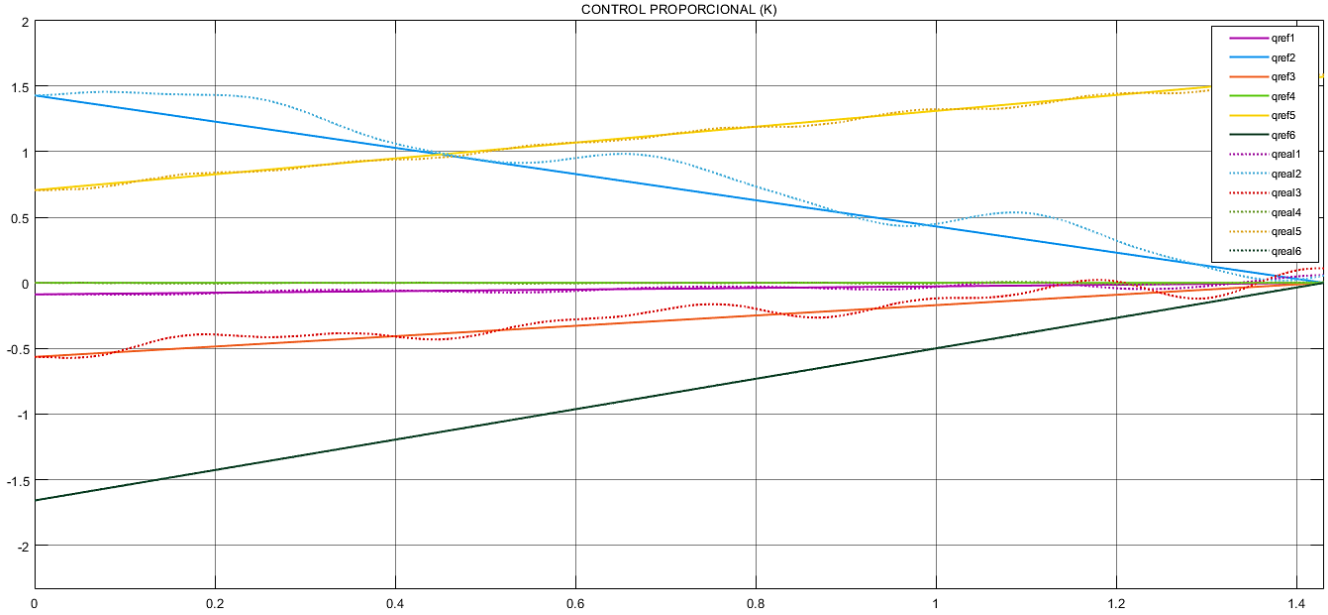


Figura 94: Seguimiento de la referencia en la segunda trayectoria (K)

- ***Tercera trayectoria – masa de 40 kg***

En este caso el dado a mover es aquel el cual presenta una masa de 40 kg (dado de color azul).

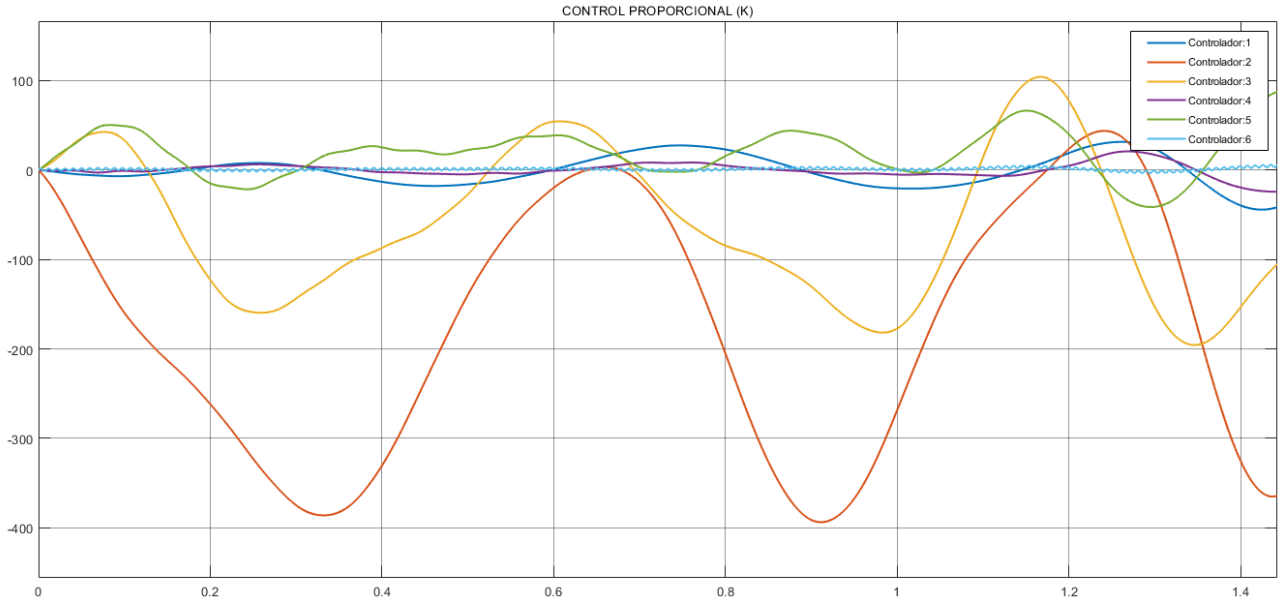


Figura 95: Pares motores aplicados en la tercera trayectoria (K)

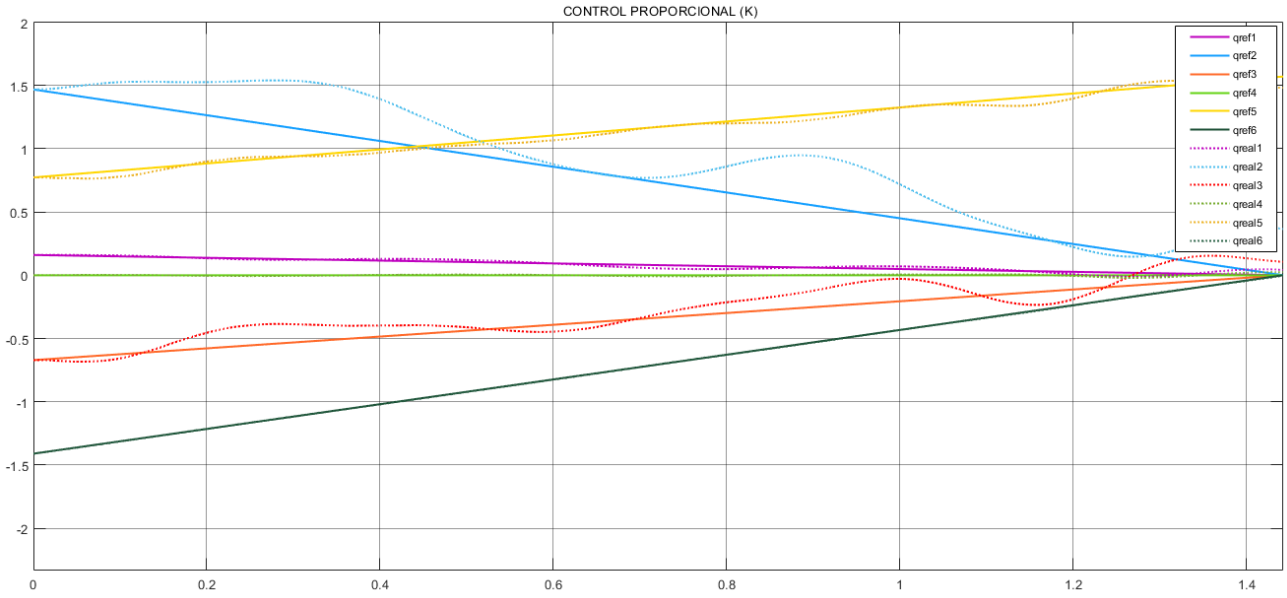


Figura 96: Seguimiento de la referencia en la tercera trayectoria (K)

Se ha podido observar que el control realizado sobre el sistema ha conseguido adaptar el par motor a cada situación; aumenta a medida que se requiere de mayor esfuerzo para levantar una carga de mayor masa, pasando de un valor máximo aproximadamente de -130 N sobre el eje que realiza mayor esfuerzo, eje 2, en la primera trayectoria, a un valor cercano a -400 N sobre este mismo eje en la última trayectoria.

El seguimiento del ángulo de salida respecto a la referencia es evidentemente peor al aumentar la masa puesto que, al aumentar el esfuerzo a realizar, aumentan las oscilaciones del robot generadas al seguir su trayectoria, así como el tiempo que tarda en realizarla.

CONTROLADOR PROPORCIONAL-DERIVATIVO (PD)

- ***Primera trayectoria – masa de 1 kg***

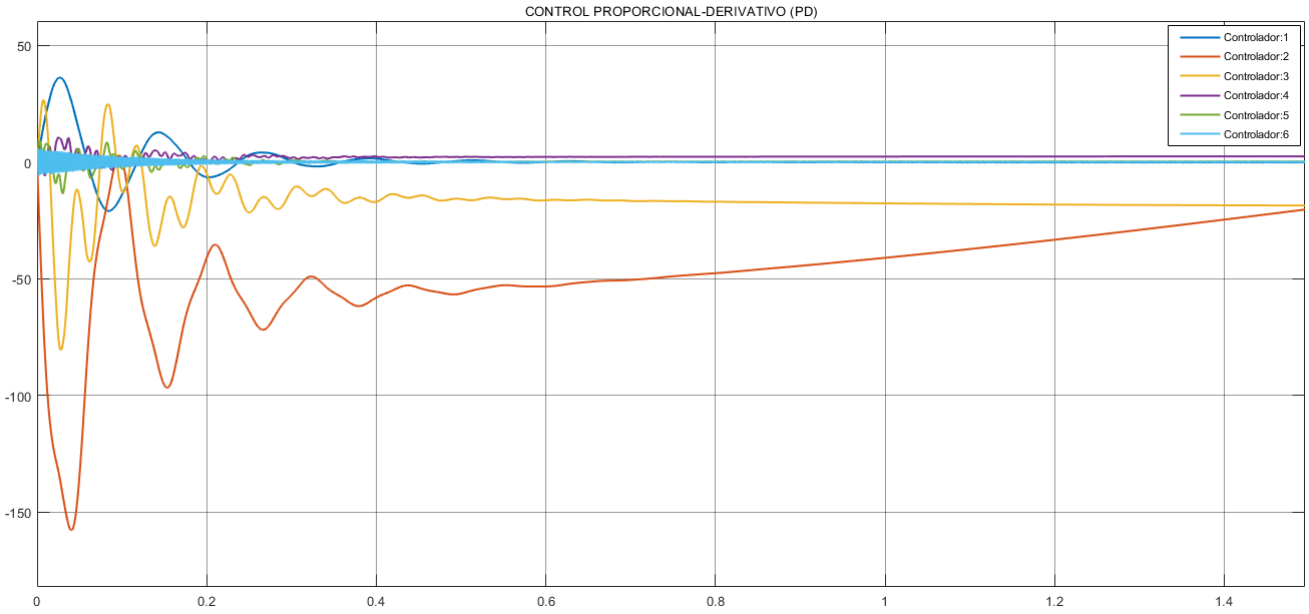


Figura 97: Pares aplicados en la primera trayectoria (PD)

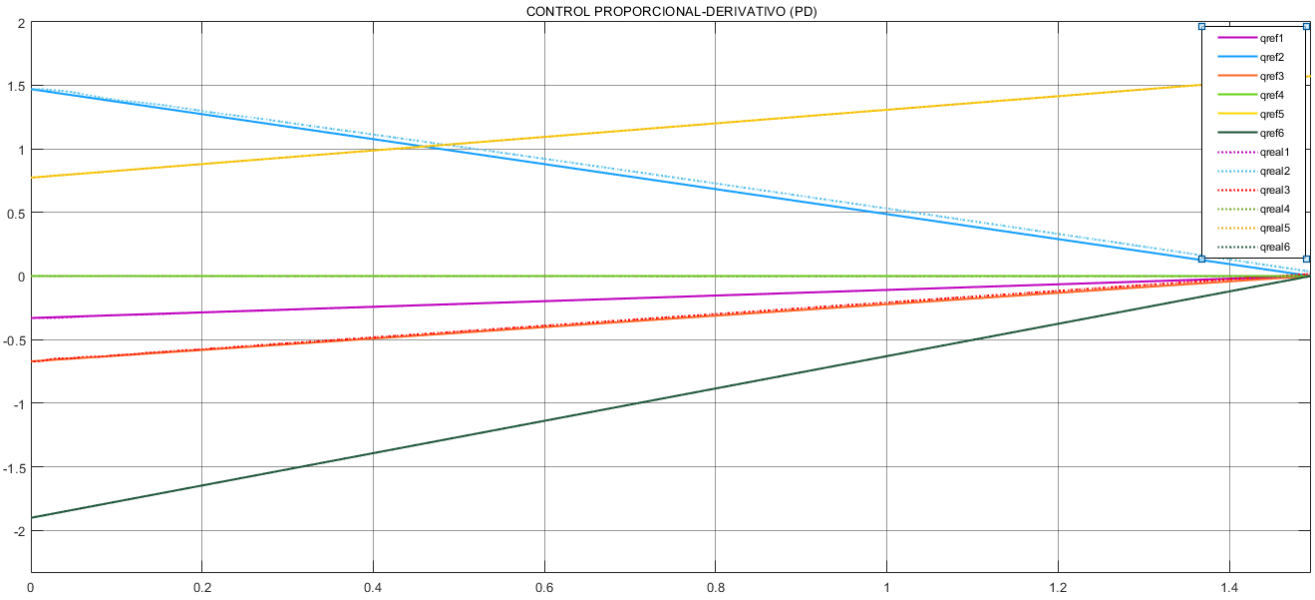


Figura 98: Seguimiento de la referencia en la primera trayectoria (PD)

