ESCUELA DE INGENIERÍAS INDUSTRIALES

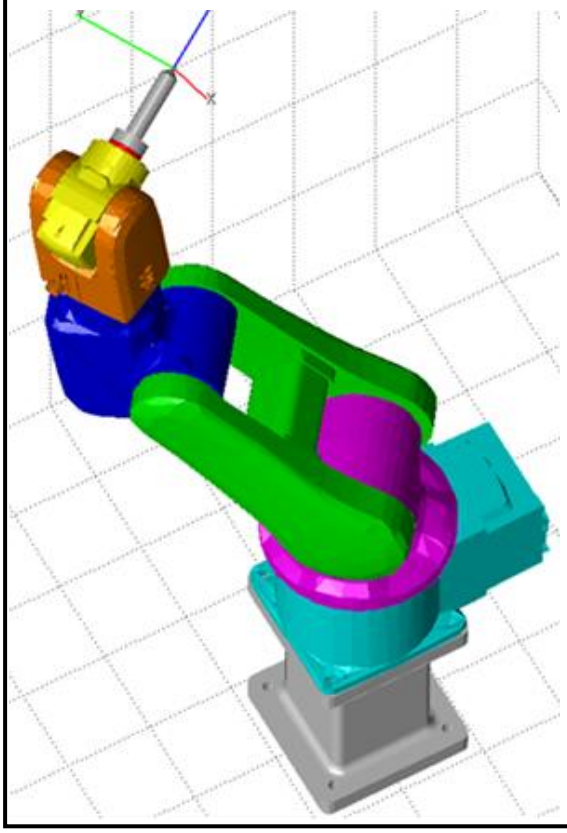
UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERÍAS INDUSTRIALES

Trabajo Fin de Grado

Grado en Ingeniería Mecánica

"Simulación, control cinemático y dinámico de robots comerciales usando la herramienta de Matlab, Robotic Toolbox"



Autor: Mato San José, Miguel Ángel.
12.370.934Q
Tutor: Dr. D. Alberto Herreros López
Departamento: Ingeniería de Sistemas y Automática

Lugar y Fecha: Valladolid, 06 de julio de 2014
Mi Ref.: tfg-irb120-6-07-14-1.docx

RESUMEN

El presente trabajo justifica y expone, la forma y el modo, en el que se ha validado el software ***Robotic Toolbox***, del entorno ***Matlab***, como herramienta de estudio, control y simulación de robots industriales.

Con dicha librería, de ***Peter Corke*** *(v.9.8-Feb 2013)*, se ha modelado numéricamente el robot manipulador ***IRB 120*** de la firma ***ABB***, desarrollando un interfaz gráfico que facilita el estudio cinemático y dinámico del robot. La aplicación contiene además mejoras de otros paquetes de software (***Solidworks***, ***SimMechanic***, ***ARTE*** y ***OPC***) que permiten al usuario interactuar, con el robot virtual y con el robot real.

Tanto el software, como el robot, son ampliamente conocidos, utilizados y accesibles en el mundo industrial y académico, y representan una buena forma de introducirse en el estudio y manejo de la robótica, para el autor y para cuantos sigan la aplicación desarrollada.

PALABRAS CLAVE:

Control, simulación, robot, IRB120, Robotic Toolbox.

ENGLISH ABSTRACT

This work justifies and outlines the form and manner, which has been validated **Robotic Toolbox** *software, the* **Matlab** *environment, as a study tool, control and simulation of industrial robots.*

With this library, by **Peter Corke** *(v.9.8-Feb 2013), has been numerically modelled the* **IRB 120** *robot manipulator by* **ABB***. We have developed a graphical interface that facilitates kinematic and dynamic study of the same. The application also contains improvements in other software packages (***Solidworks***,* **SimMechanic,** **ARTE** *and* **OPC***) that allow the user to interact with the virtual robot and the real robot*

Both the software and the robot are widely known, used and accessible in the industrial and academic world, and represent a good way to enter the study and management of robotics, for the author and for those who follow the application developed.

KEYWORDS

Control, simulation, robot, IRB120, Robotic Toolbox.

INDICE GENERAL

INDICE de FIGURAS

INDICE de TABLAS

CAPÍTULO I

1. INTRODUCCIÓN

1.1. OBJETIVOS

Pretendemos con este estudio probar y validar la librería ***Robotic Toolbox*** del entorno ***Matlab*** [1], de ***Peter Corke*** [2], como herramienta de estudio, control y simulación de robots manipuladores.

1.2. ALCANCE

Utilizaremos dicho software para modelar, controlar y simular el robot manipulador articulado tipo serie, ***IRB 120*** de la multinacional suiza ***ABB***. Su elaboración nos permitirá conocer, además de las herramientas matemáticas en las que basan sus movimientos, la compleja tecnología que utiliza para su control cinemático y dinámico. El estudio comprende:

- Modelado numérico del robot con ***Robotic Toolbox*** de ***Matlab***: relaciones geométricas, físicas y matemáticas.
- Estudio de las relaciones cinemáticas y dinámicas del robot, simulación en el entorno ***Simulink*** [3] y ***SimMechanic*** [5] .
- Modelado del robot, en un entorno de CAD *3D*, ***SolidWorks*** [3].
- Diseño de un interfaz gráfico, con un ***Guide*** [6] de ***Matlab***, desde el cual interactuar con el modelo virtual.
- Sincronización del modelo numérico con el robot real, a través del software ***OPC,*** *(*de *Matlab* [7] y de *ABB* [8]), para finalmente validar este estudio.

1.3. JUSTIFICACIÓN Y MOTIVACIÓN

No hacía falta que el gigante tecnológico ***Google*** decidiese recientemente apostar por los robots y los drones, para saber que estos, serán un importante nicho de negocio en este recién comenzado siglo XXI. Se estima, que el parque mundial de robots podrá alcanzar en 2015 la cifra de 100 millones de unidades (Fuente: International Federation of Robotics -IFR-, 2014 [9])

Conscientes, por tanto, de la necesidad de acceder al control de estas máquinas, sus fabricantes suministran el software para su control y simulación. Aprovechando la actual tendencia a sistemas abiertos, en los que los usuarios pueden acceder al control de los parámetros de funcionamiento, nos lleva a comprobar si *Matlab*, y su librería *Robotic Toolbox*, es capaz de cumplir esta función.

Además, la existencia del robot *IRB 120* de *ABB* en el departamento, permitió validar los resultados en el entorno gráfico-numérico, fundamental en cualquier estudio de simulación.

Finalmente, la aplicación desarrollada para la simulación del robot, es un excelente recurso didáctico, que permite a cualquier usuario introducirse en el estudio y manejo de la robótica, pudiendo elaborar sus propios algoritmos de control del robot.

1.4. ESTRUCTURA DEL PROYECTO

Este proyecto consta de siete capítulos y tres anexos:

- Capítulo I: Introducción

  De carácter introductorio, se presentan las motivaciones que me han llevado a desarrollar este proyecto, justificándolo y describiendo el alcance del mismo.

- Capítulo II: Estado del Arte

  Se presenta una introducción acerca de los robot industriales, del estado de la simulación y control de los mismos, y de las actuales tendencias en este campo.

- Capítulo III: Modelado y simulación del Robot

  Describe los pasos, desarrollo y fundamentos utilizados en el modelado del robot y desarrollo del interfaz gráfico.

- Capítulo IV: Estudio económico

  Referido al desarrollo y ejecución de este trabajo.

- Capítulo V: Conclusiones y futuras líneas de investigación

  Se presenta las conclusiones obtenidas, y se exponen las futuras líneas de investigación.

- Capítulo VI: Bibliografía.

A N E X O S:

- ANEXO I: Código elaborado.

CAPÍTULO II

2. ESTADO DEL ARTE

2.1. INTRODUCCIÓN

Desde que en 1961, apareciera el ***Unimate***, primer robot manipulador, hasta nuestros días, la continua y espectacular evolución de estas maquinas, ha permitido sustituir al hombre en aquellas tareas repetitivas, tediosas y peligrosas, en todo tipo de industrias y sectores, de manera eficiente y rentable.

Si en un principio, la evolución de la robótica fue ligada a la evolución de la mecánica y sus sistemas de accionamiento, más precisos y menos pesados, en la actualidad, la tecnología de fabricación de los robots, va directamente ligada al sector de la electrónica, informática y telecomunicaciones. Incorporando motores, sensores, comunicaciones y procesadores, más potentes, rápidos y económicos.

En los cincuenta años transcurridos, la precisión de posicionamiento ha mejorado desde 1 mm hasta 10 micras, los interfaces de usuario, desde una lectura de LED de 4 dígitos hasta una pantalla táctil completa tipo *Windows*, y la potencia de cálculo desde 8 kB hasta 20 GB o más. Al mismo tiempo, la fiabilidad ha aumentado hasta 80.000 horas de MTBF (tiempo medio entre fallos), y los costes se han reducido hasta el extremo de que ahora un robot cuesta, en términos reales, la mitad que hace sólo 15 años [10].

Además, ahora, los robots no sólo están en procesos súper automatizados de las cadenas de montaje de la automoción, ahora están presentes en otras industrias y procesos:

- Eléctrico-electrónico: montaje y ensamblaje de componentes y máquinas eléctricas o eléctrico.
- Metal-mecánica: nuevos procesos de mecanizado, soldadura o corte (laser, plasma), manipulación de piezas pesadas o peligrosas, etc.
- Fabricación y moldeo de plástico: manipulación de piezas que por su volumen, peso o temperatura, resultan difíciles de manejar.
- Farmacéutico y agroalimentario: manipulación y envasado de productos.

Podríamos decir que la robótica se ha hecho accesible a todo sector o aplicación, donde el proceso sea de alta precisión, repetitivo o peligroso, y sin necesidad de fabricaciones masivas.

2.2. CONCEPTOS BÁSICOS DE ROBOT MANIPULADOR

Un robot manipulador es un manipulador multifuncional reprogramable, capaz de mover materias, piezas, herramientas, o dispositivos especiales, según trayectorias variables, programadas para realizar tareas diversas.

Un robot está formado por los siguientes elementos: estructura mecánica, transmisiones, sistema de accionamiento, sistema sensorial, sistema de control y herramientas terminales (*tools*). El sistema de control se ubica en un controlador, separado físicamente del robot, en el cual se encuentra el software y hardware, capaz de hacer mover el robot (Figura 1).

La constitución física de los robots manipuladores tipo serie, guarda cierta similitud con la anatomía del brazo humano, de ahí el nombre de **antropomórfico**, por lo que en ocasiones, para hacer referencia a los distintos elementos que componen el robot, se usan términos como cuerpo, brazo, codo y muñeca.

Definiciones y conceptos de los robot manipuladores industriales:

- **Campo de trabajo:** volumen espacial donde el robot puede situar su extremo terminal.
- ***Tool Center Point* (*TCP*):** punto final de la herramienta de trabajo del robot, espacialmente define el alcance del robot.
- **Eslabón**: componente físico que existe entre articulaciones.
- **Articulación**: puntos que conectan los eslabones, pueden ser prismáticas, de revolución o esféricas.
- **Elemento terminal (herramienta -tool-):** dispositivos que interaccionar con el entorno del robot. Pueden ser tanto elementos de aprehensión como herramientas o útiles, en muchos casos diseñadas para cada aplicación.
- **Grados de libertad (GdL)**: número de parámetros necesarios para determinar la posición de un robot, es decir, el número de movimientos básicos e independientes que posicionan los elementos del robot en el espacio. Tres movimientos son para posicionar y tres para orientar el objeto en cuestión.
- **Sistema de coordenadas:** los movimientos y posiciones del robot se pueden especificar en coordenadas cartesianas, cilíndricas, esféricas o por coordinación manual.
- **Coordenadas angulares o articulares**: en manipuladores basados exclusivamente en la rotación. Una determinada posición del robot, que define un punto, en un instante de tiempo (t), se establece por el ángulo que forman los eslabones contiguos $Q(t)= q1,q2,q3,...,qGdL$.
- **Coordenadas homogéneas**: se determina la posición de un punto mediante una matriz 4x4 (***T***), que define la posición del *TCP*, así como la orientación del mismo, además de otros parámetros para su representación en *3D*.

- **Repetitividad**: es la precisión en la repetición de movimientos.
- **Resolución**: incremento mínimo de movimiento que puede hacer el manipulador.
- **Precisión**: distancia entre la posición de las coordenadas programadas y las posiciones reales.
- **Capacidad de carga**: peso que el robot puede llegar a manipular.
- **Células de fabricación**: entornos de trabajo en los que puede aparecen más de un robot para una misma tarea.
- **Tipo de accionamiento**: energía eléctrica, neumática o hidráulica que utilizan sus actuadores para imprimir movimientos a la estructura.
- **Actuadores**: dispositivos que dotan de movimiento al robot, pueden ser:
  - Rotacionales: motores (eléctricos, neumáticos o hidráulicos).
  - Lineales: cilindros (neumáticos o hidráulicos).
- **Sensores**: dispositivos que controlan la posición, velocidad, aceleración y masa de la carga. Existe una extensísima gama.
- **Entornos de simulación**: programas que permiten la manipulación de un robot virtual en un espacio *3D*, más o menos realista, sobre el cual podremos interactuar: parámetros de operación, control, movimientos, trayectorias, etc.
- **Programación off-line**: programación de operaciones que se pueden realizar con simulación, sin que el robot real tenga ejecutar las órdenes.

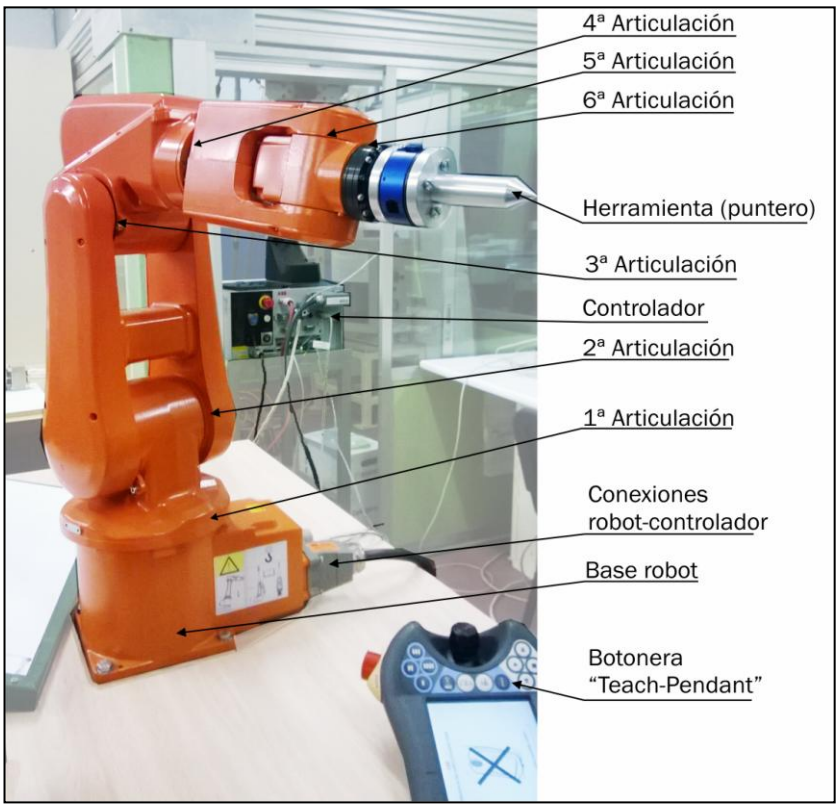


Figura 1. Elementos y partes del conjunto robótico ABB IRB120

2.3. ROBOT INDUSTRIALES Y EDUCACIONALES

En la actualidad, unas pocas multinacionales, Europeas y Japonesas, dominan el mercado mundial de robots: ***ABB***, ***Kuka***, ***Fanuc***, ***Motoman***, ***Honda*** y ***Mitsubishi*** [9] Todas tienen una amplísima gama de robots para la industria, pero además, todas, tienen pequeños robots, con la misma funcionalidad que el resto de su gama industrial, que introducen en ambientes académicos para atraer a los futuros consumidores, usuarios e investigadores.

Todas también, tienen desarrollado software propio, para el control de sus máquinas. En estos últimos años, y en tendencia con los sistemas operativos de moda -***Android***- dicho software de control se presenta en código abierto, a fin de facilitar su uso y de esta forma, ganar cuota de mercado.