- ***Segunda trayectoria – masa de 15 kg***

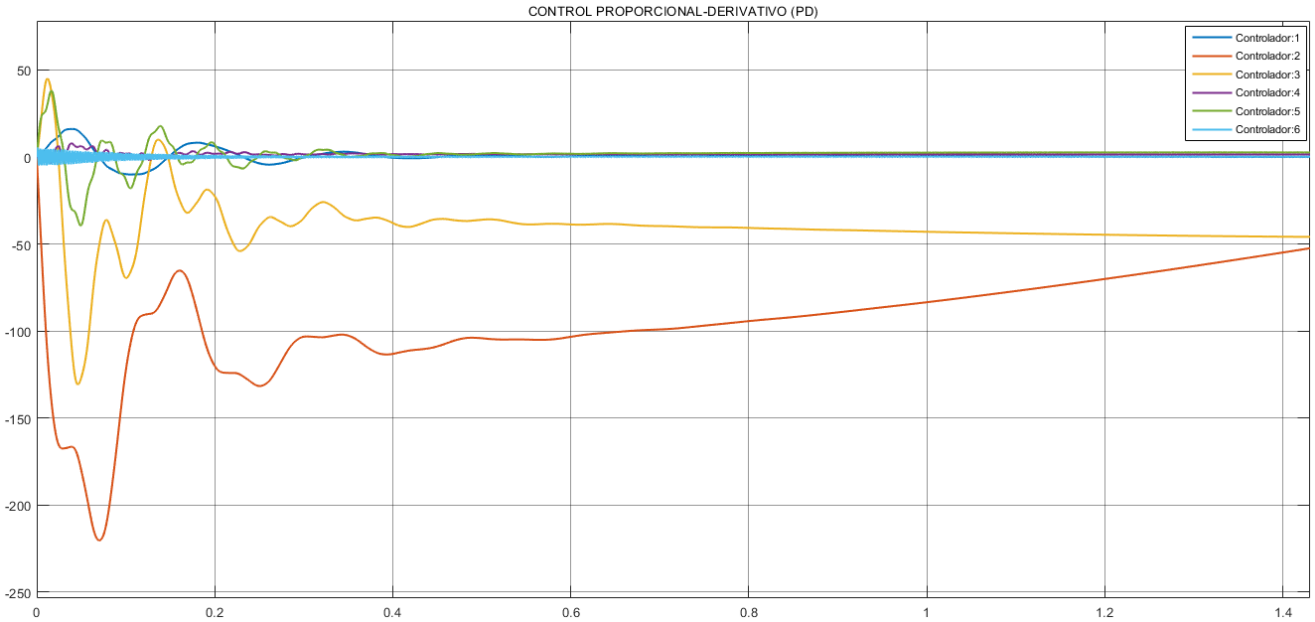


Figura 99: Pares aplicados en la segunda trayectoria (PD)

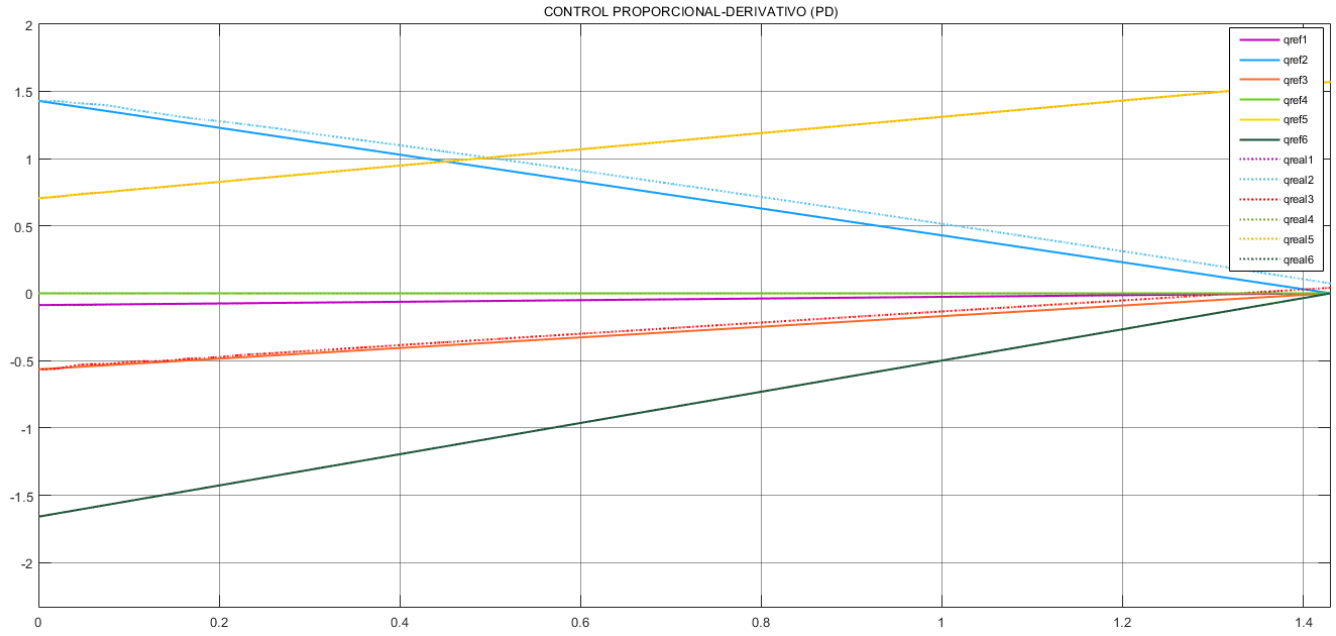


Figura 100: Seguimiento de la referencia en la segunda trayectoria (PD)

➢ ***Tercera trayectoria – masa de 40 kg***

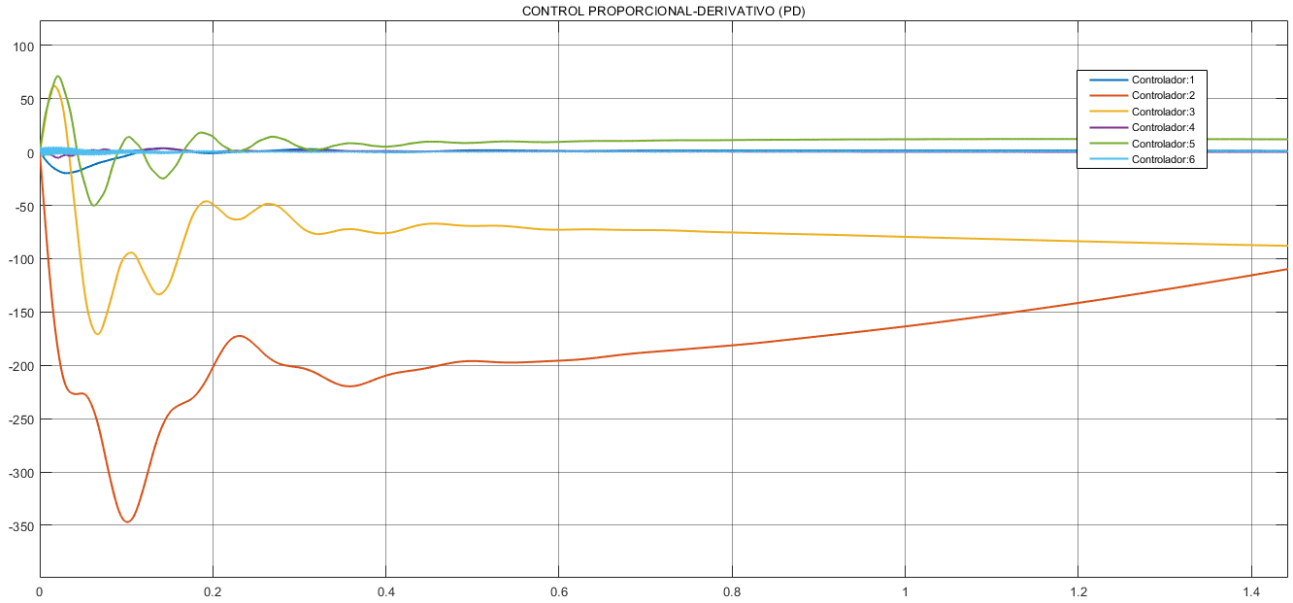


Figura 102: Pares motores aplicados en la tercera trayectoria (PD)

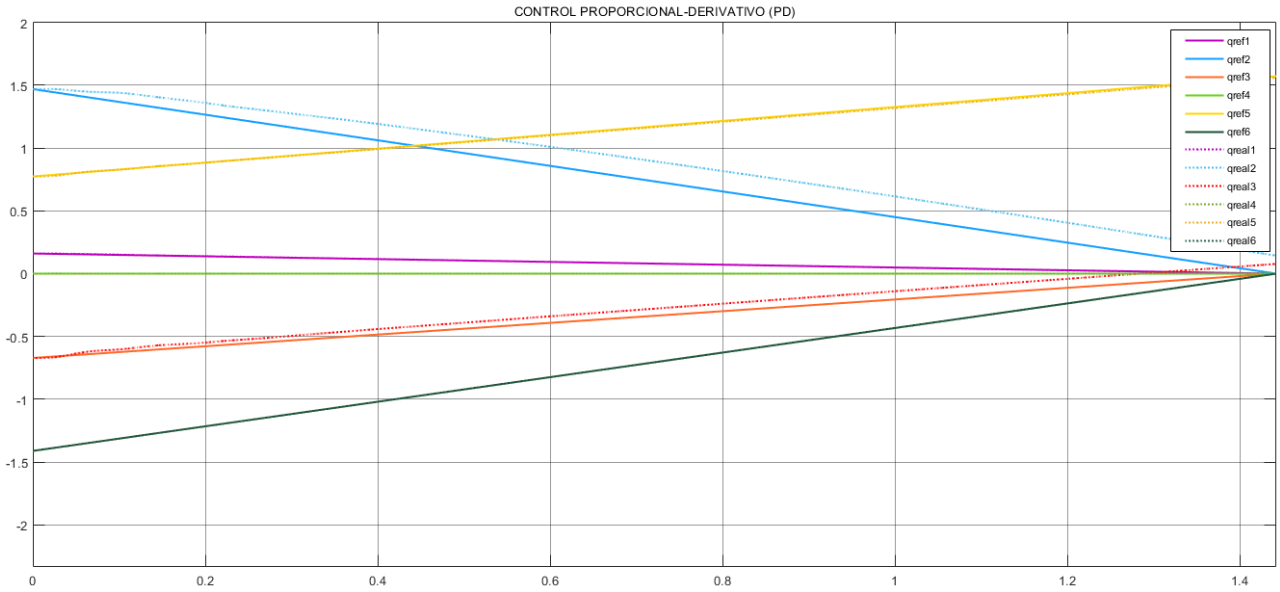


Figura 101: Seguimiento de la referencia en la tercera trayectoria (PD)

Se ha podido observar que con un controlador PD el comportamiento mejora significativamente. El esfuerzo aplicado en el eje 2 es superior que el resto, como en el caso anterior, sin embargo, en este caso, este presenta un pico al principio y luego consigue estabilizarse un poco. Apenas se producen oscilaciones, lo que provoca que el esfuerzo no sufra tantos altibajos y el seguimiento de la referencia, aunque es algo peor a medida que aumenta el esfuerzo a mover, sigue siendo muy mejorable en comparación al controlador proporcional.

3.5 COMUNICACIÓN OPC

La arquitectura TCP/IP nos proporciona una estructura y una serie de normas de funcionamiento para poder interconectar sistemas. Los sockets son un mecanismo de comunicación entre procesos que permiten la comunicación bidireccional tanto entre procesos que se ejecutan en una misma máquina como entre procesos lanzados en diferentes máquinas. Es muy común la aplicación de un socket de comunicación utilizando el protocolo TCP/IP.

Otra posibilidad de conectividad remota a través de internet que ofrece los robots ABB es la comunicación OPC. OPC es el estándar de interoperabilidad para el intercambio seguro y fiable de datos en el espacio de la automatización industrial y en otras industrias. Es independiente de la plataforma y garantiza un flujo continuo de información entre los dispositivos de múltiples proveedores. Además, a diferencia de comunicación por Socket, no es preciso que la lectura y la escritura estén sincronizadas. Una arquitectura OPC incluye uno o varios Clientes OPC y Servidores OPC comunicándose entre sí.

Para comprobar el software propuesto, debido a su mayor sencillez de implementación y a que ABB proporciona un servidor OPC, en este apartado se realiza la comunicación OPC, tipo *cliente-servidor*, mediante un servidor OPC ABB de RobotStudio y un cliente OPC de MATLAB, utilizando las funciones de la OPC Toolbox de MATLAB [9].

El sistema proporciona una serie de opciones de cara a la lectura o escritura de datos de forma remota. En este caso, se busca recoger las variables articulares generadas por la posición del robot virtual *irb120* de ABB, cuyo modelo de estación se ha generado en SIMULINK, y comprobar la interacción con el robot real pasando estas a RobotStudio y analizando que ambos robots estén coordinados y realicen las mismas acciones.

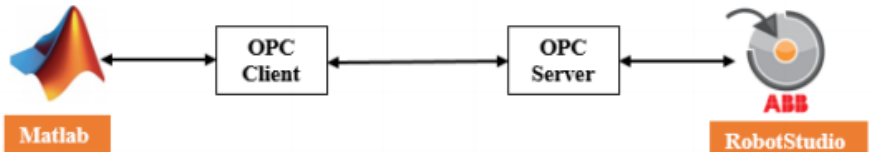


Figura 103: Estructura comunicación OPC

Para establecer la comunicación se usa la aplicación ABB IRC5 OPC. Esta permite configurar a qué controladores se conectará el servidor. Por ello, el primer paso es la ejecución de esta aplicación, que permite detectar los dispositivos de ABB conectados al PC, ya sean reales o virtuales, y establecer la comunicación.

Mediante un escaneo, la aplicación encuentra el nombre del controlador que usaremos en RobotStudio, permitiendo obtener el nombre del Alias con el que poder reconocer la conexión.

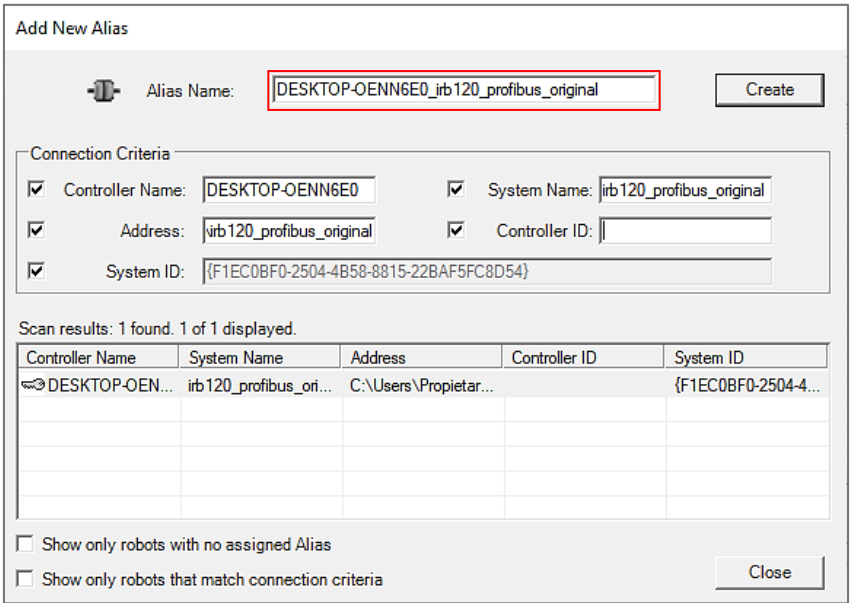


Figura 104: Creación del Alias

La configuración del servidor queda lista para cuando se quiera activar la comunicación para realizar la simulación.

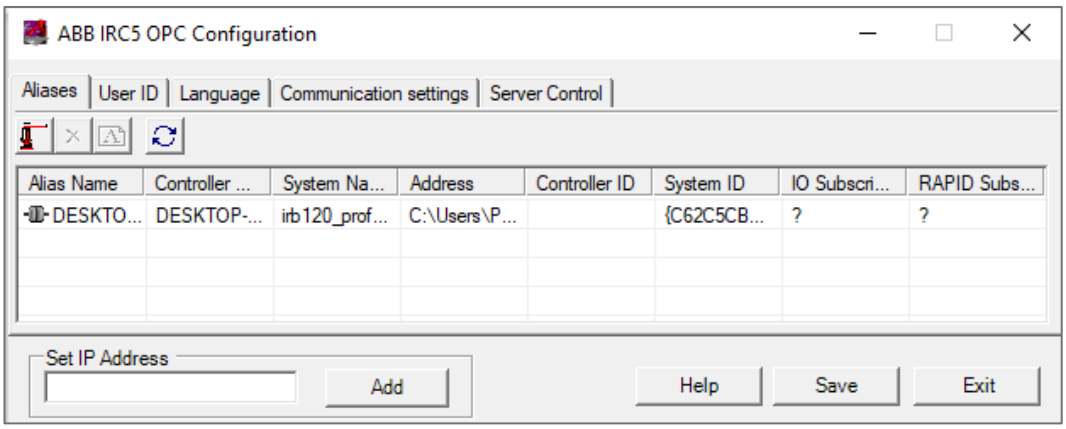


Figura 105: Configuración ABB IRC5 OPC

El siguiente paso es la creación del programa en RAPID. El servidor solo puede leer y escribir variables en RAPID de tipo Persistente (PERS), por lo que las variables que representan las configuraciones articulares han de ser configuradas como tal, véase *figura 106*.

Aunque no se ha llevado a cabo en este proyecto, también cabe la posibilidad de introducir señales en la comunicación, a través de interrupciones establecidas en el programa en RAPID activadas por el cliente desde MATLAB.

```
MODULE Com_q

    LOCAL PERS num Joint{6}:=[30,0,0,0,90,0];
    LOCAL var jointtarget JointRobot;

    PROC main_Com_OPC_q()
        TPWrite "Inicio Comunicación";
        WHILE TRUE DO
            JointRobot.robax.rax_1:=Joint{1};
            JointRobot.robax.rax_2:=Joint{2};
            JointRobot.robax.rax_3:=Joint{3};
            JointRobot.robax.rax_4:=Joint{4};
            JointRobot.robax.rax_5:=Joint{5};
            JointRobot.robax.rax_6:=Joint{6};
            MoveAbsJ JointRobot,v300\T:=0.1,z0,tool0;
        ENDWHILE
    ENDPROC

ENDMODULE
```

Figura 106: Módulo de comunicación en RAPID

Cuando se ejecute el programa, el robot quedará a la espera de recibir los valores de las articulaciones que provocarán el cambio de posición. Esta posición es cambiada desde SIMULINK cada 0.1 s.

Finalmente, antes de realizar la simulación, se configura la estación y el programa en SIMULINK.

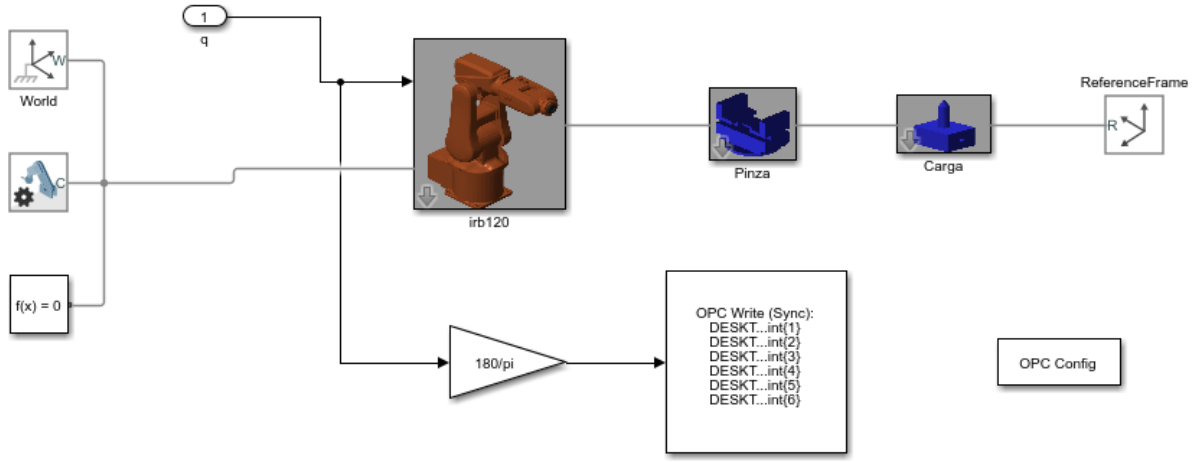


Figura 107: Estación de comunicación en SIMULINK

La estación está formada por el brazo robótico *irb120* al que se le ha acoplado una pinza como herramienta y un lápiz como carga. Mediante la interfaz de OPC Toolbox se implementan las funciones que van a permitir la comunicación; se crea un objeto OPC Data Access (DA) que representa la conexión entre MATLAB y el servidor OPC. Puesto que se pretende llevar a cabo la escritura de las variables articulares, se ha utilizado el Bloque *OPC write* y el bloque *OPC configuration.* Estos se hallan dentro de la biblioteca de Simulink, en la pestaña OPC Toolbox.

- bloque ***OPC configuration*****: s**irve para configurar Matlab como cliente de OPC, pudiendo conectarse al servidor y desconectarse. Para realizar la configuración, se abre la ventana de parámetros del bloque (*figura 108)* en la cual se pueden modificar variables como la velocidad del tiempo real.

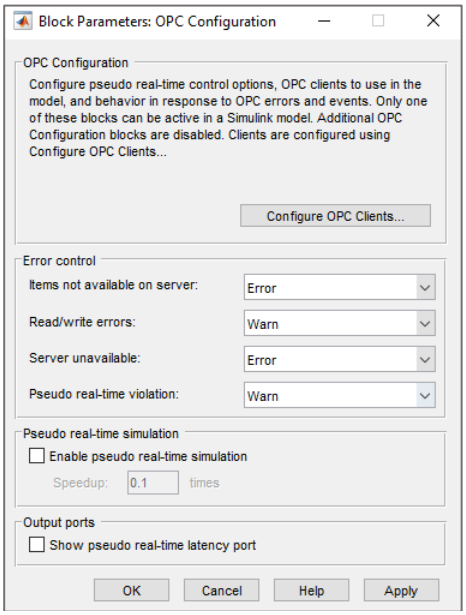


Figura 108: Parámetros del bloque OPC Configuration de SIMULINK

Pulsando el botón *Configure OPC Clients*… se abrirá la ventana que permitirá configurar las propiedades del servidor. Puesto que la conexión al servidor tiene lugar en la misma máquina que el cliente, aparece "localhost" como *hostname*. Para definir la conexión se determina la ID del servidor a usar:

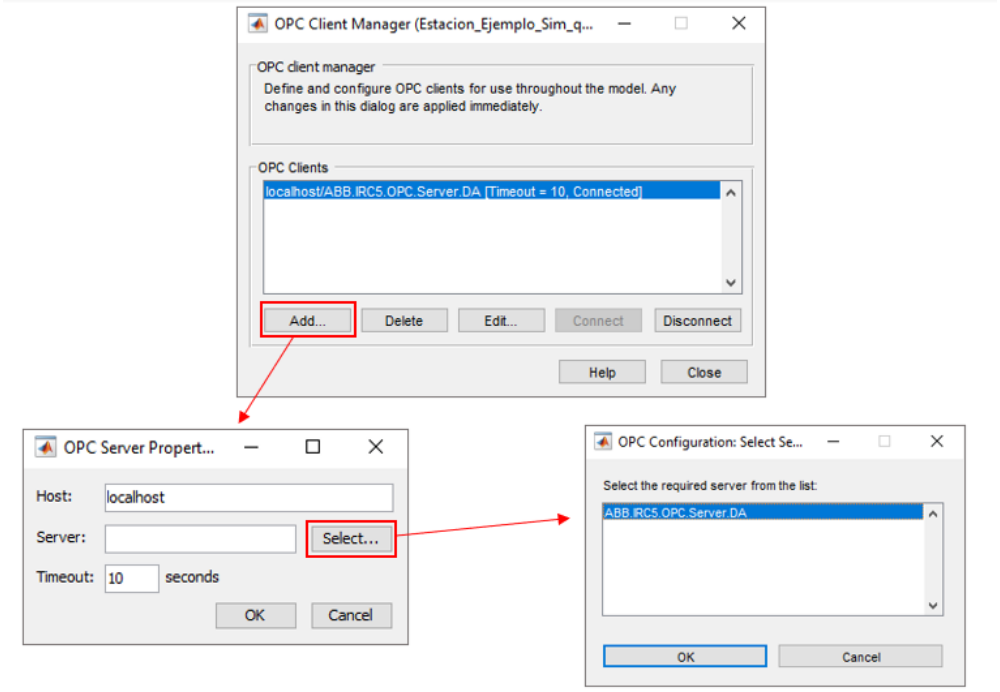


Figura 109: Configuración del bloque OPC Configuration de SIMULINK

Una vez realizada dicha configuración, en la ventana *OPC Clients Manager* se puede ver el servidor elegido y si está conectado o no.

- bloque ***OPC Write*****:** permite escribir los datos en el OPC servidor. Se ha seleccionado el tipo de escritura síncrona y se han añadido las variables a escribir desde el programa cliente, es decir, las variables articulares definidas como variables persistentes en el programa RAPID (*figura 111*). La configuración de la conexión se realiza igual que en el bloque anterior, a través de *Configure OPC Clients*.