Robots de similares características al robot ***IRB120*** de ***ABB*** (Figura 1 y Figura 2), con el que vamos a realizar el estudio, son:

- **KR5 sixx R650** de ***Kuka***.
- **RV-1A-S11** de ***Mitsubishi***.
- **VIPER S650** de ***Motoman***.
- **LR MATE 100iB** de ***Fanuc***.

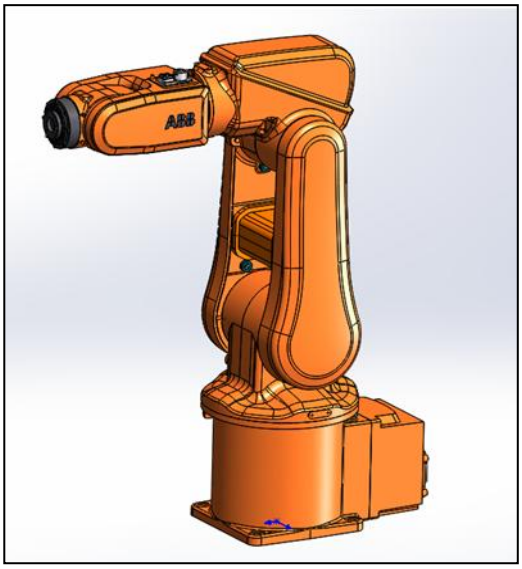


Figura 2. Robot IRB120 *CAD 3D*

Todos ellos, son robot manipuladores, con los que comparte las siguientes características:

- Arquitectura antropomórfica, tipo serie, de 6 grados de libertad.
- Reducidas dimensiones, con un alcance máximo de hasta 0,7 m.
- Capacidad de carga, de 1,5 a 6 Kg.
- Peso propio del robot, hasta 50 kg.
- Repetitividad está, entre ±0.01 y ±0.02mm.
- Precio, entorno a los 20.000 €.

Fuente: "A Brief Survey of Commercial Robotic Arms for Research on Manipulation" [11].

Decantarse por uno u otro, en un ambiente académico, dependerá en gran medida, además de su precio, de su facilidad de manejo, de las posibilidades que ofrezca su software de control y de los extras formativos que conlleve su adquisición.

En nuestro caso, el *IRB120* de *ABB*, cumplía todas las premisas, y además nuestra experiencia en el terreno industrial nos aseguraba, que la firma *ABB* tenía un importante grado de penetración en la industria automovilística de nuestro entorno.

2.4. ENTORNOS DE SIMULACIÓN DE ROBOTS

La simulación de un sistema o máquina, es el proceso de diseñar un modelo (generalmente matemático) de un sistema real o imaginario, y realizar experimentos con dicho modelo. Con los experimentos pretendemos comprender su comportamiento y evaluar estrategias de control y operación, sin requerir previamente la construcción o experimentación con el sistema físico real. La simulación en robótica, conlleva innegables ventajas:

- Simulación de nuevos diseños, con diferentes arquitecturas, dimensiones, grados de libertad, materiales, etc.
- Evaluar el comportamiento con diferentes tipos de actuadores: motores eléctricos, hidráulicos, neumáticos, etc. Igualmente con todo tipo sensores.
- Evaluar el comportamiento con diferentes tecnologías de control.
- Simular las trayectorias del robot y sus tareas, a fin optimizarlas.
- Minimizar los daños ocasionados por posibles colisiones con su entorno de trabajo.

Resulta obvio, por tanto, que todo ello conlleva un innegable ahorro económico e incremento de productividad, allí donde intervienen los robots manipuladores industriales.

Al igual que en el apartado anterior, unas pocas compañías multinacionales dominan el sector del software de simulación de robots. Este software, puede dividirse en dos grandes grupos, los paquetes de simulación de propósito general y los dedicados específicamente a robots:

- En el primer caso, encontramos software de entorno científico matemático que cubre los campos de análisis matemático, físico, sistemas control, simulaciones de todo tipo (físicas, estadísticas, financieras, etc.) De este tipo son: ***Matlab-Simulink***, ***Dymola-Modelica, 20-sim*** y ***Mathematica*** **.**
- En el segundo caso, más específicos para robots, y sobre todo multi-cuerpo, permiten robots móviles, multirobot, programación off-line, diseño de células robotizadas, análisis cinemático y dinámico, diseño mecánico, etc. Software de este tipo es: ***Gazebo***, ***ODE***, ***Bullet***, ***OpenRave***, ***Blender***, ***ARGoS***, ***Webot,*** y ***Robotran.***

Fuente: "*Tools for dynamics simulation of robots*" [12].

Existen en la red, además de las anteriores, numerosas aplicaciones o toolbox más o menos gratuitas (y en función de ello su calidad), que participan en el estudio de la robótica.

Si nos centramos en robot manipuladores industriales, nos aparecen otros paquetes, que veremos en el siguiente apartado.

2.4.1. ENTORNOS DE SIMULACIÓN DE ROBOTS MANIPULADORES

En este apartado nos aparecen también dos grandes grupos de software, los desarrollados por firmas que provienen del ***CAD*** y los desarrollados por fabricantes de robots.

2.4.1. 1) *SOTFWARE DE COMPAÑIAS DE CAD*

Estas multinacionales del *CAD*, arrastran un excelente abanico de software de diseño e ingeniería, para la validación de prototipos mecánicos.

Todos ellas trabajan, desde hace años, en modo paramétrico, de forma, que los datos que se van incorporando al diseño, comparten relaciones con el resto (restricciones), y cualquier modificación posterior afecta y modifica al resto. Todos permiten el estudio mecánico y el ensamblaje de sus componentes, así como la simulación de su movimiento. Software de este tipo, es:

- ***Inventor*** [13]

  De la Norteamericana ***Autodesk***, propietaria también del pionero (desde los 90) y archiconocido ***AutoCAD.***

  Se utiliza para diseñar sobre *CAD 3D*, sistemas mecánicos tridimensionales, realizando estudios mecánicos y ensamblajes de los diferentes componentes. Permite añadir a los modelos, parámetros de diseño reales: propiedades del material (densidad, elasticidad, resistencia), velocidad, fuerza, etc. Con esta información se diseñan el ensamblaje de sus componentes, definiendo la unión o la transmisión del movimiento).

  Incorpora módulos de análisis mecánico (análisis de tensiones y deformaciones, resistencia, etc.), análisis cinemático y dinámico de los componentes.

  Como conclusión, se puede decir que ***Inventor*** es una perfecta herramienta de diseño mecánico de prototipos, ya sean robot o máquinas, pero no específica para de control y simulación de robot manipuladores.

- ***RobotExpert*** Figura 3 [14]

  De la multinacional alemana ***Siemens PLM software***, que engloba un extenso abanico de software para la industria y fabricación, incluyendo el conocido *CAD 3D,* ***SolidEdge***.

  Permite el diseño, la simulación y la optimización de procesos robóticos, incluyendo la programación off-line. Es compatible con robots de una amplia gama de marcas (*ABB*, *Fanuc*, *Kuka*, etc.) Simula operaciones de manipulación, procesos de mecanizado, soldadura con arco y corte con láser, incluso con varios robots en procesos en células de fabricación. Utiliza un entorno gráfico *3D* muy realista e intuitivo, permitiendo optimizar rutas de robots y mejorar el tiempo de los ciclos (diagramas de *Gantt*).

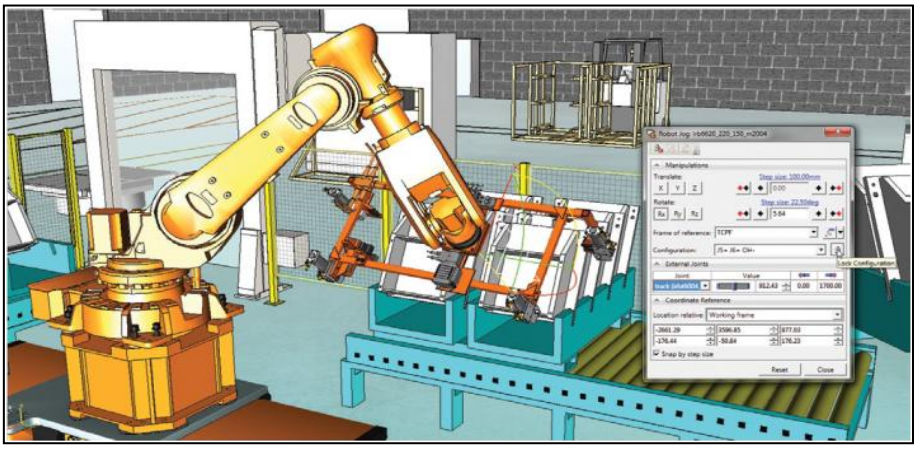


Figura 3. Simulación con *RobotExpert* de *Siemens PLM*

RobotExpert es una excelente herramienta de trabajo profesional para el control y simulación de robot manipuladores en procesos de fabricación robotizados. Es tan bueno, cómo el software de los fabricantes de robot.

- ***RobotWorks*** Figura 4 [15]

  Desarrollado por la multinacional francesa ***Dassault Systemes***, que engloba también un extenso abanico de software, aunque este, para el diseño mecánico, análisis y simulación de productos. Es propietaria también de ***SolidWorks*** y ***CATIA,*** este último, pionero por los 90, del *CAD 3D* paramétrico.

  RobotWorks se basa en el entorno gráfico de ***SolidWorks***, sobre el que trabaja y simula el control de robot manipuladores aunque admite todo tipo de prototipos móviles

  Con menos funcionalidad que ***RobotExpert***, soporta una amplia gama de robots, y es capaz de procesar su software nativo, como el anterior. Su ventaja está en el software de diseño mecánico que le rodea (a la altura de ***Ansys***), muy por encima de ***Inventor*** y ***RobotExpert.***

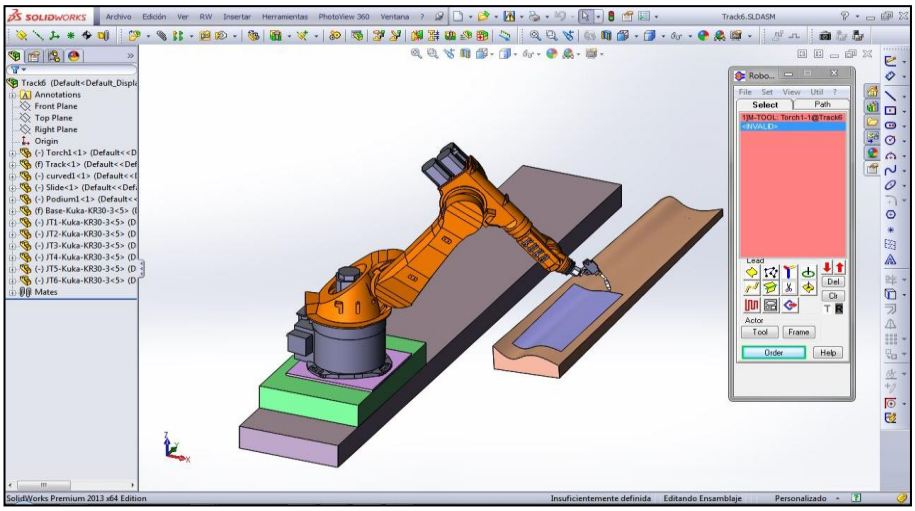


Figura 4. Simulación de robot Kuka con *RobotWorks*

2.4.1. 2) *SOFTWARE DE FABRICANTES DE ROBOTS*

El software que desarrollan y suministran los propios fabricantes de robots, que por supuesto, sólo admiten su gama. Lo pueden suministrar en versiones profesionales (de alta funcionabilidad) o docentes (limitan el tamaño y funciones).

Todos facilitan a través de sus webs, sus modelos en *CAD 3D,* con mayor o menor resolución: ***STL***, ***3DS***, ***VRML1***, ***DXF***, ***IGES***, ***GoogleSketchUp, SLDASM***, etc. Digamos, que ellos mismos son usuarios de *CAD 3D*, para realizar el diseño y análisis de sus prototipos.

Como en los anteriores paquetes, el entorno gráfico es un *3D* muy realista, permiten la simulación off-line, de uno o varios robots, operaciones en estaciones de trabajo, optimizan trayectorias, planificación de tareas, control de tiempos de ciclo, detectan colisiones, etc. Pero además, con dicho software se accede a los parámetros operacionales del robot, en tiempo real, con el que podríamos, entre otras opciones, calibrar el robot como si tuviéramos una botonera (*teach-pendant*).

Las ventajas con los anteriores son evidentes, en cuanto al conocimiento de los componentes y parámetros físicos que afectan a sus robot (masas, velocidades, momentos de inercia, etc.)

La diferencia entre los distintos paquetes de fabricantes, es que cada uno utiliza su propio lenguaje de programación de robots, más o menos abierto y compresible.

Todos, son productos de alta calidad, ampliamente testados, no olvidemos que son la herramienta con la que se ponen en marcha y mantienen, los robot en los procesos de fabricación. Paquetes de este tipo, son:

- ***KUKA.Sim*** [16], de la alemana ***Kuka***.
- ***RobotGuide*** °, de la japonesa ***FANUC***.
- ***RobotStudio***: de la suiza ***ABB***.

Por lo que nos afecta y por la similitud con el resto, analizaremos este último.

- ***RobotStudio*** Figura 5 [18]:

  Es un paquete comercial, que trabaja sobre *PCs* en *Windows*, de aspecto similar a cualquier producto gráfico de dicho entorno.

  Permite trabajar con toda la gama de robots y controladores industriales de ***ABB***. Se necesitan unos principios básicos de robótica para iniciarse en él y sacarle partido.

  El entorno gráfico *3D,* es muy realista, trabaja sobre ***CATIA***, lo cual permite incorporar geometría propia de ese *CAD*, aunque además, posea una amplia biblioteca de componentes y herramientas, que podemos incorporar a nuestro entorno robótico: vallas, soportes, *tools*, etc. Permitiendo la creación automática de cualquier tipo de estación.

El simulador ***RobotStudio*** funciona sobre ***RobotWare*** (software tipo "*driver" que se instala previamente),* que contiene los programas, funciones, configuraciones y datos, necesarios para el control del sistema robótico virtual y real (robot, controlador y *teach-pendant).* Posibilitando, la programación y simulación cinemática de las estaciones virtuales y reales.

La configuración del robot, la realizar en base:

- **Objetivos**: puntos (posición, orientación y forma) a los tiene que acceder el *TCP* del robot
- **Trayectorias**: secuencia de instrucciones de movimiento hacia los objetivos.
- **Sistemas de coordenadas**: diferentes en función de las necesidades de la tarea.
- **Configuración ejes del robot:** Especifica la forma en la que el robot accede a un objetivo (brazo, codo o muñeca, arriba o abajo).

Las instrucciones del robot se producen en lenguaje ***RAPID*** (1994), (***R**obotics **A**pplication **P**rogramming **I**nteractive **D**ialogue*) [19]. Es un lenguaje de programación estructurado de alto nivel, abierto, similar al Basic, Pascal o C. Es relativamente fácil de utilizar (con conocimientos previos de programación), que contiene las instrucciones que ejecuta el robot.

RobotStudio incorpora un editor del código ***RAPID***, de forma que podremos manipular, importar o exportar, las instrucciones que lleva implícitas.

Al igual que el controlador real, el simulador posee un panel de control para activar/desactivar el sistema robótico, así como *teach-pendant* virtual.