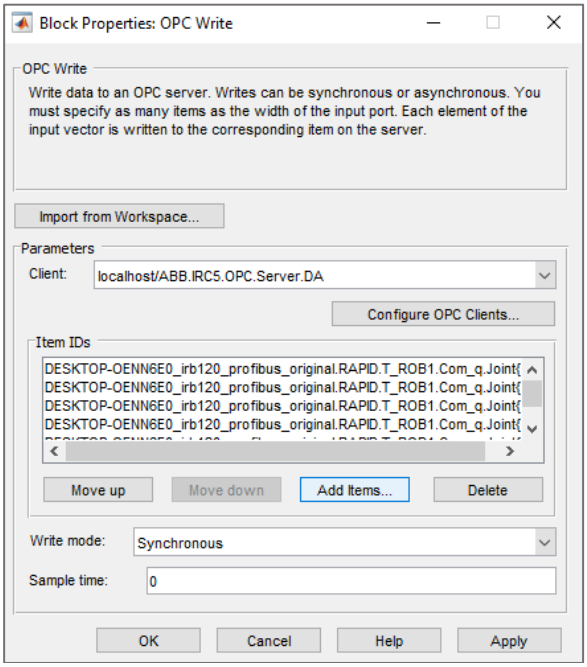


Figura 110: Parámetros del bloque OPC Write de SIMULINK

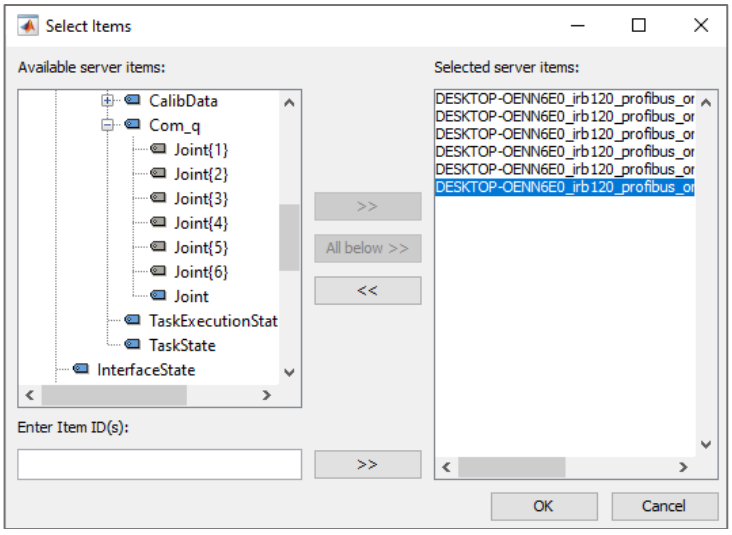


Figura 111: Selección de variables

Tras realizar estas configuraciones, se crea el programa que va a permitir la simulación. En el ejemplo realizado, se busca el movimiento de los ejes 2,3,5 y 6:

-movimiento de 0 a 40 grados alrededor del eje 2.
-movimiento de 0 a -35 grados alrededor del eje 3.
-movimiento de 0 a 40 grados alrededor del eje 5.
-movimiento de 0 a 40 grados alrededor del eje 6.

```
nPts= 1e3;
tend= 6;
t= linspace(0,tend,nPts)';
%movimiento de ejes
q= zeros(nPts,6);
q(:,2)= linspace(0,40,nPts)'*deg;
q(:,3)= linspace(0,-35,nPts)'*deg;
q(:,5)= linspace(0,40,nPts)'*deg;
q(:,6)= linspace(0,40,nPts)'*deg;
opt= []; % parámetros de simulación por defecto
sim(Estacion, t, [], [t,q]);
```

Finalmente, se procede a la simulación, para ello:

- Se activa el OPC server de ABB previamente configurado.
- Se ejecuta el programa RAPID del robot ABB; el robot queda en estado de espera.
- Se definen las variables asociadas a los iconos de la estación de SIMULINK y se ejecuta el programa que activa la simulación.

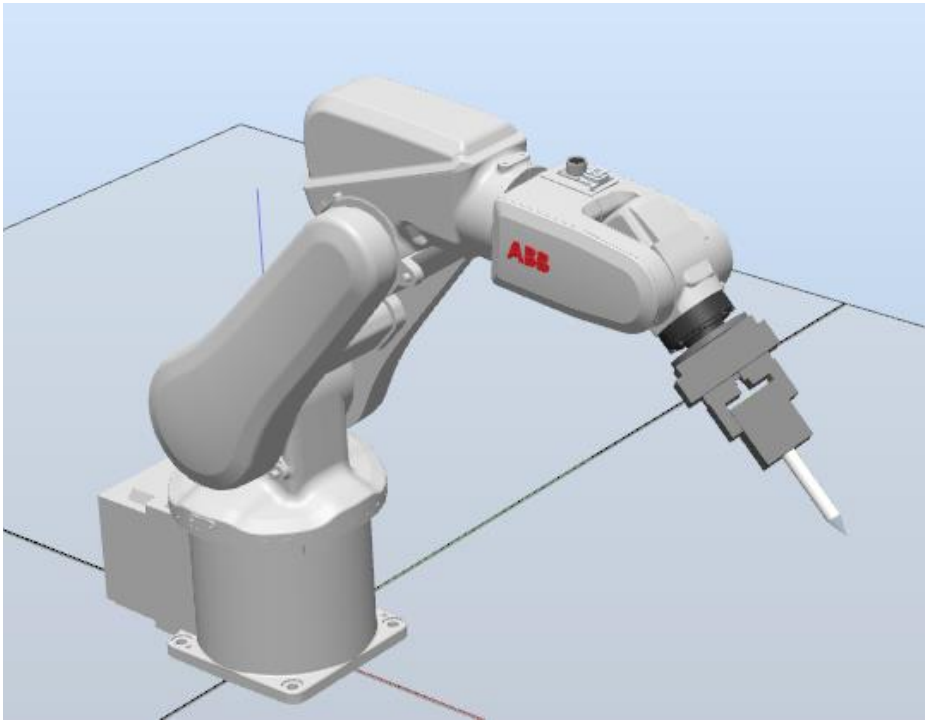


Figura 112: Posición final del brazo robótico irb120 em RobotStudio

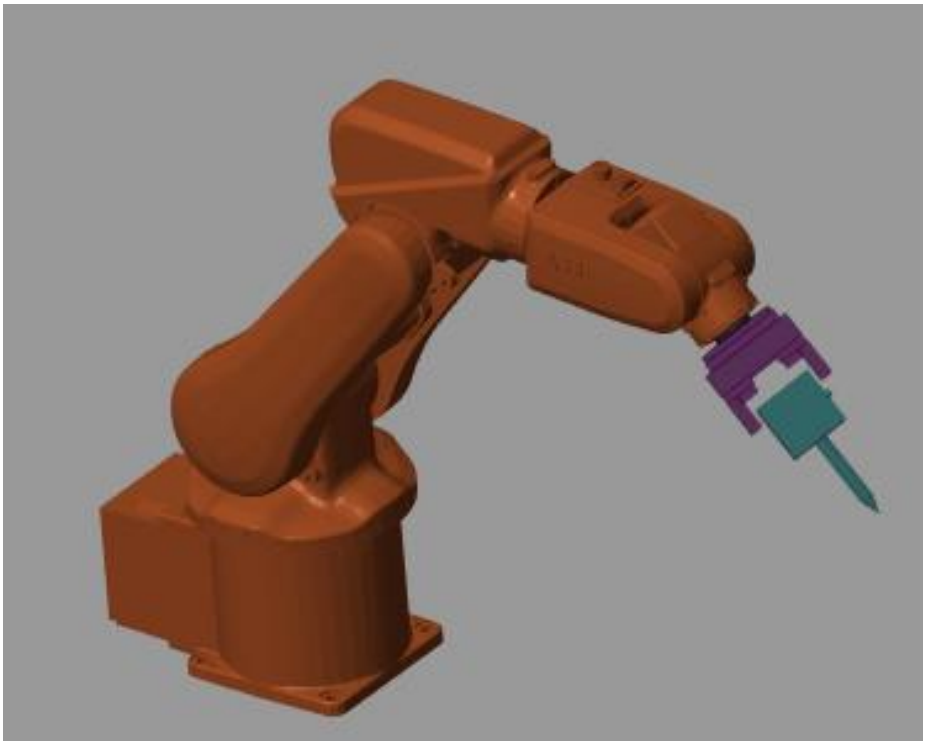


Figura 113: Posición final del brazo robótico irb120 en SIMULINK

4. CONCLUSIONES Y LÍNEAS FUTURAS DE TRABAJO

4.1 CONCLUSIONES

Una vez desarrollado el trabajo, concluimos que:

- *Robotic Toolbox* de MATLAB permite obtener la estructura de representación *RigidBody* de robots manipuladores, siendo una herramienta válida para el cálculo de los puntos de interés de la estación. Simscape Multibody proporciona un potente entorno de simulación. Se puede combinar ambas librerías y definir un modelo robótico en el entorno *Mutli-body* de SIMULINK, obteniendo su estructura equivalente *RigidBody* en MATLAB.

- Se ha conseguido disponer de una librería de iconos que pone al alcance del alumno la posibilidad de diseñar su propia estación mediante la incorporación de bloques que representan los elementos que toda estación robótica contiene. Permite incluso la posibilidad de que, si fuese requerido un elemento específico del cual no se dispone su icono en la librería, el alumno pueda obtener este a partir de sus propias modificaciones.

- La aplicación es válida como recurso didáctico, permitiendo que el alumno ponga en práctica los conocimientos teóricos adquiridos sobre robótica:

    - Permite analizar la localización espacial del sólido rígido, viéndose las definición y transformaciones de ejes que tienen lugar, así como su representación mediante el uso de la clase *Hmat().*

    - Permite abordar la resolución de la cinemática directa e inversa, obteniéndose los puntos de interés de dos maneras posibles: a través de la estructura *RigidBody* o a través de la app Teacher *Rastreador*.

    - Proporciona un algoritmo de optimización que permite que el robot manipulador realice trayectorias no lineales siguiendo el camino más sencillo.

    - Permite relacionar el movimiento de los robots y las fuerzas implicadas en el mismo, en función de las características dinámicas (masa e inerciales) de este y del control dinámico aplicado al modelo, resultando el controlador PD como el más favorable.

    - Proporciona un entorno de simulación completo y fiable en el que se representan los elementos de la estación en 3D y las interacciones que se dan entre ellos.

- Se ha podido establecer la comunicación OPC desde el robot de ABB disponible en la librería de iconos, por lo que el alumno podrá aplicar esta herramienta tanto en el robot virtual representado con RobotStudio como el real, ya que el servidor OPC es válido para ambos casos.

4.1 POSIBLES LÍNEAS DE TRABAJO FUTURO

Se proponen las siguientes líneas futuras de trabajo para aquellos interesados en el proyecto:

- Ampliar la librería de iconos mediante la incorporación de bloques de nuevos robots industriales, objetos de trabajo, cargas o herramientas.

- Mejorar la asignación del fichero *.stl a los bloques de iconos, que no requiera del uso de una biblioteca auxiliar, junto de la función *Pieza()*, cuando se quiera cambiar de representación.

- Intentar que, en la simulación de robots, al seguir sucesivas trayectorias lineales y circulares, la transición entre ellas sea más rápida y que no aparezcan interpolaciones no deseadas.

- Modificar el diseño de la interfaz gráfica de *App Designer* e incluir nuevas funciones.

- Profundizar en el estudio dinámico del robot a través de *Simscape Multibody*, añadiendo o sustituyendo componentes: sensores, actuadores, señales de entrada.

- Realizar el control dinámico sobre otras variables del sistema y comprobar su comportamiento al utilizar otro tipo de controladores.

- Estudiar la comunicación del software con ROS, con vistas a su aplicación en pequeños robots colaborativos, los cuales son pensados incorporar en la escuela de Ingenierías Industriales de la Universidad de Valladolid.

En definitiva, se trata de una nueva aplicación a usar en el ámbito docente que abre la posibilidad de plantear multitud de trabajos futuros relacionados con la mejora de esta y la ampliación de sus posibles funcionalidades.

5. BIBLIOGRAFÍA

[1] *MATLAB* (versión 2021a). Software Mathworks. https://es.mathworks.com/products/matlab.html (último acceso, junio 2021)

[2] *SIMULINK* (versión 2021a). Software Mathworks. (https://es.mathworks.com/products/simulink.html (último acceso, junio 2021)

[3] Corke, P. (2014). *Robotics Toolbox*. http://petercorke.com/Robotics Toolbox.htm (último acceso, junio 2021)

[4] Corke, P. (2013). "*Robotics, Vision & Control, Fundamental Algorithms in MATLAB*".

[5] Gil Aparecio, A. (2014). ARTE (A Robotic Toolbox for Education) Universidad Miguel Hernández (Elche, España), http://arvc.umh.es/arte/index_en.html (último acceso, junio 2021)

[6] *Robotic System Toolbox* (versión 2021a). Software Mathworks. https://es.mathworks.com/products/robotics.html (último acceso, junio 2021)

[7] *Simscape Multibody* (versión 2021a). Software Mathworks. https://es.mathworks.com/products/simscape-multibody.html (último acceso, junio 2021)

[8] *App Designer* (versión 2021a). Software Mathworks. https://es.mathworks.com/products/matlab/app-designer.html (último acceso, junio 2021)

[9] *OPC Toolbox* (versión 2021a). Software Mathworks. http://es.mathworks.com/products/opc (último acceso, junio 2021)

[10] *ROBOTSTUDIO* (versión 2020). ABB robotic. Manual del operador RobotStudio 6.01. ID de documento: 3HAC032104-005 Revision:K.

[11] *ABB* (1988), https://new.abb.com/es. (último acceso, junio 2021)

[12] *RAPID.* ABB robotic. Manual del operador: Introducción a RAPID. ID de documento: 3HAC029364-005.

[13] *OPC SERVER ABB.* ABB robotic. Manual del operador: IRC5 OPC Server help. ID de documento: 3HAC023113-001 Revision: 9.9.