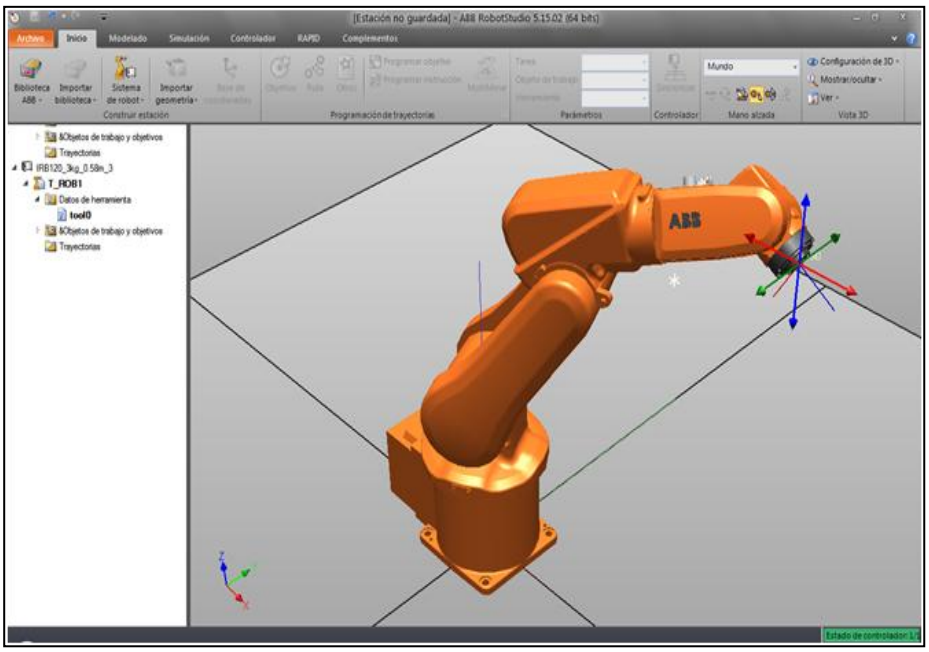


Figura 5. Robot *ABB IRB120* en *RobotStudio*

2.4.2. ENTORNOS DE SIMULACIÓN BASADOS EN *MATLAB*

Matlab es un producto de la firma norteamericana ***MathWorks***, líder en desarrollo de software de cálculo matemático para ingenieros y científicos, fundada en 1984. Es una potentísima herramienta matemática para el tratamiento de matrices, y por tanto ideal para el estudio de la robótica, ya que las matrices son la base para el estudio de los movimientos espaciales de los robots en *3D*.

Las aplicaciones que hemos encontrado en la red sobre ***Matlab***, proceden de entornos académicos, trabajan sobre robot manipuladores industriales y todas ellas son gratuitas.

- ***RoKiSim*** Figura 6 [20]

  Desarrollado por la *Escuela de Tecnología Superior de Montreal* (Canadá). Es un entorno gráfico de simulación, muy potente e intuitivo, que facilita la compresión del comportamiento cinemático del robot (articulaciones, movimientos, singularidades, ejes de DH, sistemas de coordenadas, trayectorias, etc.)

  Lleva incorporado una amplia biblioteca de robots industriales, muy conocidos, de 6 GdL. Permite incorporar otros elementos desde su biblioteca.

  Como dato anecdótico permite la simulación del robot mediante una ***wii***.

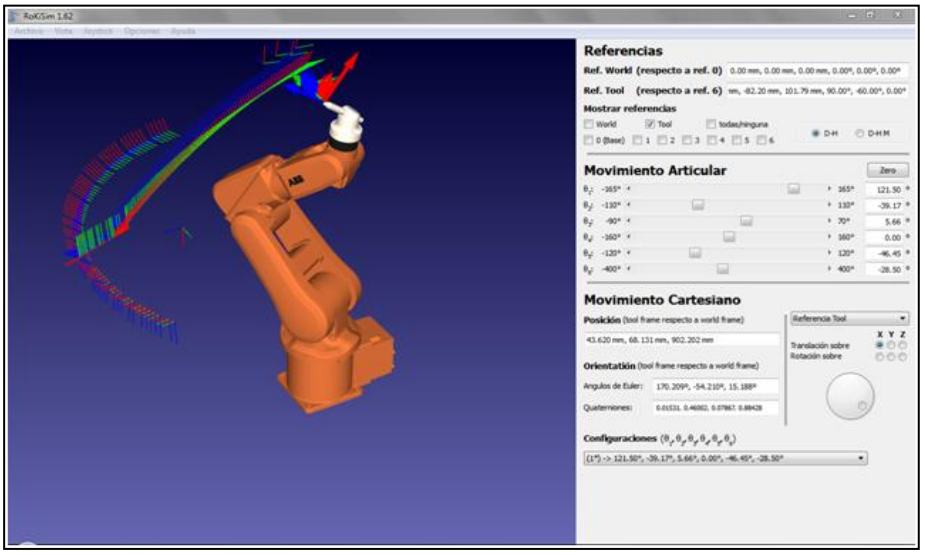


Figura 6. Robot IRB120 en el entorno gráfico *RokiSim*

- ***ARTE*** Figura 7 [21]

  "***A Robotic Toolbox for Education***" es una aplicación desarrollada en la ***Universidad Miguel Hernández*** de Elche (Alicante), por ***Arturo Gil***. Su departamento de robótica (***ARVC***) ofrece en la red [22], una excelente plataforma de estudio de la robótica, desde su principios teóricos, hasta su

implementación en *Matlab*, *RobotStudio* y *Rapid.* Aporta una magnifica documentación y una extensa colección de videos propios.

Aunque ***ARTE***, es mucho más que un entorno gráfico, el interfaz que suministra es similar al anterior, en cuanto a comprensión cinemática del robot y variedad de robots industriales que contiene.

Contiene un editor capaz de procesar el código *Rapid*, de los robots *ABB*, para hacer mover el robot virtual. Incluso, incorpora un traductor de código *Matlab* a *Rapid,* todo un alarde de dominio de la interpretación entre diferentes lenguajes de programación de robots.

Todo ello desarrollado sobre el entorno *Matlab,* desde el cual, ha desarrollando funciones, programas y ficheros, que desgranan el control y la simulación de robots manipuladores.

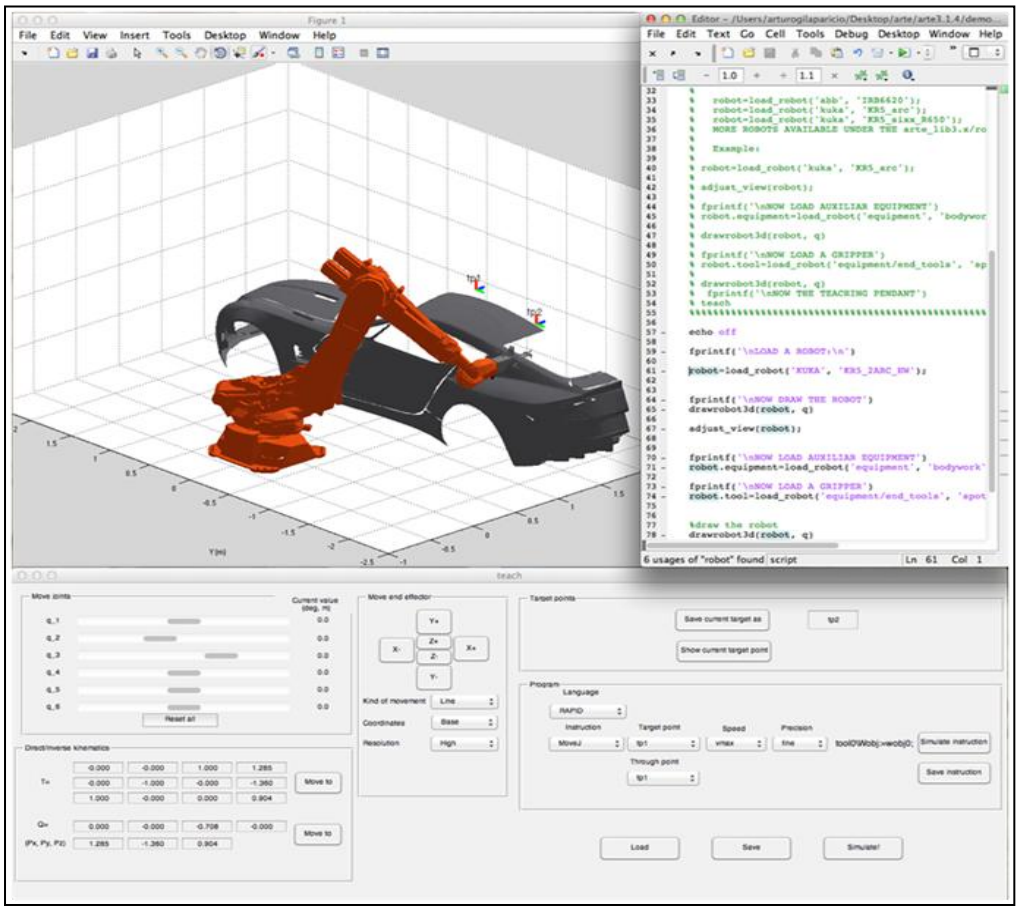


Figura 7. Interfaz de *ARTE*

- ***HEMERO*** [23]

  Es una ***HE****rramienta de* ***M****atlab para el* ***E****studio de* ***Ro****bots*, creada por ***Aníbal Ollero*** y concebida como apoyo al aprendizaje de diversos aspectos de la robótica. Proporciona gran cantidad de funciones y bloques para trabajar con la cinemática y dinámica del robot, desde ***Simulink,*** que es el entorno específico de simulación *Matlab*.

2.4.3. ROBOTIC TOOLBOX de *MATLAB*

Llegados aquí, cabe preguntarnos que nos ha hecho elegir la biblioteca ***Robotic toolbox (r.9.8),*** de ***Corke***, como herramienta de estudio de la robótica:

- Provee gran cantidad de funciones, datos y ficheros, con principios propios de la robótica, fácilmente accesibles y acompañada de una excelente información al respecto.
- Facilita la manipulación de datos en forma de vectores, transformaciones homogéneas y cuaternios (necesarios para representar la posición y orientación en el espacio).
- Se pueden aplicar a cualquier tipo robot sin más que conocer sus parámetros cinemáticos y dinámicos. Con dichos parámetros formaremos matrices que proporcionan una forma concisa de describir un robot y constituyen un medio fácil de comparar modelos y compartir información con otros usuarios.
- Posibilita la representación gráfica de la configuración del robot en un entorno *3D*, sencillo y muy intuitivo.
- Permite la interacción con el objeto gráfico (robot), mediante una botonera (*teach)*, mostrando sus múltiples posiciones y describiendo diversas trayectorias.
- El *toolbox* está en continuo proceso de mejora. Su autor, ***Peter Corke***, lo actualiza permanentemente, a menudo, en base a las sugerencias que recibe de otros investigadores o usuarios a través de su web [2], de la que se puede obtener información de primera mano.
- Tanto ***Matlab***, como ***Robotic toolbox***, son paquetes ampliamente conocidos en el mundo académico y científico.
- ***Matlab***, además, incorpora otras librerías capaces de mejorar y ampliar, otros aspectos del estudio (dinámica, sistema de control, gráficos *3D*, etc.), como ***Simulink*** [3], ***Simmechanic*** [5], ***V_Realm Builer*** [29].

CAPÍTULO III

3. MODELADO Y SIMULACIÓN DEL ROBOT

3.1. DESCRIPCIÓN DEL ROBOT IRB 120 de ABB

El robot ***IRB 120*** es el más pequeño de la multinacional suiza ***ABB***. Es un robot manipulador, articulado, antropomórfico tipo serie, con 6 ejes rotacionales, de tamaño y peso reducido (25 Kg), que ofrece una gran amplitud y capacidad de carga (580 mm y 3 Kg) y que además cuenta con todas las funciones de los grandes robots de ***ABB***. La información técnica del producto se ha obtenido de la web de ***ABB*** [24].

La suavidad y la precisión de sus movimientos se logran gracias a la utilización del nuevo y potente controlador ***IRC 5 Compact***, que contiene el hardware y el software necesario para su control y comunicación con el conjunto. La Figura 8, a la izquierda muestra las dimensiones del robot en su posición de reposo, y la derecha una imagen del conjunto robótico: robot, controlador y botonera (*teach-pendant*).

Admite diversas posibilidades de montaje: suelo, plataforma, pared e incluso inverso. Si a esta flexibilidad tan enorme, se une la compacidad, (muñeca pequeña y cableado interior), el robot resulta perfecto para aplicaciones en las que el espacio disponible sea un condicionante. También su hermeticidad y su fácil limpieza, lo hacen también apto para salas limpias.

Consecuencia de todo lo anterior, este robot es ideal para la docencia y aprendizaje. Recordemos que aparecía en la comparativa de robots educaciones del punto 2.3. [11].

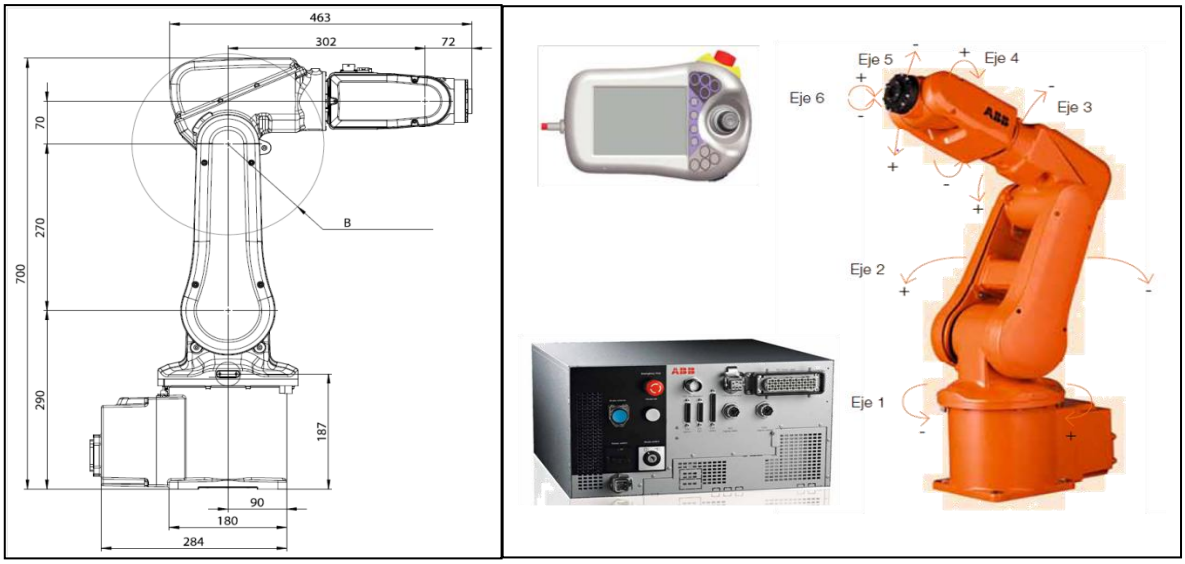


Figura 8. Dimensiones del robot y sistema robótico irb120

3.2. MODELADO Y SIMULACION DEL ROBOT EN *MATLAB*

Modelar numéricamente el robot requiere comprender todas las relaciones geométricas, físicas y matemáticas, que describen su comportamiento en el espacio. Siguiendo la excepcional documentación del libro "***Fundamentos de Robótica***" de ***Barrientos*** [25], definiremos el modelo numérico del robot, de forma que contenga todos sus parámetros cinemáticos y dinámicos.

3.2.1. CINEMÁTICA DEL ROBOT

La cinemática del robot estudia las relaciones espaciales de sus movimientos sin considerar las fuerzas que lo causan. Un robot manipulador puede considerarse una cadena cinemática abierta, formada por objetos rígidos o eslabones, unidos entre sí mediante articulaciones o ejes, que permiten alcanzar a su extremo (*TCP*), una determinada posición y orientación.

El problema cinemático del robot consiste en determinar las relaciones entre la posición y orientación del extremo del robot -*TCP*- (matriz T), y las variables articulares que adopta cada articulación (vector Q). Si conocemos estas últimas, determinar el *TCP*, es conocido como problema cinemático directo y lo contrario, se denomina problema cinemático inverso (Figura 9).