[14] *ROS (Robot Operating System)*, http://wiki.ros.org/es (último acceso, junio 2021)

[15] M. A. M. San José, "*Simulación, control cinematico y dinamico de robots comerciales usando la herramienta Matlab, "Robotic Toolbox"* ", Valladolid,2014.

[16] J. A. A. Herrero, "*Modelado de una célula robotica con fines educativos usando el programa RobotStudio*", Valladolid,2015.

[17] A. G. d. l. Santos, "*Control remoto de una celula robotizada con fines educativos mediante tecnologia WIFI*", Valladolid,2016.

[18] C. J. Jiménez, "*Diseño de un sistema robótico educativo para jugar al ajedrez con robots industriales*", Valladolid, 2019.

[19] V. L. Granado, "*Diseño, fabricación y simulación de una estación robotizada universitaria*", Valladolid, 2020.

[20] Barrientos A., Peñin L.F., Balaguer C. y Aracil P. (2007) *Fundamentos de Robótica* (2ª edición), MacGraw-Hill.

[21] *Algoritmos cinématicos inversos.* Software Mathworks. https://es.mathworks.com/help/robotics/ug/inverse-kinematics-algorithms.html (último acceso, junio 2021)

[22] Benjamin Niku S. (2010) *Introduction to Robotics. Analysis, Control, Applications* (2ª edición), Wiley.

6. ANEXOS

6.1 FICHEROS URDF DISPONIBLES EN LA ROBOTIC SYSTEM TOOLBOX

1. abbIrb120.urdf
2. abbIrb120T.urdf
3. abbIrb1600.urdf
4. abbYuMi.urdf
5. amrPioneer3AT.urdf
6. amrPioneer3DX.urdf
7. amrPioneerLX.urdf
8. atlas.urdf
9. clearpathHusky.urdf
10. clearpathJackal.urdf
11. clearpathTurtleBot2.urdf
12. fanucLRMate200ib.urdf
13. fanucM16ib.urdf
14. frankaEmikaPanda.urdf
15. kinovaGen3.urdf
16. kinovaJacoJ2N6S200.urdf
17. kinovaJacoJ2N6S300.urdf
18. kinovaJacoJ2N7S300.urdf
19. kinovaJacoJ2S6S300.urdf
20. kinovaJacoJ2S7S300.urdf
21. kinovaJacoTwoArmExample.urdf
22. kinovaMicoM1N4S200.urdf
23. kinovaMicoM1N6S200.urdf
24. kinovaMicoM1N6S300.urdf
25. kinovaMovo.urdf
26. rethinkBaxter.urdf
27. robotisOP2.urdf
28. robotisOpenManipulator.urdf
29. robotisTurtleBot3Burger.urdf
30. robotisTurtleBot3Waffle.urdf
31. robotisTurtleBot3WaffleForOpenManipulator.urdf
32. robotisTurtleBot3WafflePi.urdf
33. robotisTurtleBot3WafflePiForOpenManipulator.urdf
34. universalUR10.urdf
35. universalUR3.urdf
36. universalUR5.urdf
37. valkyrie.urdf
38. willowgaragePR2.urdf
39. yaskawaMotomanMH5.urdf

6.2 ROBOTS INCLUIDOS EN LA LIBRERÍA ARTE

ABB

- **IRB 140**
- **IRB 120**
- **IRB 1600**.
  - IRB 1600-6/1.2
  - IRB 1600-6/1.4
  - IRB 1600-6/1.45
- **IRB 1600ID**
- **IRB 2400**
- **IRB 4400**
- **IRB 4600**
- **IRB 52**
- **IRB 6620**
- **IRB 6620LX**
- **IRB 6650S**
  - IRB 6650S-125/3.5
  - IRB 6650S-200/3.0
  - IRB 6650S-90/3.9
- **IRB 760**
- **IRB 7600**
  - IRB 7600 150-350 M2000
  - IRB 7600 400-255 M2000
  - IRB 7600 500-230 M2000

ADEPT

- **Viper s1700D**

EPSON

- **ProSix C3-A601C**

FANUC

- **LR MATE 200iC**

KUKA

- **KR5 2ARC HW**
- **KR 5 arc**
- **KR5 sixx R650**
- **KR5 sixx R850**
- **KR5 scara R350 Z200**
- **KR 6-2**
- **KR 16 arc HW**
- **KR 30 L16-2**
- **KR 30 jet**
- **KR 90 R2700 pro**
- **KR 90 R3100 EXTRA**
- **KR 1000-1300 TITAN**

MITSUBISHI

- PA-10 6 DOF
- RV-6S

STÄUBLI

- **RX160L**
- **RX170BL**

UNIMATE

- **PUMA 560**

EXAMPLE ARMS

- **The Stanford arm**
- A SCARA example arm.
- A 2 DOF planar arm.
- A 3 DOF planar arm.

- A 3 DOF spherical arm.

PARALLEL ROBOTS

- A 5R parallel robot
- A 3RRR parallel robot
- A Delta robot

6.3 FUNCIONES *PIEZA()* Y *PEGAREN()*

```
function Pieza(figIcono, figPieza)
% Devuelve una variable de nombre el texto de figIcono a la base.
     if nargin<1
          error('Pieza(figIcono, figPieza)')
     end
     pieza.base= zeros(1,6);
     pieza.tcp= zeros(1,6);
     pieza.masa= 0;
     pieza.cdg= [0,0,0];
     pieza.inercia= [1,1,1]*1e-3;
     pieza.body= '';
     pieza.color= [rand(1,3), 0.7];
     if nargin==1
        figPieza= 'Null';
     end
     pieza.figIcono= figIcono;
     eval(['! copy .\Biblioteca\', figPieza,'.stl  ','.\auxStl\',
figIcono,'.stl'])
     eval(['! copy .\Biblioteca\', figPieza,'.png  ','.\auxStl\',
figIcono,'.png'])
     % Devuelve el valor en la variable de la base figIcono
     assignin('base', figIcono, pieza);
end
```

```
function PegarEn(figIcono, pieza1, base)
   if nargin<2
      error('[pieza2, pieza1]= Pegar(figIcono, pieza1, base)')
   elseif nargin==2
      base= zeros(1,6);
   end
   pieza2= pieza1;
   pieza2.base= base;
   pieza2.figIcono= figIcono;
   eval(['! copy .\auxStl\',pieza1.figIcono,'.stl
','.\auxStl\',pieza2.figIcono,'.stl']);
   eval(['! copy .\auxStl\',pieza1.figIcono,'.png
','.\auxStl\',pieza2.figIcono,'.png']);


   assignin('base', figIcono, pieza2);
   Pieza(pieza1.figIcono);
end
```

6.4 CLASE *HMAT()*

```
function this= Hmat(H)
       % Constructor: Crea una matriz homógenea objeto
       % Con argumento de entrada H: almacena matriz homogénea
            /R rotación /t traslación

function T= H(this, Pose)
       % Con argumentos: Introduce una T
       % Sin argumentos: Muestra la T del objeto

function Deg(this)
       % trabaja en grados

function Rad(this)
       % Trabaja en radianes

function disp(this)
       % Representa la matriz homogénea (R | t)

function Tinv= Inv(this)
       % Devuelve la T invertida

function obj3= mtimes(obj1, obj2)
       % Multiplica dos objetos.
       % Devuelve el objeto resultante

function out= Plot(this, mag, name)
       % Muestra la posición de los ejes en MATLAB
       % mag indica la longitud y ancho de los ejes
       % name permite nombrar el eje representado

function T= T3pts(this, pts)
       % Genera un eje a partir de tres puntos (pts)
       %    Dos puntos en el eje x
       %    Un punto en el eje y

function [x, y, z]= Txyz(this, x, y, z)
       % Trasforma vectores de puntos por T
       % La entrada puede ser en tres bloques x, y, z
       %     x, y, z pueden ser matriz o vector
       % La entrada puede ser en un bloque compacto xyz

function vect= Tpose(this, vect, mag, form)
       % Trasforma vectores de puntos y ángulos o cuaternios por T
       % Entrada: vect la matriz de puntos y ángulos
       %          mag: Magnitud para el plot
       %          form: es el formato de entrada (defecto tRzyx).
       % Salida: vect trasformado. Puede dibujar los target en el
     espacio
```

```
function T= Offs(this, vec)
       % Modifica T en la posición relativa de vec

function r = t(this, x, y, z)
       % Con argumentos: Convierte [x,y,z] en T
       % Sin argumentos: Muestra [x,y,z] de T

function r= R(this, Rot)
       % Con argumentos: Introduce una rotación R en T
       % Sin argumentos: Muestra la matriz de rotación de T

function r = Ry(this, t)
       % Introduce una rotación del eje y en T

function r = Rz(this, t)
       % Introduce una rotación del eje z en T

function T= DH(this, DHm, joints, type)
       % Convierte una matriz DH (DHm) en T
       %   joints: Cinemática directa
       %   type: Tipo de articulación
       %         (rotacional (defecto) o prismática)

function PlotDH(this, DHm, joints, mag, type)
       % Dos argumentos: Muestra en pantalla el DH
       % Tres argumentos: Cinemática directa (joints)
       % mag: magnitud de los  ejes del robot
       % type: Tipo de articulación 'R' rotacional 'P' prismática

function [DHopt, err]= IDH(this, w, Nt)
       % [DHopt, err]= IDH(this, w, Nt)
       % Obtiene los parámetros DH que optimizan la matriz homogénea
       %     w: peso de cada componente [x,y,z,Rx,Ry,Rz]
       %     Nt: Repeticiones del algoritmo de optimización local

function out = tRzyz(this, vect)
       % Con argumentos: Transforma [x,y,z,Rz,Ry,Rz] en T
       %                 Transforma [Rz,Ry,Rz] en T
       % Sin argumentos: Transforma T en [x,y,z,Rz,Ry,Rz]
       % Peter Corke

function out= tRxyz(this, vect)
       % Con argumentos: Transforma [x,y,z,Rx,Ry,Rz] en T
       %                 Transforma [Rx,Ry,Rz] en T
       % Sin argumentos: Transforma T en [x,y,z,Rx,Ry,Rz]
       % Peter Corke
```

```
function out= tRzyx(this, vect)
       % Con argumentos: Transforma [x,y,z,Rz,Ry,Rx] en T
       %                 Transforma [Rz,Ry,Rx] en T
       % Sin argumentos: Transforma T en [x,y,z,Rz,Ry,Rx]
       % Peter Corke

function out= tRyxz(this, vect)
       % Con argumentos: Transforma [x,y,z,Ry,Rx,Rz] en T
       %                 Transforma [Ry,Rx,Rz] en T
       % Sin argumentos: Transforma T en [x,y,z,Ry,Rx,Rz]
       % Peter Corke

function out= tQ(this, Q)
       % Con argumentos: Transforma [x,y,z,q1,q2,q3,q4] en T
       %                 Transforma [q1,q2,q3,q4] en T
       % Sin argumentos: Transforma T en [x,y,z,q1,q2,q3,q4]
       % Peter Corke

function [qt, qdt, qddt] = jTraj(this, q0, q1, tv, qd0, qd1)
       % [Q,QD,QDD] = JTRAJ(Q0, QF, M) is a joint space trajectory Q
      (MxN) where the joint coordinates vary from Q0 (1xN) to QF (1xN).
      A quintic (5th order) polynomial is used with default zero
      boundary conditions for velocity and acceleration.
       % Time is assumed to vary from 0 to 1 in M steps.  Joint velocity
      and acceleration can be optionally returned as QD (MxN) and QDD (
      MxN) respectively.
       % The trajectory Q, QD and QDD are MxN matrices, with one row per
      time step, and one column per joint.
       % [Q,QD,QDD] = JTRAJ(Q0, QF, M, QD0, QDF) as above but also
      specifies initial QD0 (1xN) and final QDF (1xN) joint velocity for
      the    trajectory.
       % [Q,QD,QDD] = JTRAJ(Q0, QF, T) as above but the number of steps
      in the trajectory is defined by the length of the time vector T
      (Mx1).
       % [Q,QD,QDD] = JTRAJ(Q0, QF, T, QD0, QDF) as above but specifies
      initial and final joint velocity for the trajectory and a time
      vector.

function pose = tRzyxTraj(this, pose1, pose2, r, traj)
           % Normaliza los ángulos para hallar la interpolación por el
           % lugar más cercano
           % Se  interpola  desde pose 1 a pose 2  en tralación y
           rotación.

function [rot1, rot2]= dRot(this, rot1, rot2)
       % Modifica rot2 de forma que al interpolar entre rot1-rot2
       % se gire por el lugar más cercano.
       % Ejemplos: rot1= -180 rot2= 180 -> rot2= -180 (en radianes)
       %           rot1= 120 rot2= -120 -> rot2= 240 (en radianes)
```