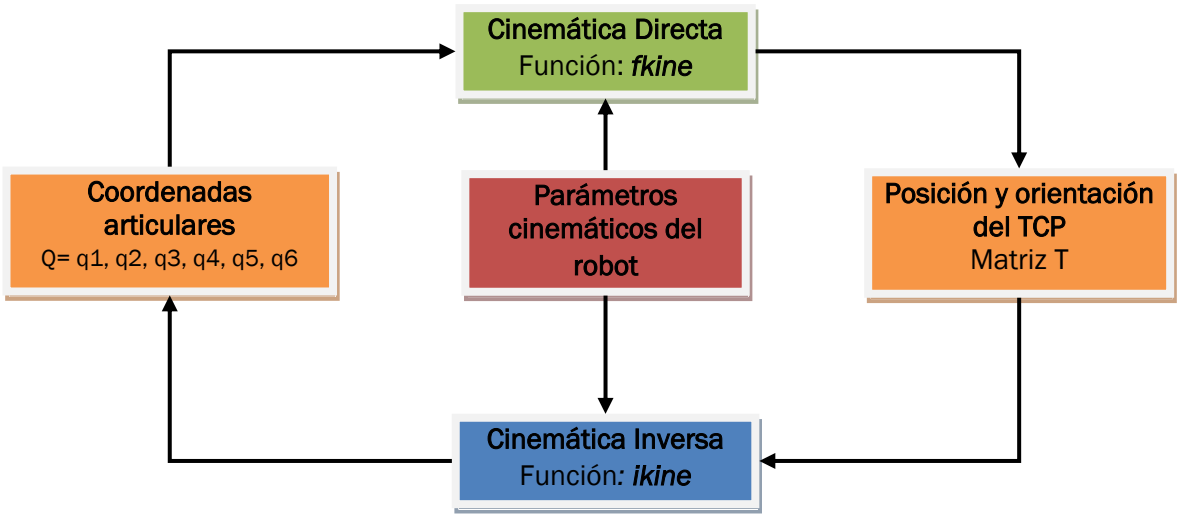


Figura 9. Esquema obtención parámetros cinemáticos del robot

Ambas soluciones pasan por la utilización del algebra matricial a los objetos matemáticos del espacio en *3D*, (puntos, vectores y rotaciones), en coordenadas homogéneas. Con la utilización de matrices de transformación homogéneas de 4x4 (que llamaremos matriz ***T***), podemos pasar los objetos matemáticos de un sistema de referencia a otro, aplicando una traslación y una rotación.

La matriz resultante tiene el aspecto de la Figura 10.Donde **n**, **s** y **a**, son los tres vectores unitarios que definen la rotación del segundo sistema de referencia respecto del primero, ***p*** es un vector de traslación, ***f*** representa los valores de la perspectiva (en nuestro caso siempre cero) y finalmente el escalado, es un número de factor de escala, en nuestro caso, siempre es 1.

$$T = \begin{bmatrix} \text{matriz de rotación} & \text{vector de posición} \\ f_{1x3} & \text{escalado} \end{bmatrix} = \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 10. Forma de la matriz homogénea

Una matriz de transformación homogénea, de la forma ${}^{i-1}A_i$, representa el paso de coordenadas desde el sistema **i-1** al sistema **i.** De forma, que un punto P_1, en coordenadas del sistema ***1***, tendrá la siguiente expresión en coordenadas del sistema **0**:

$$P^0 = {}^0A_1 * P_1$$

Aplicado al robot, y estableciendo el sistema de referencia fijo, situado en la base del robot, se describe la localización de cada uno de los eslabones con respecto a dicho sistema de referencia, con una matriz de la forma ${}^{i-1}A_i$. Resultando que la matriz de transformación homogénea que define el paso de las coordenadas de la base del robot al extremo del mismo, sería:

$$T = {}^0A_6 = {}^0A_1 * {}^1A_2 * {}^2A_3 * {}^3A_4 * {}^4A_5 * {}^5A_6$$

Donde cada matriz ${}^{i-1}A_i$ representa dos articulaciones consecutivas unidas por un eslabón. Permitiendo definir el cambio entre sistemas de referencia de **i-1** y al sistema **i**.

Una recurso ampliamente utilizado en robótica, para definir la matriz de Transformación Homogénea del robot, es el algoritmo de ***Denavit-Hartenverg*** [27].

3.2.2. ALGORITMO DE DENAVIT-HARTENBERG

El algoritmo de ***Denavit-Hartenberg*** (***D-H***), es un recurso para obtener de manera sistemática la matriz de transformación homogénea **T**, que define la geometría cinemática del robot. Los 4 valores a obtener dependen únicamente de las características geométricas de cada eslabón y de la articulación que lo une con el anterior. Para cada eslabón **i**, estos serían (Figura 11):

- θ_i: ángulo de la rotación, en torno al eje Z_{i-1}, del eje X_i, medido en un plano perpendicular al eje Z_{i-1}. Dicho ángulo es positivo, aplicando la regla de la mano derecha (*rmd*).

- **d_i**: distancia a lo largo del eje **Z_{i-1}**, desde el origen del sistema de coordenadas **i-1**, hasta la intersección con el eje **Xi.**
- **a_i**: distancia a lo largo del eje **X_i**, que va desde la intersección con el **Z_{i-1}**, hasta el origen del sistema **i**. El signo lo define el sentido de **X_i.**
- **α_i**: ángulo de rotación, en torno al eje **X_i**, del eje **Z_i**, medido en un plano perpendicular al eje **X_i**. El sentido positivo sale al aplicar la *rmd*.

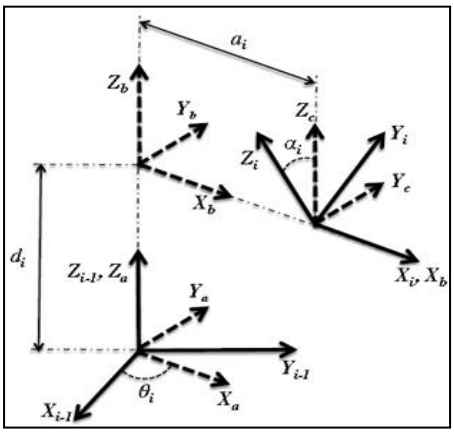


Figura 11. Valores de D-H

Típicamente, uno de dichos parámetros lo constituirá la **variable articular** que conecta ambos eslabones. La matriz resultante para cada eslabón se obtiene al sustituir los 4 valores en la matriz de la Figura 12.

$$ {}^{i-1}A_i = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Figura 12. Matriz de transformación de cada eslabón

La Figura 13 muestra los eslabones y los ejes del robot numerados, según requiere el algoritmo *D-H*, en un gráfico *CAD 3D* (a la izquierda), y en el de *Matlab*, más conceptual a la derecha.

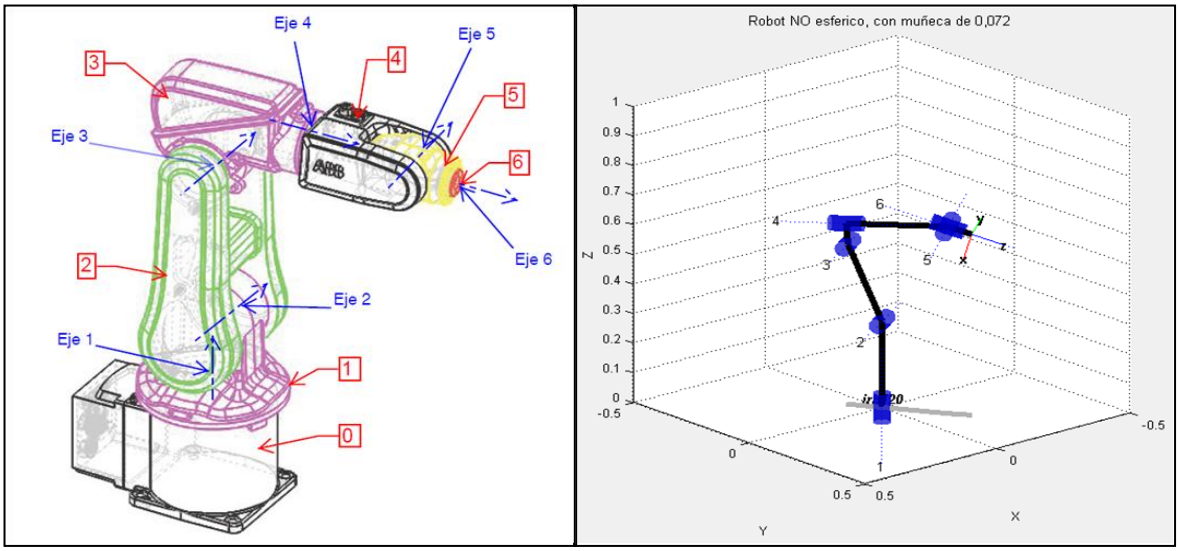


Figura 13. Estructura del robot *3D* y conceptual en un gráfico de *Matlab*

La Figura 14 muestra la posición de los ejes y las dimensiones precisas para obtener la Tabla 1 , que recoge los parámetros D-H obtenidos. En la última columna figuran los valores máximos de la variable articular.

Eslabón	θ_i	d_i	a_i	α_i	Rango (θ_i)
1	θ_1	290	0	-90º	+165 - 165º
2	θ_2-90º	0	270	0º	+110º -110º
3	θ_3	0	70	-90º	+70º - 110º
4	θ_4	302	0	90º	+160º - 160º
5	θ_5	0	0	-90º	+120º -120º
6	θ_6+180º	72	0	0º	+400º - 400º

Tabla 1. Tabla de valores Denavit-Hartenberg del IRB 120

El proceso seguido para conseguir la Figura 14 y la Tabla 1, partiendo de la geometria del robot, ha sido el siguiente:

» Desde su posición inicial, se van colocando los ejes desde la base y aritulación tras articulación, se enumeran los eslabores, asignando el **0** a la base, y **n-1** para el último, siendo **n** el número de **GdL** o articulaciones.

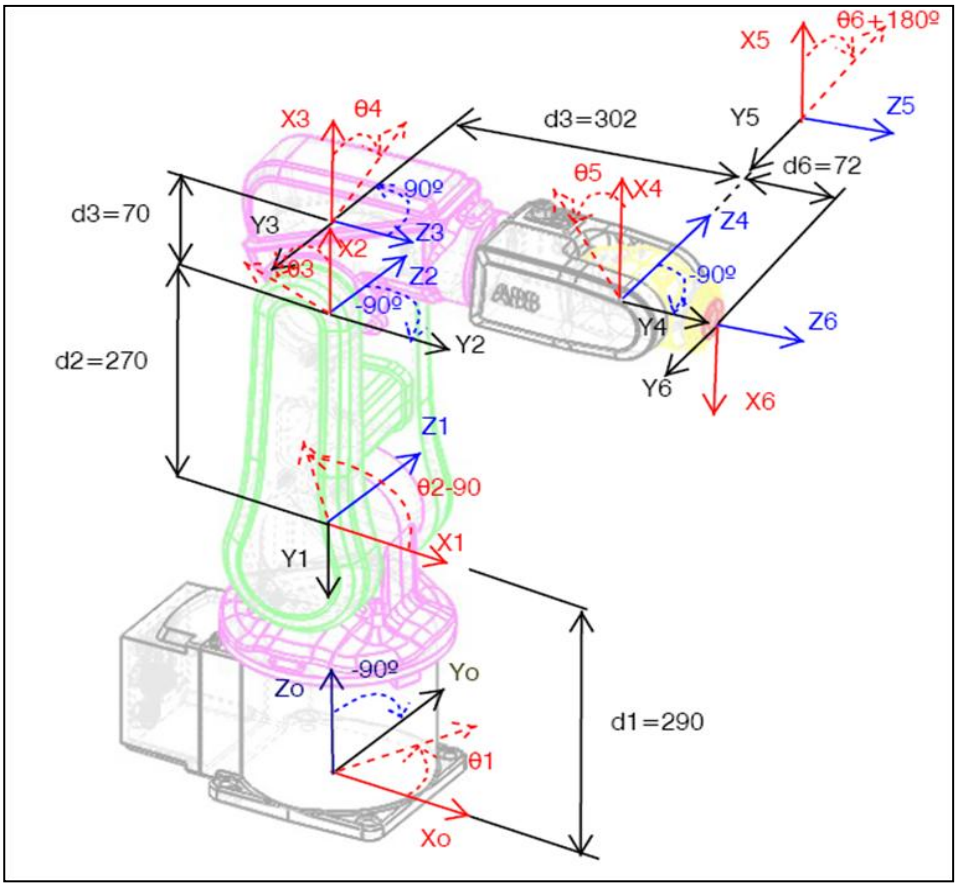


Figura 14. Descripción de ejes y parámetros *D-H* en el Robot

- El sistema de coordenadas en la base, se establece con el eje **Zo**, localizado a lo largo del eje del movimiento de la articulación 1 y apuntando hacia fuera del hombro del robot.
- El sistema de referencia, de cada eslabón, se coloca al final del mismo, en el extremo de la articulación a la cual esta concectado. El eje **Z** debe quedar alineado con el eje.
- El angulo o desplazamiento de cada eslabón, se mide tomando como base el sistema de referencia del eslabón anterior.
- Teniendo, además, en cuenta que los ejes:
  - **Z**: son ejes de coordenadas se asignan a los ejes de las articulaciones.
  - **X**: se asignan a las normales comunes A_i y A_{i+1} .

Con los valores de la Tabla 1 y la función ***SerialLink*** del *toolbox*, iniciamos el proceso de modelado del robot en nuestra función ***mdl_irb120***. En ella, y en el transcurso del estudio, se han ido añadiendo el resto de parámetros físicos que definen su dinámica, quedando abierta a posibles modificaciones y variaciones del robot. Podríamos también, si quisiéramos, modelar un nuevo robot, de similares características a este.

Aprovechando dicha funcionalidad, se ha introducido en el interfaz, diversas configuraciones del robot:

- Con y sin base-soporte: suplemento sobre el que se apoya el robot.
- Con y sin herramienta: puntero de 0,1 m, colocado en el último eslabón.
- Muñeca esférica o no, que hacen al robot esférico.

El resultado de estos datos iniciales del robot, se puede ver, en una ventana opcional, del interfaz gráfico, tal y como muestra la Figura 15. También, y como opción, se pueden ver los ejes del robot definidos con el algoritmo de D-H (Figura 16).

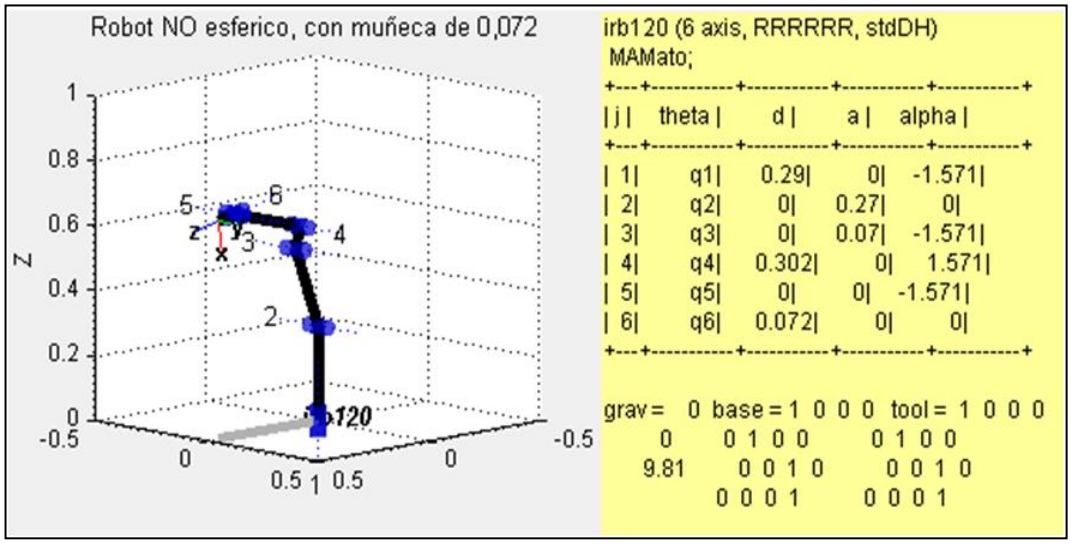


Figura 15. Parámetros iniciales y básicos del robot irb120

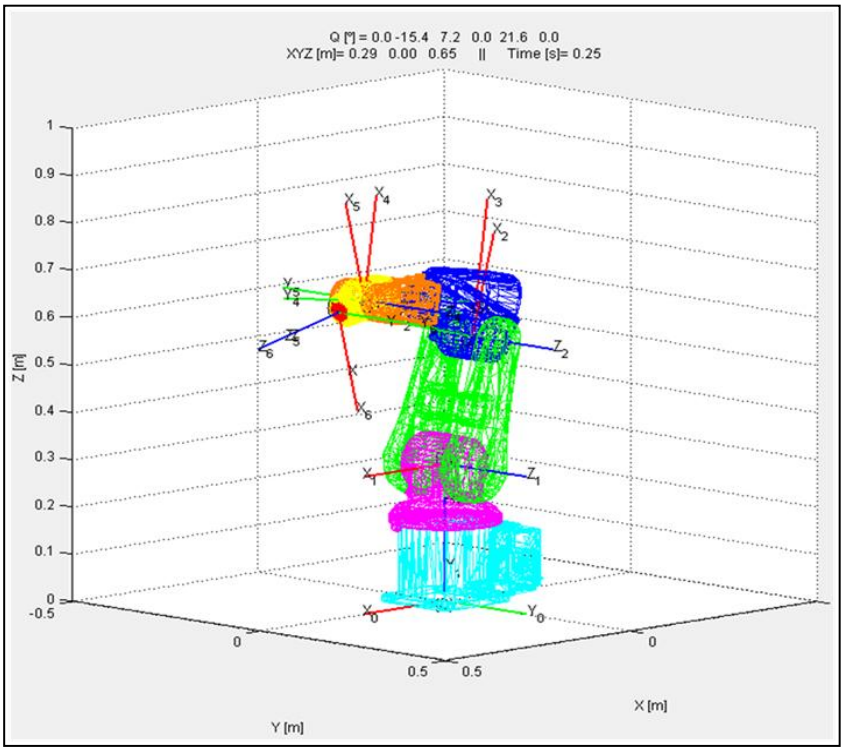


Figura 16. Ejes del robot *D-H* desde la aplicación desarrollada

Podremos ver también, las matrices de Transformación Homogéneas, del robot, y las de cada eslabón, desde una ventana opcional (Figura 17).