6.5 FUNCIONES OBJETO *KIN*

```
function this= Kin(robot, mdl, tool, q)
            % this= Kin(robot, mdl, tool, q)
            % sincroniza el modelo importado en RigidBody, robot
            % con estación de SIMULINK, (Simscape Multibody), mdl
            % tool es el efector final con el que se trabaja
            % q es la posición articular incial


    function Body(this, name, body0, Htool, stl, Hstl, type)
            % Body(this, name, body0, Htool, stl, Hstl, type)
            % Introduce cuerpos al objeto Rigid-Body


    function DelBody(this,name)
            % Borra cuerpos del objeto Rigid-Body

    function Tool(this, name, body0, Htool, stl, Hstl)
            % Tool(this, name, body0, Htool, stl, Hstl)
            % 2 argumentos: Modifica la herramienta a name (Rigid-
              Body
            % Mayor de 2. Introduce una herramienta nueva en Rigid-
              Body


    function Wobj(this, name, Hwobj, stl, Hstl)
            % Wobj(this, name, Hwobj, stl, Hstl)
            % Introduce un objeto nuevo en el objeto Rigid-Body


    function robot= Robot(this)
            % robot= Robot(this)
            % Devuelve el objeto RigidBody asociado al objeto Kin


    function pose= Pose(this, q, Hwobj)
            % pose= Pose(this, q)
            % Cinemática directa
            % Devuelve la posición pose= [[x,y,z]mm,[Rz,Ry,Rx]rad]
            % Si se añade el argumento Hwobj se convierte los pose a
              ese eje
            % correspondiente a q o por defecto la posición del
              robot actual


    function q= Joint(this, pose, w)
            % q= Joint(this, pose, w)
            % Cinemática inversa
            % Devuelve la posición q correspondiente a pose=
              [[x,y,z]mm,[Rz,Ry,Rx]rad]
```

```
 % w es el peso dado a cada elemento pose para su obtención por
 % optimización. El peso es [Rz,Ry,Rx,x,y,z] (de 0-1).
 % Ejemplo [0,0,0,1,1,1] solo optimiza la posición


function q= MoveAbsJ(this, qN, nPts)
        % q= MoveAbsJ(this, qN, nPts)
        % Posiciones q para mover la tool eleguida hasta la
          posición
        % qN en nPts pasos
        % Si no hay argumentos de salida, se simula el
          movimiento


function [q, poseN]= MoveJ(this, poseN, Hwobj, nPts)
        % [q, poseN]= MoveJ(this, poseN, Hwobj, nPts)
        % Posiciones en q y pose= [[x,y,z]mm,[Rz,Ry,Rx]rad]
        % desde la posición actual a poseN.
        % Se mueven todos los ejes de forma simultánea
        % Se toma por referencia Hwobj (Matriz homogénea)
          defecto base
        % Si no hay argumentos de salida, se simula el
          movimiento


function [q, poseN]= MoveL(this, pose1, Hwobj, nPts)
        % [q, poseN]= MoveL(this, pose1, Hwobj, nPts)
        % Posiciones en q y pose= [[x,y,z]mm,[Rz,Ry,Rx]rad]
        % del movimiento lineal entre la posición actual del
          tool
        % y la posición final pose1
        % Se toma por referencia Hwobj (Matriz homogénea)
          defecto base
        % Si no hay argumentos de salida, se simula el
          movimiento

function [q,poseN]= MoveC(this, y1, x1, Hwobj, nPts)
        % [q,poseN]= MoveC(this, y1, x1, Hwobj, nPts)
        % Solo admite puntos y1 x1, con coordenadas [x,y,x], sin
          ángulos
        % El ángulo es el de la posición que se parte
        % Posiciones q y pose [[x,y,z]mm,[Rz,Ry,Rx]rad]
        % del punto actual a x1 pasando por y1 en arco de
          circunferencia
        % Si no hay argumentos de salida, se simula el
          movimiento


function Show(this, q, pose, t)
        % Show(this, q, pose, t)
        % Simula los cambios q en el sistema RigidBody (matlab)
```

```
        % o MultiBody (Simulink)

        % Si q=[] solo rastrea las posiciones pose en Simulink
```

```
    function Sim(this, type, value,  mdl)
            % Sim(this, type, value, mdl)
            % Parámetros para simulación (Simulink)
            %    mdl nombre del fichero
            %    type 'T' intervalo fijo 'V' velocidad fija
            %    value valor del T o V

    function Rec(this, value)
            % Control de grabación de simulación
            % Value=1 empieza a grabar
            % Value=0 deja de grabar

    function Rep(this)
            % Simula las posiciones grabadas
```

6.6 ÍNDICE DE PROGRAMAS

Manual_URDF_Mat_Sim

- Definición del Objeto Rigid-Body (Matlab Robotic Toolbox)
- Definición del Objeto Multi-Bodt (Simulink SimScape Toolbox)
- Definición de los parámetros de un robot a partir de un fichero *.urdf
- Lectura de fichero *.urdf como Rigid-Body
- Lectura de fichero *.urdf como Multi-Body
- Conversión de diagrama Multi-Body a objeto Rigid-Body

Manual_Clase_Kin_Mat

- Definición de la clase Kin para manipulación de objetos Rigid-Body (Robotic Toolbox)
- Crear un objeto Kin a partir de un objeto Rigid-Body.
- Añadir nuevos objetos de trabajo y herramientas al objeto inicial.
- Movimiento de la herramienta del objeto, MoveAbsJ, MoveJ, MoveL, MoveC
- Grabación de trayectorias y reproducción de las mismas.

Manual_Clase_Kin_Sim

- Creación de un diagrama Simulink Multi-Body a partir de los robots importados de *.urdf, objetos de trabajo y herramientas.
- Conversión usando Kin en un objeto Rigid-Body, con las herramientas controladas.
- Simulación de movimientos en Multi-Body usando Kin. Visualización en Mechanics-Explore.
- Movimiento MoveAbsJ, MoveJ, MoveL, MoveC.
- Grabación de trayectorias y reproducción de las mismas.
- Descripción de app para mover el robot como Teacher.

Manual_Clase_Hmat

Describe la clase Hmat:

- Creación de ejes por traslación-rotación simple.
- Creación de ejes por tres puntos.
- Movimiento de puntos entre ejes.
- Conversión de ejes en sus distintas parametrizaciones, Matriz Homogénea, traslación-rotación Euler, Cardano, ángulo y cuaternios.
- Creación de trayectorias y movimiento de trayectorias entre ejes.
- Creación de ejes por la parametrización de Denavit–Hartenberg (D-H).

Ejemplo_Hmat_pAjedrez

- Ejemplo de aplicación de la clase Hmat a un sistema Multibody compuesto de piezas estáticas.
- Piezas de ajedrez referidas a distintos sistemas de referencia.
- Análisis de las relaciones y transformaciones que tienen lugar en el movimiento de las piezas de ajedrez sobre el tablero.

Ejemplo_CD_simq

- Ejemplo de simulación de un modelo Multi-Body sin el uso del objeto Kin.
- Cinemática directa: Movimiento a partir del valor de los ángulos articulares.

Ejemplo_CI_ik

- Cinemática inversa.
- Ejemplo de movimiento de un modelo Multi-Body a partir de señal $[x, y, z, R_z, R_y, R_x]$.
- Uso de iconos de Rigid-Body de cinemática inversa.

Ejemplo_Cin_KIN

- Ejemplo de simulación de movimientos en Multi-Body usando Kin. Visualización en Mechanics-Explore.
- Modelo creado a partir de iconos de objetos de trabajo y herramientas.
- Conversión usando Kin en un objeto Rigid-Body.
- Movimiento MoveAbsJ, MoveJ, MoveL, MoveC.
- Grabación de trayectorias y reproducción de las mismas.

Ejemplo_Control

- Ejemplo de sistema con entrada par y sistema de control en lazo cerrado.
- Controlador proporcional, proporcional-derivativo, proporcional-integral.
- Simulación del sistema para distintos controladores viendo su dinámica y vibraciones.

Ejemplo_Sim_OPC

- Ejemplo de estación básica con movimientos en q.
- Comunicación usando OPC con el servidor ABB.

Estacion_Ejemplo_abbIrb120

- Ejemplo de uso de la función Piezas para crear objetos y herramientas.
- Ejemplo de uso de la función PonerEn para cambiar objetos y herramientas.
- Sincronizar los movimientos del robot para que parezca que el robot las mueve.

Estacion_Ejemplo_irb120DH

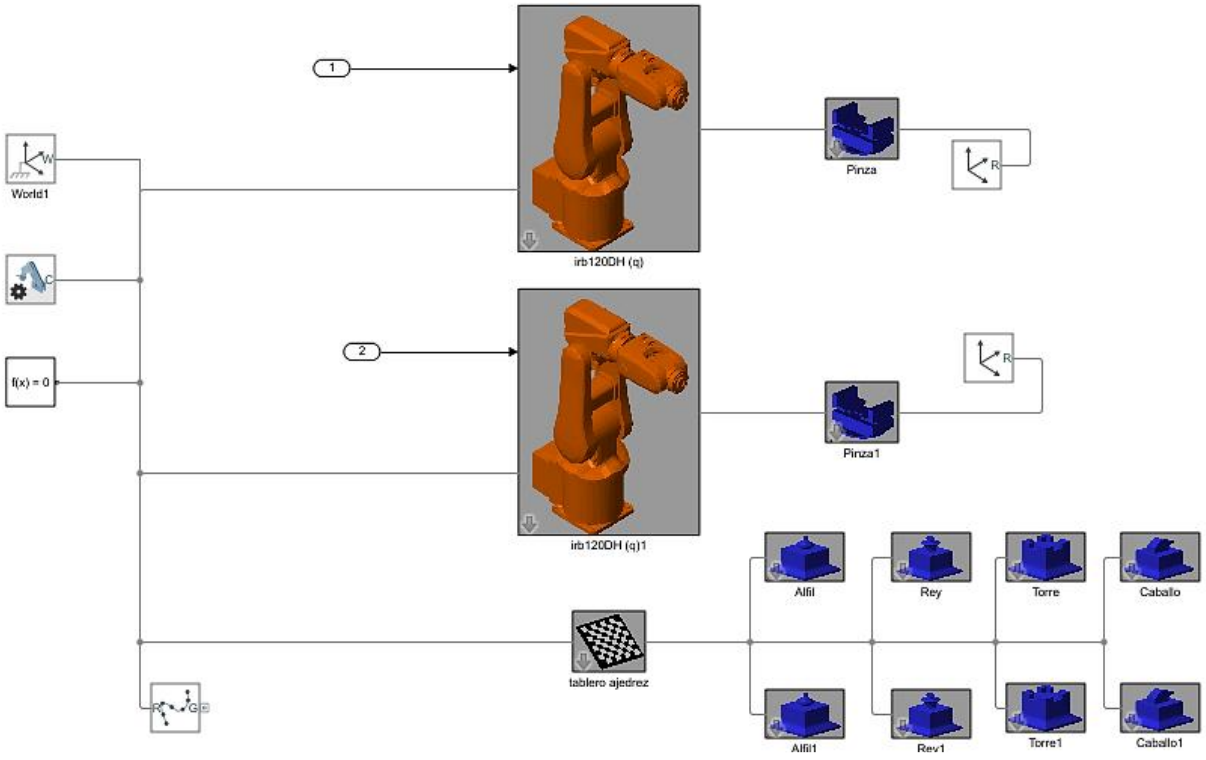
- Modelado un robot a partir de sus parámetros de Denavit–Hartenberg (D-H). No es preciso tener su fichero *urdf.
- Mismo ejemplo que el anterior.

6.7 ESTACIONES DISEÑADAS

- ***Estacion_Ejemplo_dobleirb120DH_Init***

```
clear all
clear robot rob
deg= pi/180;
%%
Estacion= 'Estacion_Ejemplo_irb120DH.slx';
open_system(Estacion)
% Se inicia la clase con el robot con el interface de simulink
% VARIABLES DE SIMULINK

% Posición inicial del robot y del rastreador
q0= zeros(6,1);
pose= randn(3,3)*1e-3;

% Posición del robot
robot1.base= [[0,0,0]*1e-3, [0,0,0]*deg];

% Tool load y wobj Se genera una variable estructura

Pieza('Pinza', 'Pinza_Completa');
Pinza.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza.color= [0.4,0,0.4,0.7];
Pieza('Carga', 'Lapiz');
Carga.tcp= [[0,0,148]*1e-3, [0,0,0]*deg];
Carga.color= [0,0.4,0.4,0.7];
Pieza('Esfera', 'Esfera');
Esfera.base= [[0,-350,0]*1e-3, [0,0,0]*deg];
Esfera.color= [0,0.3,0.8,0.7];
Pieza('Tablero', 'Tablero');
Tablero.base= [[150,-200, 150]*1e-3, [0,0,0]*deg];
Tablero.color= [0.3, 0.8, 0.2, 0.7];
Pieza('Cilindro', 'Cilindro');
Cilindro.base= [[0, 0, 900]*1e-3, [0, 90, 0]*deg];
Cilindro.color= [0.4,0.7,0.2,0.7];

% Load vacio para poner el lapiz fuera de la carga
Pieza('Lapiz');

Pieza('Torre','Torre');
% Poner la torre en función del tablero (wobj2)
%load2.base= [[525, -175, 155]*1e-3, [0, 0, 0]*deg];
h= Hmat; h.Rad;
Px= 3; Py= 4;
h.H(h.tRzyx(Tablero.base)*h.tRzyx([[25+50*Px,25+50*Py,7]*1e-
3,[0,0,0]*deg]));
Torre.base= h.tRzyx;
Torre.color= [0,0,0,1];

%% Importar el fichero de Multi-body con RigidBody
% Se obtiene la estructura sin las gráficas.
% Se obtiene dos esquemas idénticos.
```