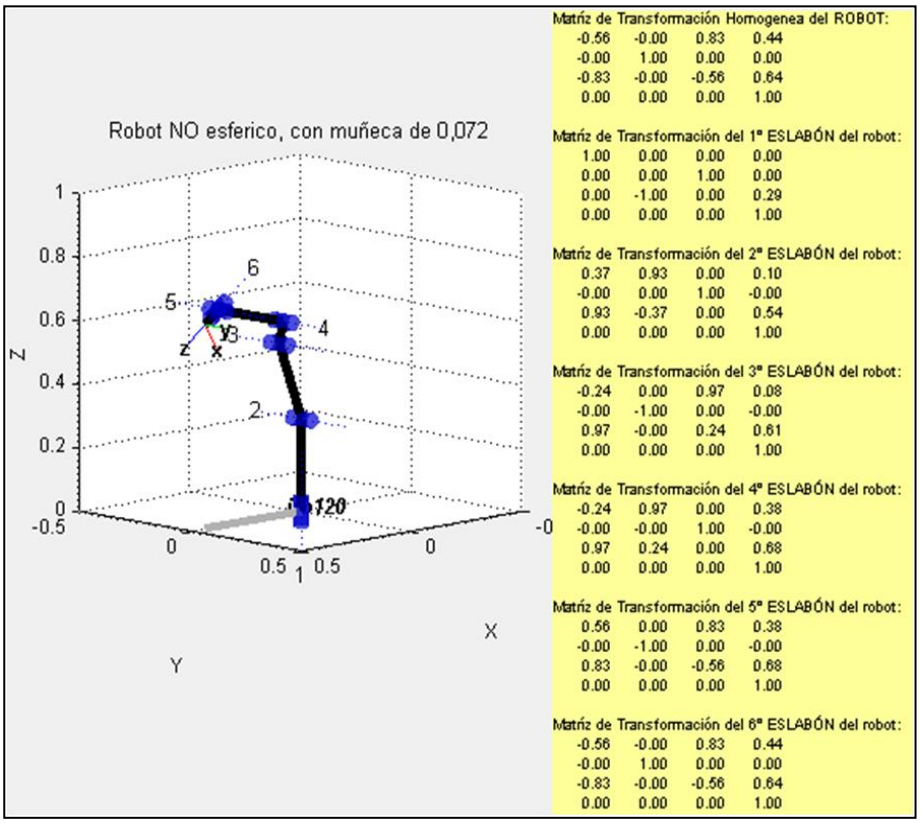


Figura 17. Matrices de transformación homogéneas del

3.2.3. RESOLUCIÓN DEL PROBLEMA CINEMÁTICO

Cómo habíamos dicho, el problema cinemático del robot consiste en determinar las relaciones entre la posición y orientación del extremo del robot -*TCP*- (matriz T), y las variables articulares que adopta cada articulación (vector Q), y para ello habíamos ilustrado la Figura 9.

La cinemática directa, no presenta ningún problema, conocido el valor de las 6 variables articulares, para cada instante de tiempo Q(t)= q1, q2, q3, q4, q5, q6; conoceremos la posición y orientación del *TCP* (T), con la función ***fkine*** del *toolbox*.

Contrariamente a la sencillez de la anterior, la cinemática inversa, presenta una considerable dificultad, ya que para cada posición y orientación del *TCP* en el espacio, aparecen para este robot de 6 *GdL*, múltiples configuraciones posibles, y no todas alcanzables. Para resolver este problema, nuestro *toolbox* tiene dos funciones ***ikine*** e ***ikine6s***:

- La primera, ***ikine,*** es una resolución matemática por iteración, computacionalmente lenta, no siempre muy precisa en la solución, y que además, converge a una única solución dependiendo del punto partida. Dicho de otra forma, funciona correctamente cuando el punto desde el que se parte está cercano al que tiene que ir, y este, lejano a puntos singulares (puntos que no puede acceder el robot).
- La segunda función, ***ikine6s***, de resolución geométrica, no funciona con nuestro robot. Con el robot de referencia del *toolbox* (***Puma560***) funciona correctamente, ya que su muñeca es esférica, y en nuestro caso, no lo es, es decir, el mecanismo de posicionamiento de la muñeca, no converge en el mismo punto.

Para salvar el anterior inconveniente desarrollamos una función basada en optimización local, que aunque efectiva en encontrar el vector ***Q***, resultaba computacionalmente lenta. Función ***ikine_nueva***.

Llegados a este punto, y con el objetivo de ganar velocidad en la aplicación gráfica, recurrimos a una librería ajena a la nuestra y ya mencionada, ***ARTE*** [21], para adoptar su código en la resolución de la cinemática inversa.

La solución, en este caso, se consigue por métodos geométricos, descomponiendo la cadena cinemática del robot en varios planos geométricos para luego resolver el sub-problema geométrico asociado a cada plano mediante relaciones trigonométricas. De las 8 soluciones que aporta, para cada posición definida por una matriz T, no todas son válidas, ya que ciertos valores sobrepasan los límites de la articulación. A la función la hemos dado el nombre ***ikine_irb120***.

Cualquiera de las tres funciones se puede elegir como algoritmo de resolución de la cinemática inversa en el interfaz.

3.2.4. DINÁMICA DEL ROBOT

La dinámica estudia el movimiento de los cuerpos debido a fuerzas externas e internas de los mismos. El modelo dinámico relaciona matemáticamente la localización del robot a través de la velocidad y aceleración, mediante las fuerzas y pares aplicados por los actuadores en las articulaciones (Figura 18).

La dinámica directa encuentra la respuesta articular (posición, velocidades y aceleración) de cada eslabón del robot, en función de los pares y/o fuerzas que intervienen. La dinámica inversa encuentra los pares y/o fuerzas de los actuadores requeridos para generar una trayectoria deseada del sistema robótico. La resolución de ambas es un problema muy complejo, estudiado desde 1965, por no pocos investigadores, y con diferentes enfoques y algoritmos.

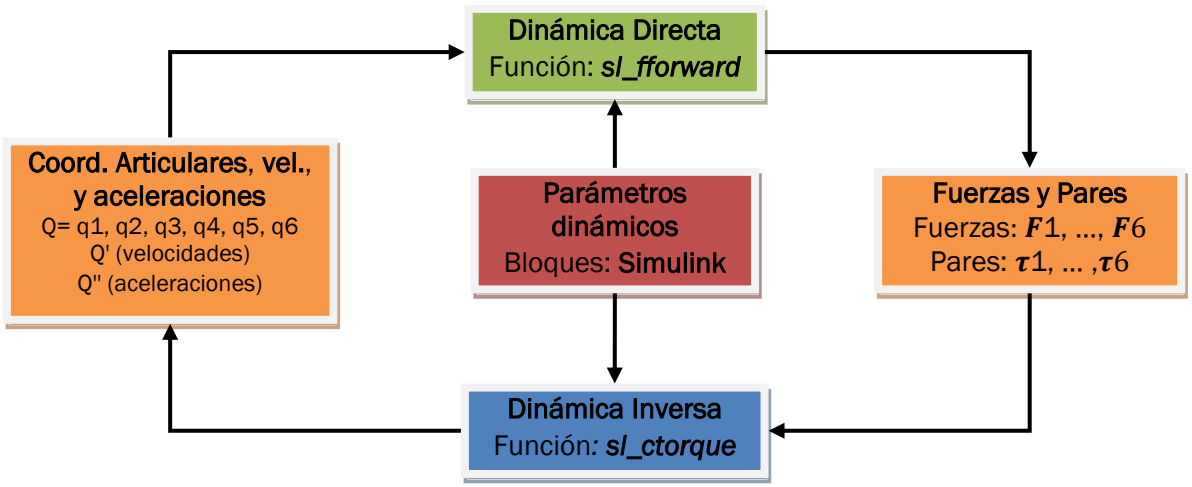


Figura 18. Esquema obtención parámetros dinámicos del robot

Siguiendo la documentación de ***Corke***, al respecto [28], la ecuación que rige el comportamiento dinámico del robot (ecuación de espacio de estados) es:

$$\tau = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + F(q) + G(q)$$

Donde:

- τ : es la matriz o tensor de fuerzas asociado a las coordenadas articulares.
- ***M***: es la matriz o tensor de inercia del robot.
- ***q***: es el vector de las coordenadas articulares del robot.
- ***C:*** Matriz que incluye las fuerza de Coriolis y centrífugas.
- $\dot{q} = \frac{dq}{dt}$, es el vector de velocidades de las articulaciones.
- $\ddot{q} = \frac{d^2q}{dt^2}$, es el vector de aceleraciones de las articulaciones.

- *F*: describe la fricción viscosa y de Coulomb de las articulaciones.
- *G:* es el vector de fuerzas gravitacionales.

Llegados a este punto, el estudio requiere incorporar a nuestro modelo numérico, los datos dinámicos del robot real: masas, centros de gravedad, momentos de inercia, etc. Para obtener estos datos de forma fiable, hay que recurrir al fabricante *ABB*, y este, no los suministra por motivos de confidencialidad del producto. Conseguirlos de forma teórica o experimental sobre el robot real, resulta imposible para el alcance de nuestro proyecto. Dichos datos dinámicos, se podrían conseguir, en la red, para modelos de referencia tipo *Puma560*, aún así, muchos son una aproximación (así lo considera también el propio *Corke*).

Para solventar este problema se ha recurrido a un modelo *CAD 3D*, de bastante calidad y resolución, que el propio fabricante *ABB* suministra [26], y después, desde el paquete *SolidWorks*, con un comando del mismo, obtener dichos parámetros dinámicos para cada eslabón del robot.

La Figura 19 muestra la obtención del cuarto eslabón del robot. Dicho proceso conlleva una mayúscula y errónea simplificación, que es la considerar los eslabones del *CAD* como un sólido de la misma densidad -aluminio (2.000 kg/m^3)-, siendo en realidad una carcasa metálica, más o menos hueca, que puede incorporar: bastidor, motores, reductoras, poleas, correas, cableado, etc. Resultando sus centros de gravedad y momentos de inercia, con toda seguridad bien distintos de los obtenidos.

En la Figura 20 se muestra, a la izquierda, la sección del segundo eslabón del robot en el modelo *CAD*, y a la derecha, el mismo eslabón, pero del robot real, apreciándose la notable diferencia entre ambos.

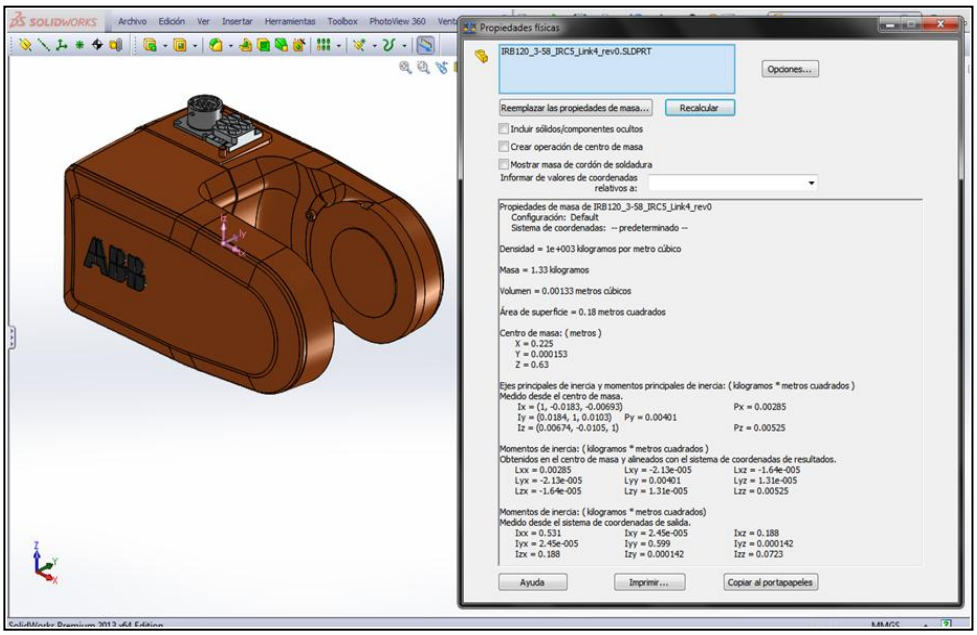


Figura 19. Extracción parámetros dinámicos del 4º eslabón en *SolidWorks*

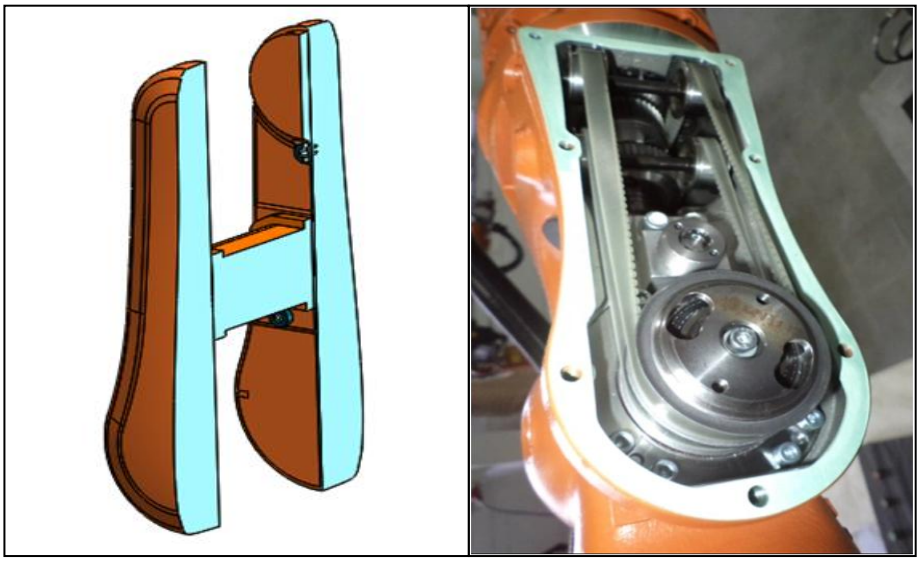


Figura 20. Segundo eslabón del robot, CAD 3D y robot real.

A pesar de dicha contrariedad, los datos obtenidos son perfectamente válidos, para nuestro estudio, ya que estos, a medida que se consigan más fiables, se pueden introducir en nuestra función del modelado del robot (***mdl_irb120***).

Con los datos obtenidos de cada eslabón, y los que teníamos del anterior apartado (D-H), definimos completamente todos los parámetros del robot (Tabla 2). Todos ellos, están contenidos en nuestra función ***mdl_irb120***, que se pueden ver como opción en nuestro interfaz Figura 21.

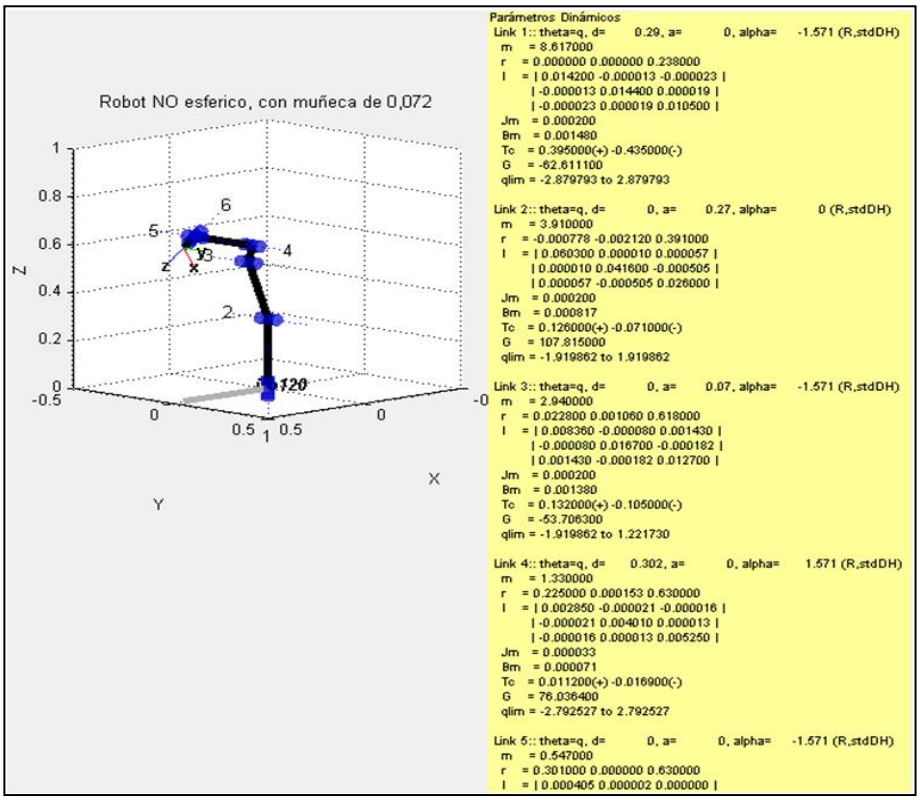


Figura 21. Todos los parámetros del robot en el interfaz

La Tabla 2 contiene todos los valores de la matriz de 20 filas, que define completamente un robot según el *Robotic Toolbox*. A esta matriz la hemos añadidos 2 filas que contienen los valores límites de las articulaciones.

Parámetros Matriz dyn				ELEMENTOS DEL ROBOT						
Col.	Simbolo	Uds	Descrip.	BASE	1	2	3	4	5	6
1	theta	rad	Parámetros de Denavit-Hartenberg	-	q1	q2-1,571	q3	q4	q5	q6+3,14
2	d	m		-	0,29	0	0	0,302	0	0,072
3	a	m		-	0	0,27	0,07	0	0	0
4	alpha	rad		-	-1,571	0	-1,571	1,571	-1,571	0
5	sigma	Tipo articulación		-	R	R	R	R	R	R
6	masa	kg	Masa	6,21	3,06	3,91	2,91	1,33	0,55	0,01
7	rx	m	Centro de gravedad	-0,04	0,00	0,00	0,02	0,23	0,30	0,00
8	ry	"		0,00	0,00	0,00	0,00	0,00	0,00	0,00
9	rz	"		0,08	0,24	0,39	0,62	0,63	0,00	0,10
10	Ixx	$kg*m^2*10^{-5}$	Inercia referido al centro de masas del eje	2,47	14,20	60,30	8,36	2,85	0,41	0,00
11	Iyy	"		4,91	14,40	41,60	16,70	4,01	0,00	0,00
12	Izz	"		4,72	10,50	26,00	12,70	5,25	0,82	0,00
13	Ixy	"		0,00	0,00	0,00	0,00	0,00	0,00	0,00
14	Ixz	"		0,13	0,00	0,00	1,43	0,00	0,00	0,00
15	Iyz	"		0,00	0,00	-0,51	1,43	5,25	0,00	0,00
16	Jm	Inercia motor-actuador		0	0	0	0	0	0	0
17	G	Ratio reductora		0	-62,61	107,82	-53,71	76,04	71,92	76,69
18	B	Fricc. Viscosa actuador		0	0,00	0,00	0,00	0,00	0,00	0,00
19	Tc+	Fricc. Coulomb +		0	0,40	0,13	0,13	0,01	0,01	0,00
20	Tc-	Fricc .Coulomb -		0	-0,44	-0,07	-0,11	0,02	0,01	-0,01
21	Qlim -	rad	Angulo Máx	0	-2,88	-1,92	-1,92	-2,79	-2,09	-6,98
22	Qlim +	rad	Angulo Min.	0	-2,88	1,92	1,22	2,79	2,09	6,98

Tabla 2. Parámetros D-H y dinámicos del robot

3.2.5. MATRIZ JACOBIANA E INDICE DE MANIPULABILIDAD

El algoritmo de control del robot puede establecer qué velocidad y aceleración debe fijar a cada articulación, de modo que se consiga que el extremo desarrolle una trayectoria temporal concreta. La relación entre ambos vectores se obtiene través de la denominada matriz Jacobiana. Esta matriz es sumamente importante en el análisis y control del movimiento de un robot ya que, entre otras cualidades, permite conocer el área de trabajo, destreza y puntos singulares del robot.

La información acerca de dicho comportamiento la podremos conocer mediante el valor del índice comportamiento cinemático (o índice de manipulabilidad), de la función ***SerialLink.maniplty,*** del *toolbox*, con sus dos algoritmos de resolución, ***Yoshikawa*** y ***Asada***. El interfaz informa de su valor, para cada posición alcanzada, y para el conjunto de puntos que definen una trayectoria.

La importancia de todo lo anterior, aunque sabida, *ABB* no la pudo desarrollar comercialmente en sus robots hasta el año 2005, en el que lanzó varias funciones basadas en el estudio de su dinámica, a efectos de mejorar su productividad [10]:

- ***QuickMove***, en la que se determina la máxima aceleración en cualquier movimiento y se utiliza al menos en un eje para alcanzar la posición final en el menor tiempo posible. Así se reduce al mínimo la duración del ciclo, que no depende únicamente de las velocidades de los ejes.
- ***TrueMove,*** es la desviación mínima respecto a la trayectoria programada. Esta función garantiza que la ruta de movimiento seguida es la misma con independencia de la velocidad y ahorra la necesidad de ajustar la trayectoria cuando los parámetros de velocidad se ajustan en línea.
- ***FlexFinishing,*** (control de fuerza de mecanización), muy útil para operaciones delicadas de mecanizado, en especial para rectificado, desbarbado y pulido de piezas de fundición.

3.2.6. MODELADO GRÁFICO DEL ROBOT *CAD 3D* EN *MATLAB*

La representación gráfica de los robots con el *Toolbox* de *Corke* es una representación sencilla y conceptual, perfectamente válida para las pretensiones del autor y para cualquiera que se acerque al estudio de los robots, pero un poco pobre comparada con los paquetes comerciales de simulación que ya hemos citado.

Si buscamos una representación gráfica más realista, tenemos que acudir a librerías o funciones del entorno *Matlab.* Siguiendo esta premisa, de la web de *Corke*, obtuvimos la información necesaria para el tratamiento gráfico del robot: mediante la función ***patch*** y el formato gráfico ***STL*** para los eslabones:

- Un ***patch*** es un objeto gráfico de *Matlab* compuesto por uno o más polígonos (con vértices, caras y aristas), que pueden estar o no conectados. Es utilizada para el modelado de sólidos *3D* en los ***axis*** (ventana gráfica de *Matlab*).

- Los archivos ***STL*** (acrónimo de "***ST****ereo* ***L****ithography")* es un formato para *CAD 3D* (de reciente aparición, 2010) muy utilizados en la actualidad para impresión *3D*. Define la geometría de los sólidos en forma de triángulos (con vértices, aristas y caras).

Los eslabones los obtenemos desde el paquete ***SolidWorks***, con el modelo del robot que ya teníamos (Figura 22-inferior izqda.-) descargado de ***ABB*** [26], y exportándoles al formato *STL -ASCII-* (Figura 22-superior izqda.-). Tendremos que especificar la posición del origen de los ejes de coordenadas de los eslabones (que serán los de D-H), y además la resolución de la conversión gráfica, en nuestro caso baja, para no ralentizar la aplicación.

La función desarrollada, ***plot_irb120***, contiene todo el código necesario para la representación del robot y sus complementos, en un ***axis***, con el modelo numérico que teníamos desarrollado en la función ***mdl_irb120***. Contiene además, todas las opciones de representación: transparencia, ejes, con o sin soporte, con o sin puntero, etc.

Los resultados, en términos visuales, aún en baja resolución, con respecto al modelo del toolbox, son espectaculares, como se puede ver en el acoplamiento de ambos de la Figura 23. En términos de tiempo de respuesta computacional, no resulta nada lento, sabedores que todo aquello que represente trabajar con sólidos *3D*, requiere mucho código y además mucha computación.

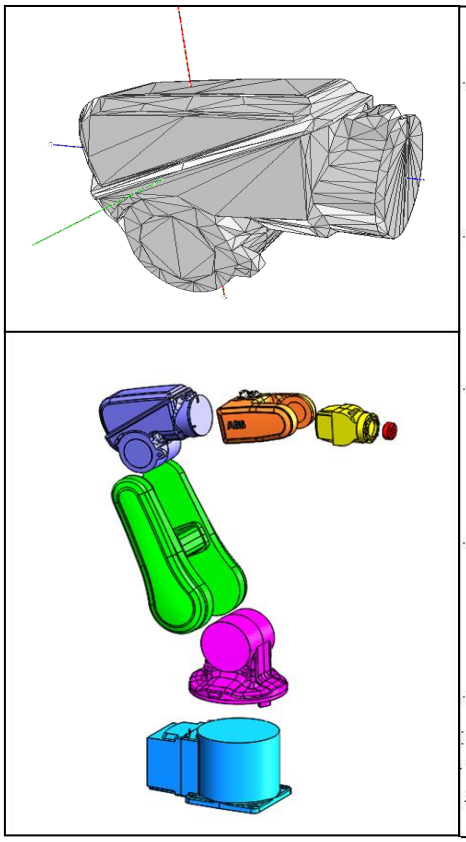


Figura 22. Modelo *CAD* SolidWorks

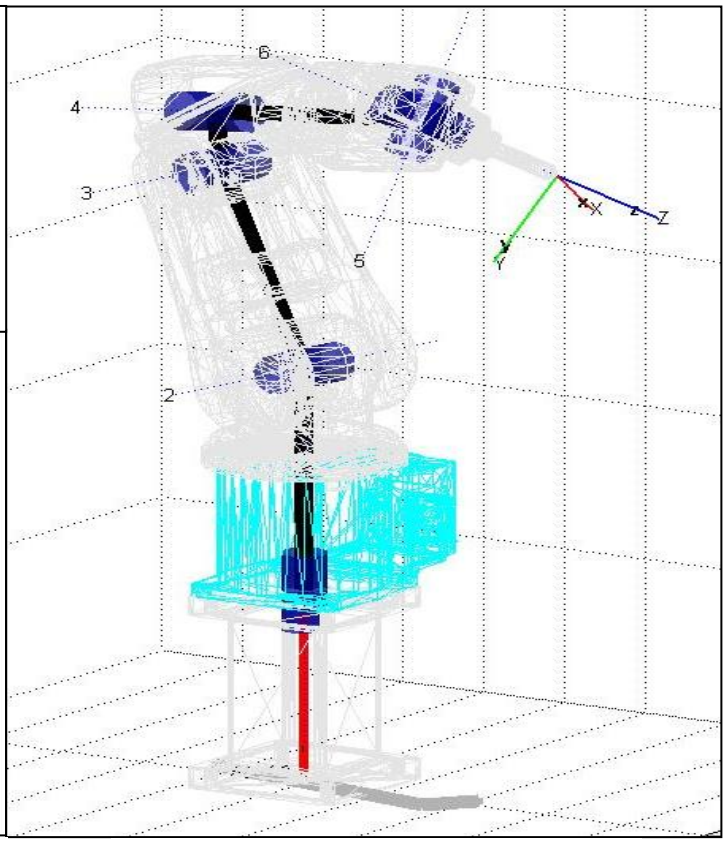


Figura 23. Robot 3D CAD con robot

3.2.7. FUNCIONES Y VARIABLES UTILIZADAS EN LA APLICACIÓN

Describimos a continuación todas aquellas funciones y variables, propias o del toolbox, que intervienen en la aplicación:

3.2.7. 1) *FUNCIONES DE ROBOTIC TOOLBOX (r 9.8)*

- **Relacionadas con la definición física del robot:**

revolute	Definición específica para cada eslabón (link) de un robot del tipo de revolución. Contiene todos los parámetros geométricos y físicos de cada eslabón descritos en la Tabla 2. Para el primer eslabón:
	`L(1) = Revolute('d', 0.29, 'a', 0, 'alpha', -pi/2,.. 'offset', 0,... % Desfase articulación`
	`'I', [0.0142, -1.29e-5, -2.31e-5; -1.29e-5, 0.0144, 1.93e-5;-2.31e-5, 1.93e-5, 0.01050],…`
	`'r', [0, 0, 0.238], ...%cdg`
	`'m', 8.617, ... % masa`
	`'Jm', 200e-6, ... %`
	`'G', -62.6111, ...`
	`'B', 1.48e-3, ...`
	`'Tc', [0.395 -0.435], ...`
	`'qlim', [-165 165]*rad ); % límites de la articulación`

serialLink	Define y contiene todos los parámetros del robot, en forma de *estructura* de *Matlab*. ***irb120***, es la estructura que contiene la definición completa del robot numérico. Podremos acceder a sus valores como si fuera una estructura anidada. `irb120=SerialLink(L,'name','irb120','manufacturer','ABB', 'comment','MAMato');`
.base	Añade una base al robot. En coordenadas homogéneas, un soporte de base de 0,25 m `irb120.base=transl(0,0,0.25);`
.tool	Añade una herramienta al *TCP*. Añadimos un puntero de 0,1 m. `irb120.tool=transl(0,0,0.1);`
.n	Número de eslabones del robot `irb120.n; % devuelve el número de eslabones del robot`
config	Tipo de articulación. 'RRRRRR' son todas de revolución.
mdh	Tipo de parámetros D-H. (0=DH, 1=MDH)
.char	Presenta todos los parámetros cinemáticos del robot. `dh=irb120.char; % string con todos los param. Cinemat.1`

- Para la definición gráfica del robot

plot	Representación gráfica del robot (*conceptual).* en un *axis* de *Matlab.* Contiene una extensa lista de parámetros que modifican el aspecto del robot. `irb120.plot(Q);% Representa las distintas posiciones de Q`
teach	Muestra una botonera (tipo slider) que interactúa con el robot del axis. El robot aparece en la posición q1, y valores en grados. `irb120.teach (q1, 'degrees'); % Botonera con robot en q1`
trplot	Dibuja marco de ejes de coordenadas (posición y orientación) `trplot(t, 'frame', 'P2', 'length',0.15,'width', 1 , 'rgb', 'thick', 1); %punto con posición y orientación`

- Para la cinemática del robot

fkine	Cinemática directa. Con las coordenadas articulares (Q), obtengo las coordenadas (posición y orientación) del *TCP*, en homog. (T). `irb120.fkine(q1); % Obtengo todos el valor de t1`
ikine	Cinemática inversa usando iteración. Con T, obtengo Q. Utiliza parámetros para mejorar la búsqueda. `irb120.ikine(t1); % Obtengo el valor de q1`
ikine6s	Cinemática inversa geométrica para robot de 6 GDL con muñeca esférica. `irb120.ikine(T); % Obtengo os los valores Q`
jtraj	Trazado de trayectoria en coordenadas articulares `Q_tray = jtraj(q1,q2,20); %Trayectoria en 20 partes`
ctraj	Trazado de trayectoria recta en coordenadas homogéneas `t = ctraj (t1, t2, 5); % Trayectoria recta entre t1 y t2, en 5 partes`
jacob0	Matriz Jacobiana con referencia a la base
jacobn	Matriz Jacobiana con referencia a *TCP*
isspherical	Mostrar si el robot tiene la muñeca esférica
ishomog	Mostrar si el punto es un punto en coord. homogéneas
maniplty	Índice de manipulabilidad en q o Q, permite dos opciones. `valor=irb120.maniplty(q1,metodo);`