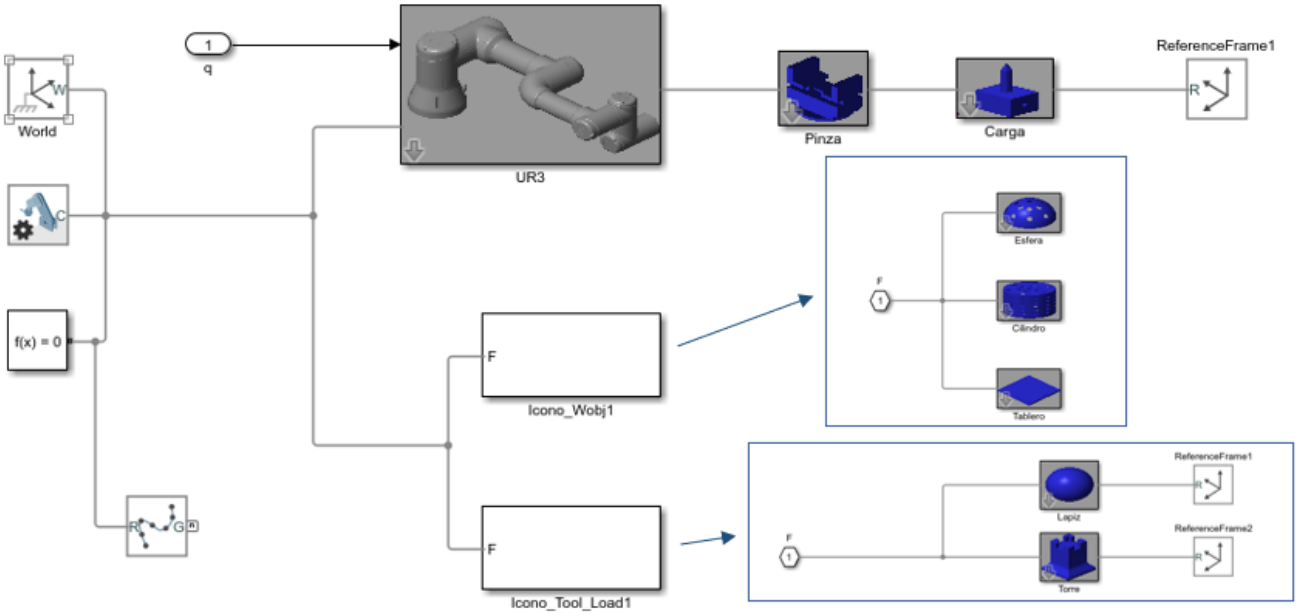
```
% Hay que ver en detalles qué cuerpos son las herramientas y cargas. Los
numera por orden de aparición en SIMULINK.
robot= importrobot(Estacion);
robot.showdetails
% Puntos de interes para cinemática inversa

% Efectores finales
Pinza.body= 'Body10';
Carga.body= 'Body11';
```



➢ *Estacion_Ejemplo_UniversalU3_Init*

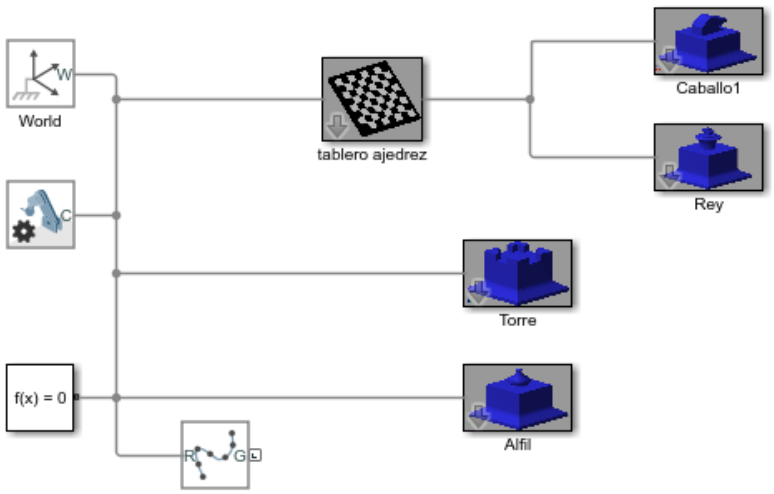
```
clear all
clear robot1 rob
deg= pi/180;

Estacion= 'Estacion_Ejemplo_UniversalU3.slx';
open_system(Estacion)
% Se inicia la clase con el robot con el interface de SIMULINK
% Variables de SIMULINK

% Posición inicial del robot y del rastreador
q0= [0,-90,0,-90, -90,0]*deg;
pose= randn(3,3)*1e-3;

% Posición del robot
robot1.base= [[0,0,0]*1e-3, [0,0,0]*deg];

% Tool load y wobj Se genera una variable estructura

Pieza('Pinza', 'Pinza_Completa');
Pinza.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza.color= [0.4,0,0.4,0.7];
Pieza('Carga', 'Lapiz');
Carga.tcp= [[0,0,148]*1e-3, [0,0,0]*deg];
Carga.color= [0,0.4,0.4,0.7];
Pieza('Esfera', 'Esfera');
Esfera.base= [[0,-350,0]*1e-3, [0,0,0]*deg];
Esfera.color= [0,0.3,0.8,0.7];
Pieza('Tablero', 'Tablero');
Tablero.base= [[150,-200, 150]*1e-3, [0,0,0]*deg];
Tablero.color= [0.3, 0.8, 0.2, 0.7];
Pieza('Cilindro', 'Cilindro');
Cilindro.base= [[0, 0, 900]*1e-3, [0, 90, 0]*deg];
Cilindro.color= [0.4,0.7,0.2,0.7];

% Load vacio para poner el lapiz fuera de la carga
Pieza('Lapiz');

Pieza('Torre','Torre');
% Poner la torre en función del tablero (wobj2)
%load2.base= [[525, -175, 155]*1e-3, [0, 0, 0]*deg];
h= Hmat; h.Rad;
Px= 3; Py= 4;
h.H(h.tRzyx(Tablero.base)*h.tRzyx([[25+50*Px,25+50*Py,7]*1e-
3,[0,0,0]*deg]));
Torre.base= h.tRzyx;
Torre.color= [0,0,0,1];
```

```
% Importar el fichero de Multi-body con RigidBody
% Se obtiene la estructura sin las gráficas.
% Se obtiene dos esquemas idénticos.
% Hay que ver en detalles qué cuerpos son las herramientas y cargas. Los
numera por orden de aparición en SIMULINK.
robot= importrobot(Estacion);
robot.showdetails;
% Puntos de interes para cinemática inversa

% Efectores finales
Pinza.body= 'Body11';
Carga.body= 'Body12';
```



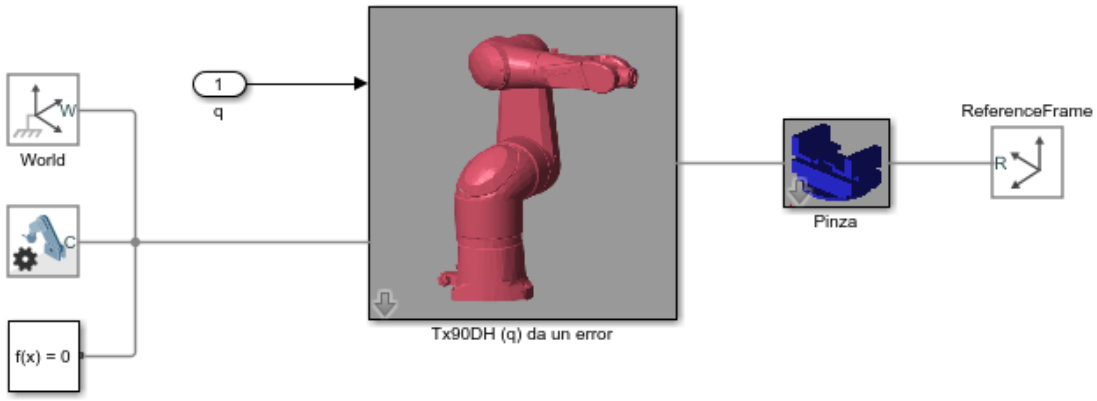
➢ *Estacion_ajedrez*

```
clear all
clear robot1 rob
deg= pi/180;
Estacion= 'Estacion_ajedrez.slx';

%definición del tablero de ajedrez
%no uso de función Pieza, no se puede asignar un solo *.stl
%presenta *.stl para figuras blancas/*.stl figuras negras
tab_ajedrez.color=[0,0,0,1];
tab_ajedrez.color1=[1,1,1,1];
pose= randn(3,3)*1e-3;
tab_ajedrez.base= [[0,200,0]*1e-3, [0,0,0]*deg];
tab_ajedrez.masa= 0;
tab_ajedrez.cdg= [0,0,0];
tab_ajedrez.inercia= [1,1,1]*1e-3;
tab_ajedrez.tcp= zeros(1,6);

%Torre relativa coord. globales
Pieza('Torre','Torre');
Torre.base= [[225,75, 0]*1e-3, [0,0,0]*deg];
Torre.color= [0.9, 0.8, 0.2, 1];
%Alfil relativo coord.. globales
Pieza('Alfil','Alfil');
Alfil.base= [[300,-200, 0]*1e-3, [0,0,0]*deg];
Alfil.color= [0, 0.8, 0.7, 1];
%caballo relativo al tablero de ajedrez
Pieza('Caballo','Caballo');
Caballo.color= [0.163, 0.73, 0.164,1];
Caballo.base=([[125,75,0]*1e-3,[0,0,0]*deg])
%Rey relativo al tablero de ajedrez
Pieza('Rey','Rey');
Rey.color= [0.9, 0, 0.8,1];
Rey.base= [[225,75,0]*1e-3, [0,0,0]*deg];
```



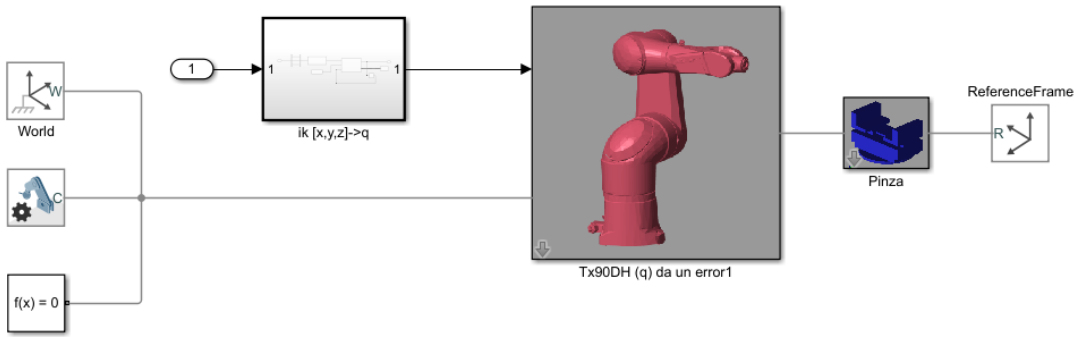
- ***Estacion_CD_sim_q***

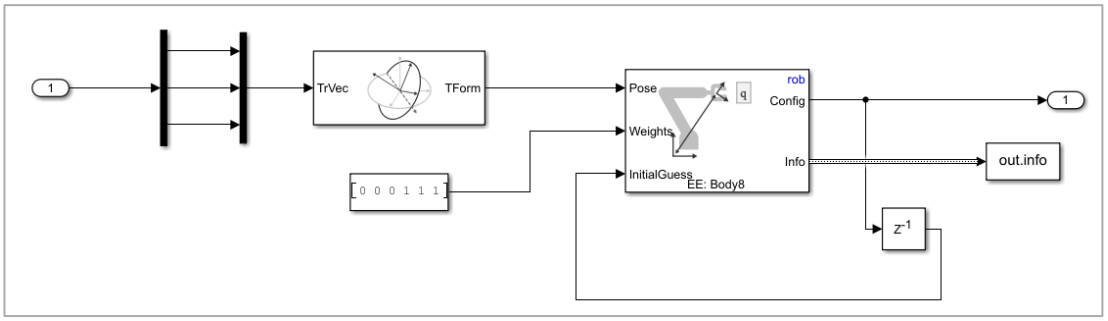
```
clear all
clear robot1 rob
deg= pi/180;

Estacion= 'Estacion_CD_sim_q.slx';
% open_system(Estacion)
% Se inicia la clase con el robot con el interface de simulink

% Variables de SIMULINK

% Posición inicial del robot y del rastreador
q0= zeros(6,1);
pose= randn(3,3)*1e-3;

% Posición del robot
robot1.base= [[0,0,0]*1e-3, [0,0,0]*deg];

% Tool load y wobj Se genera una variable estructura de nombre tool1
Pieza('Pinza', 'Pinza_Completa');
Pinza.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza.color= [0.4,0,0.4,1];
```



➢ *Estacion_CD_sim_q*

```
clear all
deg= pi/180;
Estacion= 'Estacion_CI_ik.slx';

% Posición inicial del robot
q0= zeros(1,6);

% Posición del robot
robot1.base= [[0,0,0]*1e-3, [0,0,0]*deg];

%tool
Pieza('Pinza', 'Pinza_Completa');
Pinza.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza.color= [0.4,0,0.4,0.7];

% Importar el fichero de Multi-body con RigidBody
% Se obtiene la estructura sin las gráficas.
% Se obtiene dos esquemas idénticos.
% Hay que ver en detalles qué cuerpos son las herramientas y cargas. Los
numera por orden de aparición en SIMULINK.
rob= importrobot(Estacion);
```