- Para la dinámica del robot

accel	Aceleración de la articulación
coriolis	Fuerza de Coriolis en una articulación
dyn	Muestra los valores dinámicos del robot, eslabón por eslabón `irb120.dyn`
fdyn	Velocidad de la articulación
friction	Fuerza de fricción
gravload	Fuerza de gravedad en la articulación
inertia	Matriz de inercia en la articulación
nofriction	Pone los parámetros de fricción a cero-
rne	Fuerza/Par en la articulación
payload	Añade una carga al final del *TCP*
perturb	Añade una perturbación aleatoria a los parámetros dinámicos del eslabón

3.2.7. 2) *VARIABLES UTILIZADAS (NOMENCLATURA Y RANGO)*

dh	6 x 5	Matriz cinemática (Denavit-Hartenberg). 6 es número Link Tabla 1
dyn	6 x 20	Matriz cinemática y dinámica. **¡Error! No se encuentra el rigen de la referencia.**

q	1 x 6	Vector de coordenadas angulares, vector de 6 valores.
		`q0=[0,0,0,0,0,0]; q0=zeros(1,6); % q0 (home)`
		`q1=[0.5 0 0 0 0 1.5]; % Posición q1`

Q	m x 6	Trayectoria de coordenadas angulares de m-puntos. `Q=[0,0,0,0,0,0; 0.5,0,0,0,0,1.5]; %Posición q0 y q1`
qd	1 x 6	Vector de velocidades angulares
Qd	m x 6	Trayectoria de velocidades angulares de m-puntos
qdd	1 x 6	Vector de aceleraciones angulares
Qdd	m x 6	Trayectoria de aceleraciones angulares de m-puntos
t	4 x 4	Representación de un punto en matriz homogénea.
T	4 x4xm	Trayectoria de puntos en coordenadas homogénea de m-puntos

3.2.7. 3) *FUNCIONES Y ARCHIVOS DESARROLLADOS*

- **Funciones de modelado y simulación del robot:**

mdl_irb120	Contiene la definición de todos los parámetros del robot, cinemáticos y dinámicos. Define los eslabones (link) con la función *revolute* y compone el robot con la función *serialLink*. Contiene además todas las configuraciones posibles del robot (conceptual, CAD *3D*): con y sin soporte-base, con y sin tool, muñeca esférica o no.
plot_irb120	Dibuja el robot CAD *3D* en el axis del interfaz. Los valores de los eslabones los recoge de los archivos gráficos STL (ASCII). Link1.STL, etc.
espacio_trabajo_lin	Dibuja el espacio de trabajo del robot manteniendo las tres primeras articulaciones fijas Figura 35
ikine_irb120(ARTE) *dh* *solve_spherical_wrist*	Cinemática inversa por método geométrico de ARTE. Se apoya en las funciones: ***dh***: devuelve la matriz de transformación de los eslabones: `A=dh(theta, d, a, alpha)` ***solve_spherical_wrist***: devuelve las últimas 3 variables articulares del robot. `q = solve_spherical_wrist(robot, q, T, wrist, method)`
ikine_nuevo	Cinemática inversa por método de optimización local
save_trayec	Graba las coordenadas articulares de una trayectoria `fid=save_trayec(datos)`
graf_puntos	Gráfica de la desviación entre los puntos de la trayectoria y los alcanzados realmente por el robot. `graf_puntos(T1,T2,t)`
OPCitem	Son funciones para la sincronización del robot virtual con el real. Conecta y localiza las variables de ABB IRC Son llamadas desde la función *plot_irb120*
OPCjoint	Conecta con la aplicación, y coge las variables articulares del robot virtual
Joint2OPC	Normaliza las variables articulares,

- Archivos de datos

Link...STL	Archivo gráfico tipo .STL (ASCII). Contiene la definición gráfica del elemento: eslabón, base, soporte, puntero.
datos.txt	Archivo en formato *.txt* Contiene los valores de una trayectoria de m-puntos en variables articulares: (m-puntos x 6)
datos_dyn.txt	Archivo en formato .txt, Contiene los valores D-H y dinámicos de cada eslabón del robot. Se puede ver en una ventana como opción.

- Funciones del GUIDE (interfaz gráfico)

guide_irb120.fig	Contiene todos los elementos gráficos del interfaz tal y como se muestra en pantalla: rótulos, botones, slider, ventanas, axis, etc.
guide_irb120.m	Es la función principal del interfaz gráfico de *Matlab*. Desde ella accedemos al resto de funciones de la aplicación. Es la función más extensa y compleja de las desarrollada. Es complementaria con la función *guide_irb120.fig* Utiliza la estructura ***handles***, que contiene todo lo relativo a la aplicación: variables, opciones, datos, etc. Permite estructuras aninadas. Puedo acceder desde cualquier función o subfunción, pasándole esta instrucción. `handles.irb120;% Contiene la definición del robot)` Funciones para trabajar con estructuras: ***fieldnames ():*** para saber los campos de la estructura ***isfield (ST,s):*** saber si la cadena *s* es un campo de una estructura *ST* ***isstruct*** (*ST*): saber si *ST* es o no una estructura ***rmfield*** (*ST*,*s*): elimina el campo *s* de la estructura *ST* ***getfield*** (*ST*,*s*): devuelve el valor del campo especificado. Si la estructura es un array hay que pasarle los índices como *cell array* (entre llaves {}) como segundo argumento ***setfield***(*ST,s,v*): da el valor *v* al campo *s* de la estructura *ST*. Si la estructura es un array, hay que pasarle los índices como *cell array* (entre llaves {}) como segundo argumento.

Subfunciones de *guide_irb120.m*	
guide_irb120_67_OpeningFcn	Función de inicio del *guide*, contiene todos los parámetros iniciales de la aplicación, antes de la presentación en pantalla.
boton_...	Son botones de las múltiples opciones del interfaz: `boton_max_vel_Callback`
demo_....	Funciones de demostración. Por ejemplo: `demo_trazar_multipuntos_Callback`
Exit_Callback	Sale de la aplicación y de *Matlab*.
grabar_..	Función de almacenar datos, sin grabarles en fichero `grabar_homogeneas_Callback`
Home_Callback	Pone el robot a la posición q0
*panel_*	Son los paneles de múltiple selección de opciones
pa1_..	Sucesivos puntos en coordenadas articulares, pa1, pa2,...
ph1_..	Puntos en coordenadas homogéneas: ph1, ph2,..
RESET_Callback	Resetea todos las funciones y objetos en el axis y pone en *HOME* el robot.
*robot_*	Diferentes configuraciones del robot, desde la pestaña inicio
save_..	Función para salvar datos, en coordenadas articulares u homogéneas
slider_..	Funciones de control de los sliders.
trazar_...	Traza la trayectoria con cinemática directa o inversa
text_...	Registro de los valores que se muestran en las ventanas
update	Actualiza constantemente la posición del robot en el axis: cualquier movimientos de los slider, entrada por teclado, opciones, etc.
web_...	Funciones de la pestaña ayuda que llevan a hipervínculos de la red

VARIABLES de la estructura *handles* del *guide_irb120.m*	
handles.q0	Valore de la posición inicial de robot: *zeros (1,6)*
handles.q	Array de una posición del robot de variables articulares
handles.Q	Matriz de una trayectoria en variables articulares
handles.t	Punto del *TCP* en coordenadas homogéneas
handles.T	Trayectoria del *TCP* en coordenadas homogéneas
handles.rpy	Ángulos del *TCP*
handles.mnpQ	Índice de manipulabidad de una trayectoria en coordenadas
handles.mnpT	Índice de manipulabidad de una trayectoria en coordenadas homogéneas
handles.velocidad	Escalar de la velocidad. Es parámetro nuestro
handles.met_mnpy	Método del índice de manipulabidad
handles.alg_cin_inv	Selección algoritmo resolución cinemática inversa
handles.mov_eje	Selección eje, sobre el cual se va a mover el *TCP*
handles.puntos_grabados	Guarda los puntos grabados (temporalmente)
handles.ver_puntos	Control de la variable ver puntos en el axis
handles.ver_transp	Control de la variable ver el robot CAD*3D* alámbrico
handles.ver_ejes	Control de la variable ver los ejes del robot
handles.time_opc	Tiempo de sincronización *OPC*

3.3. MODELADO *SIMULINK* CON *SIMMECHANIC*

Una vez definidos todos los parámetros físicos del robot, el siguiente paso de la simulación es colocar los actuadores y elementos de fuerza que generarán el movimiento del robot, así como los sensores que medirán el valor de sus desplazamientos, velocidades, aceleraciones o esfuerzos. Una vez definidos estos, visualizamos el robot virtual, con un comportamiento dinámico semejante al robot real.

Para hacer esto, utilizaremos una herramienta específica del entorno ***Matlab***, que es ***SimMechanic***, basada en el interfaz gráfico ***Simulink***. Permite **modelar y simular sistemas mecánicos** de forma fácil y sin necesidad de complejas ecuaciones y modelos matemáticos. Dicho software es capaz de simular un amplio rango de mecanismos *3D* de lazo abierto y cerrado, que contengan juntas esféricas, de revolución, prismáticas, cilíndricas, universales, etc.

En el diagrama de bloques de la Figura 24 se muestra el proceso de obtención del modelo *Simulink* mediante *SimMechanic* y *SolidWorks*. La información necesaria para realizar estas operaciones la encontramos en el texto [30], en la web de *SimMechanic* [5] y de los estudios [31] y [32].

De forma resumida, la instalación del software *Simscape/SimMechanic* en el entorno de *Simulink* crea en *SolidWorks* un comando en la pestaña de complementos Figura 25, que permite exportar el modelo *CAD 3D*, al formato ***mdl*** de *Simulink*, el cual contiene la definición física y gráfica, del robot con sus típicos bloques (Figura 26).

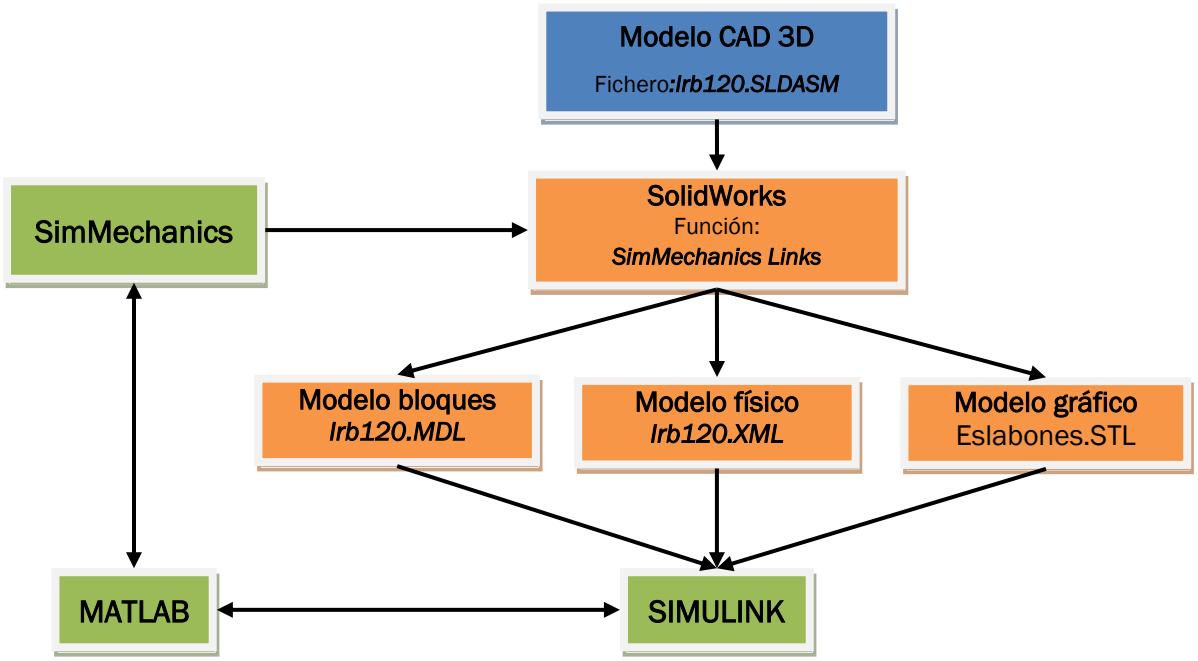


Figura 24. Esquema para obtener el modelo con SimMechanic

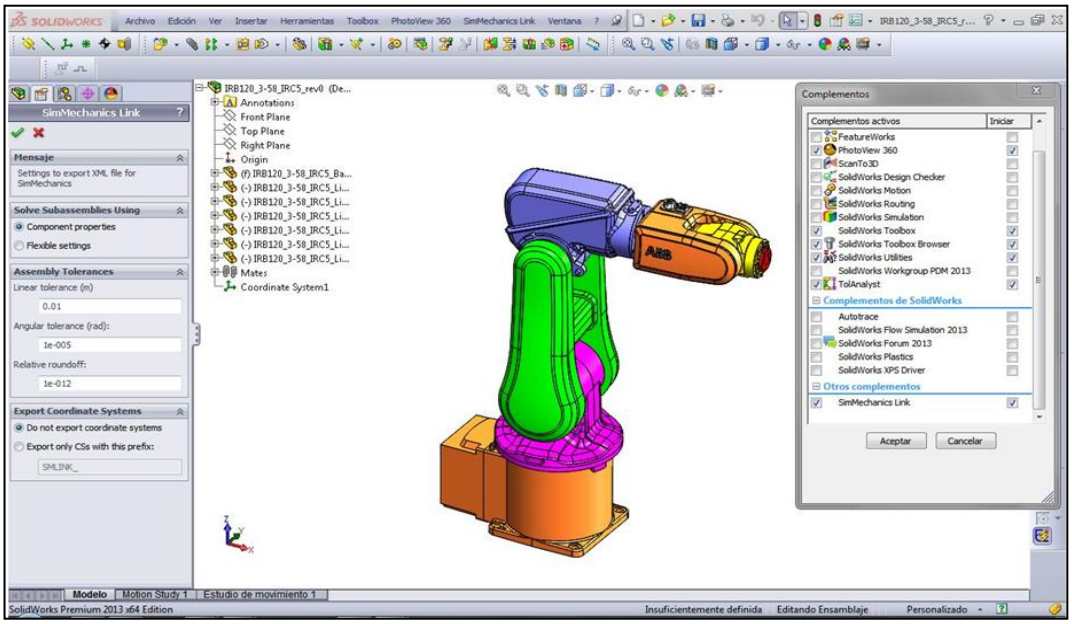


Figura 25. Exportar robot 3D en *SolidWorks* a *Simulink* mediante *SimMechanic*

El proceso genera los siguientes archivos:

- ***irb120_simulink.mdl:*** contiene el modelo del robot tipo bloques simulink.
- ***irb120.xml:*** archivo tipo ***xml*** versión "1.0" encoding="UTF-8". Contiene todos los parámetros físicos del robot, se incorporan y actualizan desde el archivo ***mdl***.
- ****.STL***: estos archivos contienen la información gráfica de cada uno de los componentes del robot, pero esta vez en formato binario.