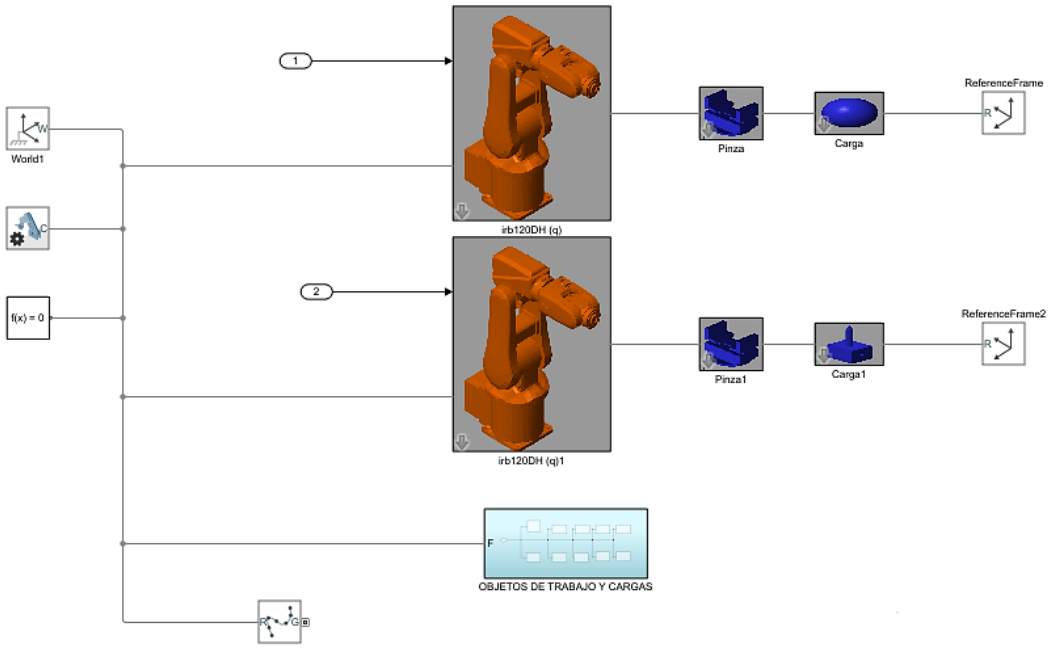


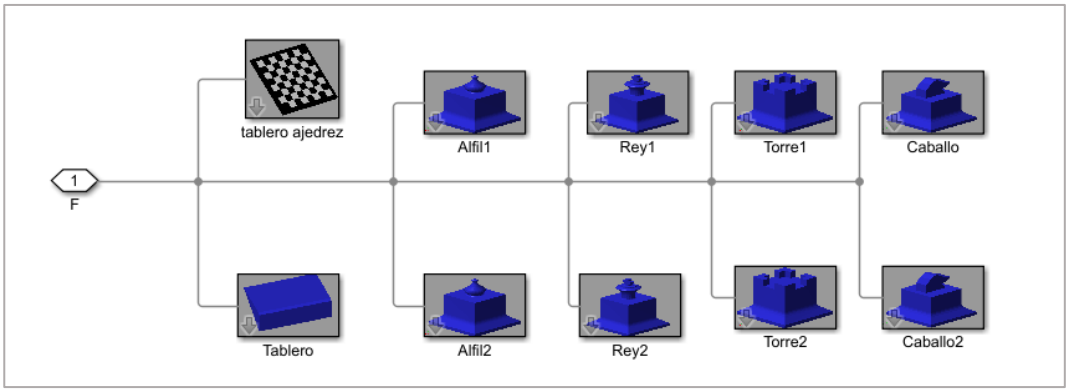
- ***Estacion_Ejemplo_dobleirb120DH_Init***

```
clear all
clear robot1 rob
deg= pi/180;
Estacion= 'Estacion_doble_irb120Cin.slx';
% open_system(Estacion)
% Se inicia la clase con el robot con el interface de simulink

% Variables de SIMULINK
% Posición inicial de los robots
pose= randn(3,3)*1e-3;
q0= zeros(6,1);
q02= zeros(6,1);
robot1.base= [[0,0,0]*1e-3, [0,0,0]*deg];
robot2.base= [[1200,-400,0]*1e-3, [230,0,0]*deg];

% Tool load y wobj Se genera una variable estructura
Pieza('Pinza1','Pinza_Completa');
Pinza1.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza1.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza1.color= [0.4,0,0.4,0.7];
Pieza('Pinza2','Pinza_Completa');
Pinza2.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza2.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza2.color= [1,1,0,1];
Pieza('MesaInclinada', 'mesa_inclinada');
MesaInclinada.base= [[1000,-450, 0]*1e-3, [-100,0,0]*deg];
MesaInclinada.color= [1, 0.4, 0.5, 0.8];

% Carga vacía para coger las piezas
Pieza('Carga1');
Pieza('Carga2', 'Lapiz');
Carga2.tcp= [[0,0,148]*1e-3, [0,0,0]*deg];
Carga2.color= [0,0.4,0.4,1];

%definición del tablero de ajedrez
%no uso de función Pieza, no se puede asignar un solo *.stl
%presenta *.stl para figuras blancas/*.stl figuras negras
tab_ajedrez.color=[0,0,0,1];
tab_ajedrez.color1=[1,1,1,1];
pose= randn(3,3)*1e-3;
tab_ajedrez.base= [[275,-270,0]*1e-3, [0,0,0]*deg];
tab_ajedrez.masa= 0;
tab_ajedrez.cdg= [0,0,0];
tab_ajedrez.inercia= [1,1,1]*1e-3;

%PIEZAS BLANCAS
%Torre
Pieza('Torre1','Torre');
```

```
h= Hmat; h.Rad;
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[75,25, 0]*1e-3, [0,0,0]*deg]));
Torre1.base= h.tRzyx;
Torre1.color= [1, 1, 1, 1];
%Alfil
Pieza('Alfil1','Alfil');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[175,25, 0]*1e-3, [0,0,0]*deg]));
Alfil1.base= h.tRzyx;
Alfil1.color= [1, 1, 1, 1];
%caballo
Pieza('Caballo1','Caballo');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[275,25,0]*1e-3,[0,0,0]*deg]));
Caballo1.base= h.tRzyx;
Caballo1.color= [1, 1, 1,1];
%Rey
Pieza('Rey1','Rey');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[375,25,0]*1e-3, [0,0,0]*deg]));
Rey1.base= h.tRzyx;
Rey1.color= [1, 1, 1,1];

%PIEZAS NEGRAS
%Torre
Pieza('Torre2','Torre');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[75,375, 0]*1e-3, [0,0,0]*deg]));
Torre2.base= h.tRzyx;
Torre2.color= [0, 0, 0, 1];
%Alfil
Pieza('Alfil2','Alfil');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[175,375, 0]*1e-3, [0,0,0]*deg]));
Alfil2.base= h.tRzyx;
Alfil2.color= [0, 0, 0, 1];
%caballo
Pieza('Caballo2','Caballo');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[275,375,0]*1e-3,[0,0,0]*deg]));
Caballo2.base= h.tRzyx;
Caballo2.color= [0, 0, 0, 1];
%Rey
Pieza('Rey2','Rey');
h.H(h.tRzyx(tab_ajedrez.base)*h.tRzyx([[375,375,0]*1e-3, [0,0,0]*deg]));
Rey2.base= h.tRzyx;
Rey2.color= [0, 0, 0, 1];
%importación estructura RigidBody
robot= importrobot(Estacion);

%Efectores finales
Pinza2.body='Body17'; %robot 2
Carga2.body='Body18';
Pinza1.body='Body09'; %robot1
Carga1.body='Body10';
```

```
Generar el objeto Kin con el robot RigidBody, el equivalente MultiBody y
la herramienta desea.
q=[0,0,0,0,0,0,0,0,0,0,0,0];
rob= Kin(robot, Estacion, Pinza1.body,q);
```

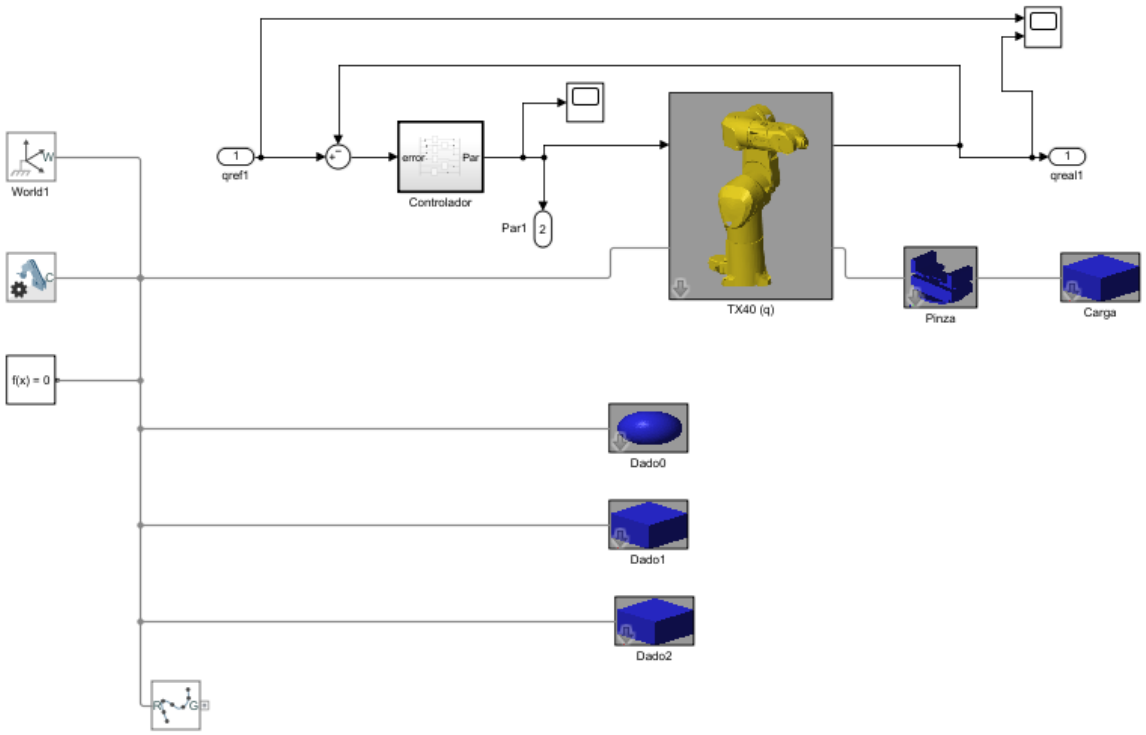




- ***Estacion_Control***

```
clear all
deg= pi/180;
Estacion= 'Estacion_Control.slx';
% open_system(Estacion)

% Variables de SIMULINK
% Posición inicial del robot
q0= zeros(1,6);
pose= rand(3,3)*1e-3;
% Posición del robot
Robot.base= [[0,0,0]*1e-3, [0,0,0]*deg];

% Tool load y wobj Se genera una variable estructura de nombre tool1
Pieza('Pinza', 'Pinza_Completa');
Pinza.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza.masa= 1;
Pinza.cdg= [0,0,0];
Pinza.inercia= [1,1,1]*1e-3;
Pinza.color= [0.4,0,0.4,0.7];
% Carga vacía para coger los dados
Pieza('Carga');

Pieza('Dado0', 'Dado'); %masa de 1kg
Dado0.base= [[400,-100,0]*1e-3, [0,0,0]*deg];
Dado0.tcp= [[0,0,0]*1e-3, [0,0,0]*deg];
Dado0.masa= 1;
Dado0.cdg= [0,0,0];
Dado0.inercia= [1,1,1]*1e-3;
Dado0.color= [0.4,0.6,0.2,1];
Pieza('Dado1', 'Dado'); %masa de 15 kg
Dado1.base= [[400,0,0]*1e-3, [0,0,0]*deg];
Dado1.tcp= [[0,0,0]*1e-3, [0,0,0]*deg];
Dado1.masa= 15;
Dado1.cdg= [0,0,0];
Dado1.inercia= [1,1,1]*1e-3;
Dado1.color= [1,0,0.7,1];
Pieza('Dado2', 'Dado'); %masa de 40 kg
Dado2.base= [[400,100,0]*1e-3, [0,0,0]*deg];
Dado2.tcp= [[0,0,0]*1e-3, [0,0,0]*deg];
Dado2.masa= 40;
Dado2.cdg= [0,0,0];
Dado2.inercia= [1,1,1]*1e-3;
Dado2.color= [0.4,0.6,1,1];

% Control
% Es preciso una K en torno a 1e5 para eliminar errores
Kcontrol= diag(1e3*ones(1,6));
```

```
%importación estructura RigidBody
robot= importrobot(Estacion);
robot.showdetails
Pinza.body= 'Body8';



Generar el objeto Kin con el robot RigidBody, el equivalente MultiBody y
la herramienta desea.
rob= Kin(robot, Estacion, Pinza.body, q0);
```



- ***Estacion_Com_q***

```
clear all
clear robot1 rob
deg= pi/180;

Estacion= 'Estacion_Com_q.slx';
% open_system(Estacion)

% VER VERSIÓN SIMULINK
% Variables de simulink

% Posición inicial del robot y del rastreador
q0= zeros(6,1);
pose= randn(3,3)*1e-3;

% Posición del robot
robot1.base= [[0,0,0]*1e-3, [0,0,0]*deg];

% Tool load y wobj Se genera una variable estructura
Pieza('Pinza', 'Pinza_Completa');
Pinza.base= [[0,0,0]*1e-3, [90,0,0]*deg];
Pinza.tcp= [[0,0,70]*1e-3, [0,0,0]*deg];
Pinza.color= [0.4,0,0.4,0.7];
Pieza('Carga', 'Lapiz');
Carga.tcp= [[0,0,148]*1e-3, [0,0,0]*deg];
Carga.color= [0,0.4,0.4,0.7];
```