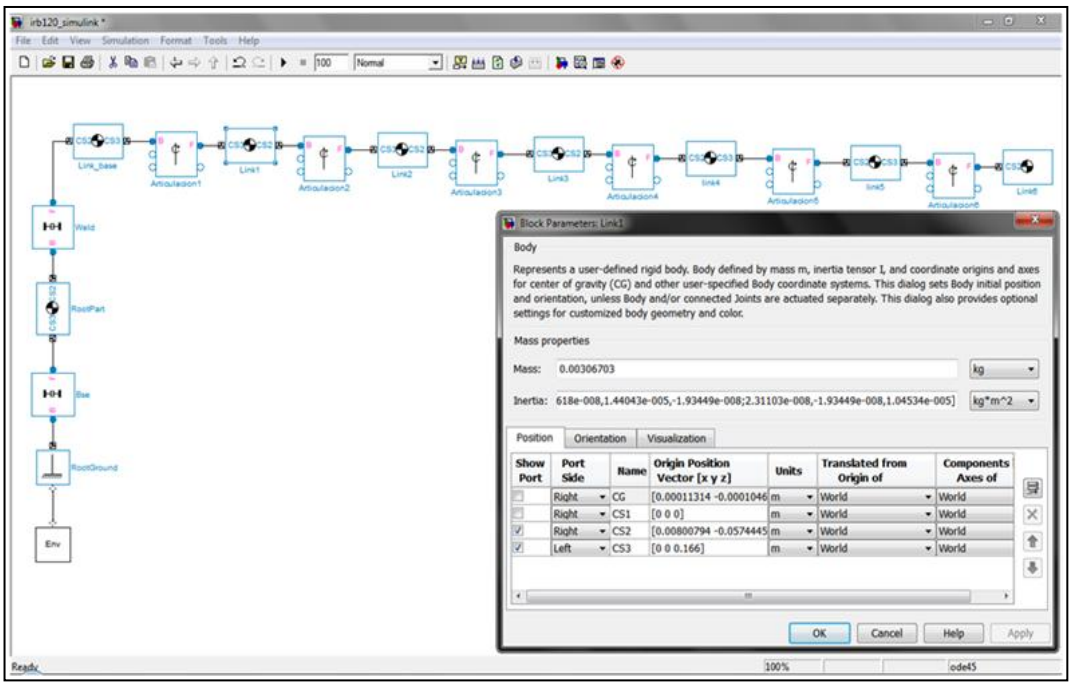


Figura 26. Modelo del robot Simulink y pestaña de parámetros del eslabón 1

Al modelo básico, así obtenido, se le van añadiendo los bloques propios de *SimMechanic* (Figura 29):

- Valores iniciales y límites de la articulación (IC).
- Actuadores y sensores sobre las articulaciones.
- Señales de entrada y salida.

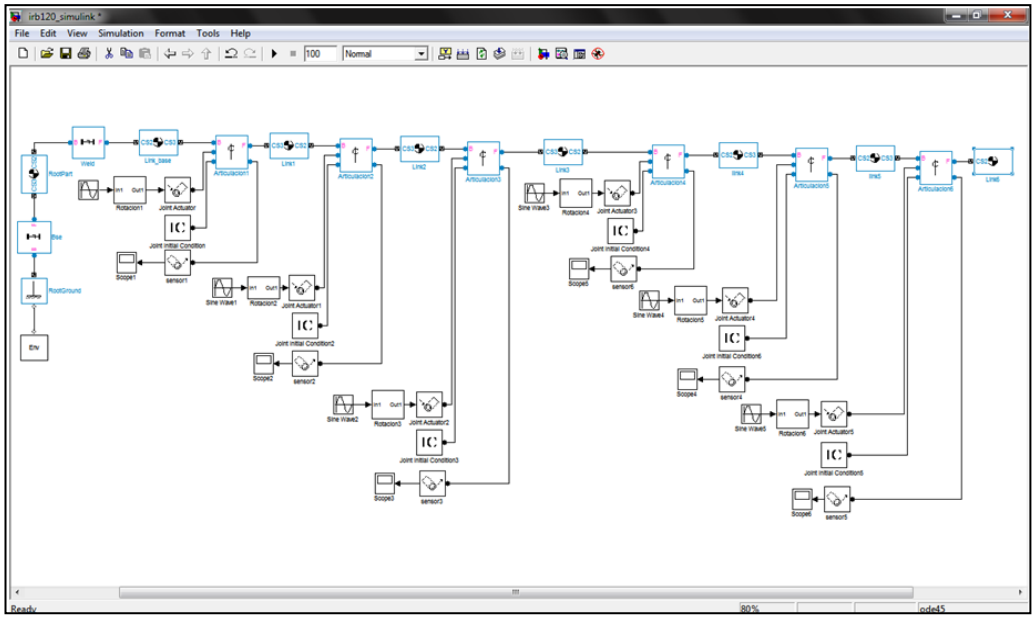


Figura 27. Modelo simulink del robot irb120 completo

Consiguiendo finalmente el modelo de la Figura 27, que al simularlo, se consiguen los efectos dinámicos deseados (Figura 28). Este modelo (abierto) es susceptible de incorporar multitud de elementos: motores, reductores, juntas, etc.

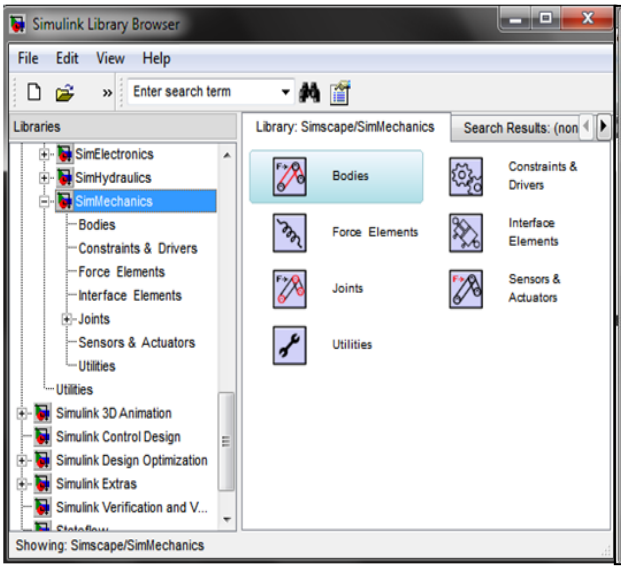


Figura 29. Librería SimMechanic

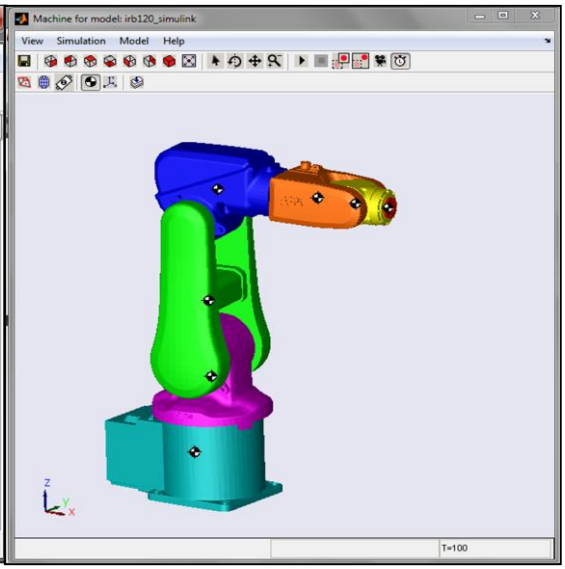


Figura 28. Simulación con SimMechanic

La Figura 30 contiene, detalladamente, parte de la definición del robot:

- Los bloques del entorno del robot: ***Env***.
- Parámetros físicos de la base: ***Link_base***.
- Parámetros físicos del primer eslabón: ***Link1***.
- Definición de la primera articulación: ***Articulación1***. En ella se sitúan como ocurre en el robot real, los elementos de control y transmisión del movimiento:
  - Actuador (***Joint Actuator***): dota de movimiento la articulación (velocidad o fuerza).
  - Tratamiento de la señal de entrada: ***Rotación 1***
  - Señal de entrada, en este caso sinusoidal.

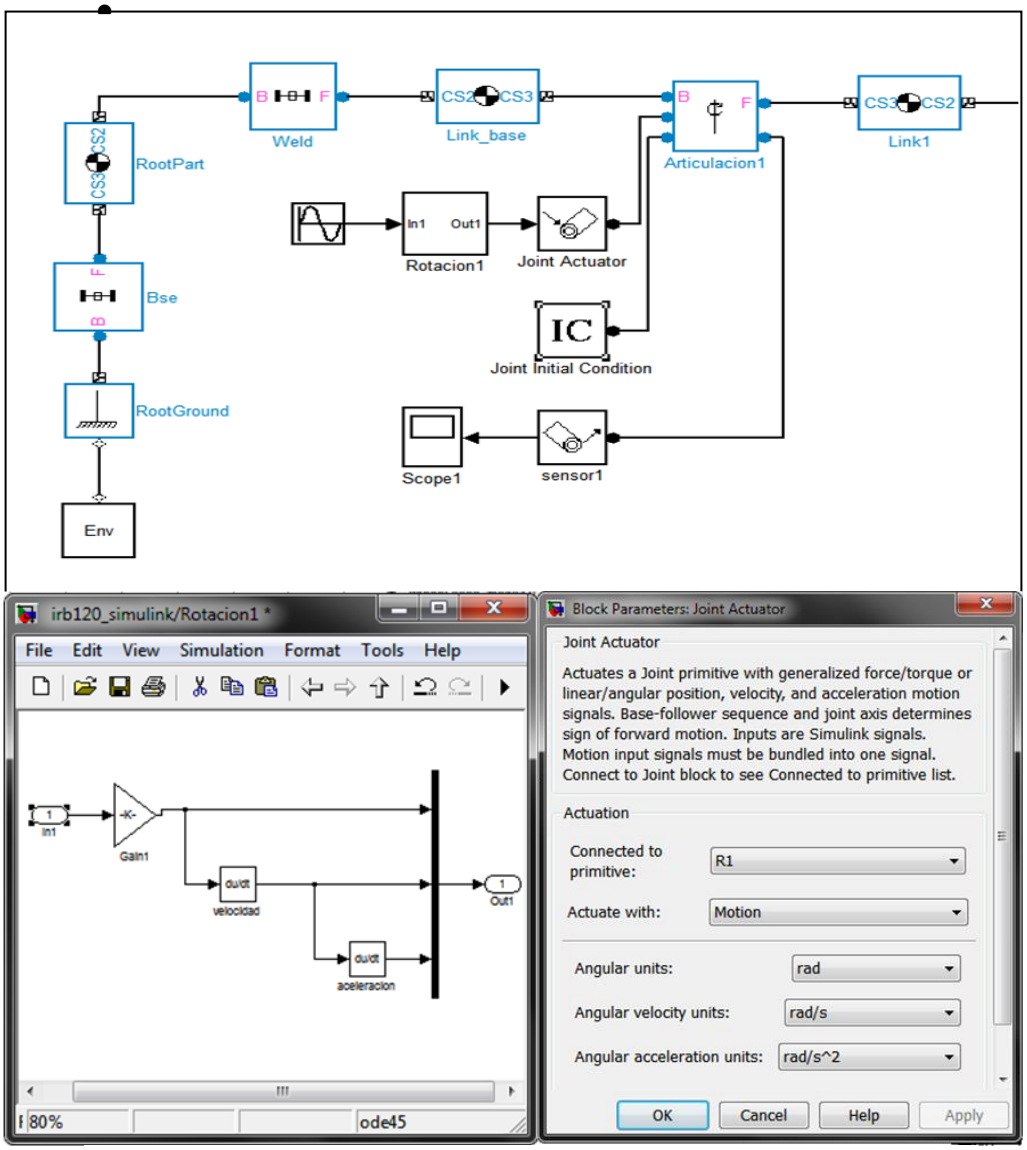


Figura 30. Modelado primera articulación del robot

3.4. CONEXIÓN MEDIANTE OPC AL ROBOT REAL

Nuestro trabajo consiste ahora en recoger las variables articulares que genera cada posición del robot virtual (al realizar una trayectoria) y pasárselas al robot real en formato compresible, para que este realice las mismas acciones que el virtual , sincronizando ambos. Para hacer esto recurrimos a un software de comunicación industrial ampliamente utilizado en la actualidad: ***OPC***.

Las siglas ***OPC*** (***OLE for Process Control***) son un estándar de comunicación en el campo del control y supervisión de procesos industriales, basado en una tecnología ***Microsoft***, que ofrece una interfaz común para comunicación que permite que componentes software individuales interaccionen y compartan datos.

En nuestro caso la comunicación se realiza a través de una arquitectura ***Cliente-Servidor***, utilizando el software al *OPC* de *Matlab* [7] y el de *ABB* [8], de forma que:

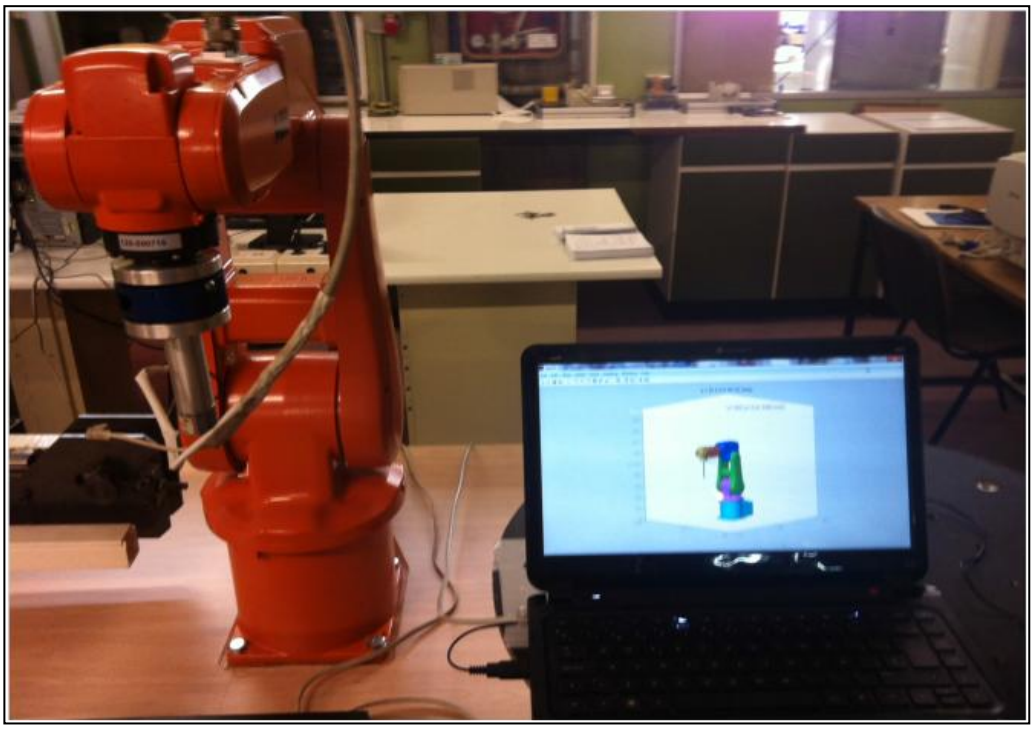
- El ***Cliente OPC***, es la fuente de datos, en nuestro caso el robot virtual, que al trazar una trayectoria, genera las sucesivas variables articulares de la posición del robot $q_i(t)$. Estas, las extraemos en la función ***plot_irb120*** mediante la función ***Joint2OPC.***
- El ***Servidor OPC***, es el controlador del robot real, el IRC5 del robot [33], que aperece en el interfaz de *OPC* como ***'ABB.IRC5.OPC.Server.DA'***), al cual le enviamos el valor de las variables articulares (normalizadas), cada cierto tiempo, establecido en la variable ***handles.time_opc***, de forma que robot real y virtual sincronizan sus movimientos (Figura 31).
- 

Figura 31. Sincronización del robot real y virtual

3.5. INTERFAZ GRÁFICO DE LA APLICACIÓN

El interfaz desarrollado sobre ***Guide*** de *Matlab*, trata de ser un entorno amigable e intuitivo para el usuario (Figura 32), de forma que pueda:

- Introducir fácilmente, diferentes configuraciones de robot.
- Observar el movimiento del robot, de sus eslabones, del espacio de alcanzable.
- Mover el robot de forma intuitiva, en coordenadas articulares o en coordenadas homogéneas.
- Grabar puntos en ambos formatos, para trazar trayectorias. Con articulares mediante cinemática directa y con puntos del TCP en coordenadas homogéneas, con la cinemática inversa.
- Detectar la dificultad computacional que entraña la cinemática inversa.
- Trazar trayectorias, exportar e importar sus datos.
- Elegir entre las diversas opciones de algoritmos, para el control del mismo.
- Obtener la información de todos los parámetros que intervienen en el modelado y simulación del robot.
- Acceder a ***simmulink***, para el estudio dinámico del robot.
- Conectar la aplicación con el robot real, mediante software *OPC*, permitiendo la sincronización del robot virtual y del real.

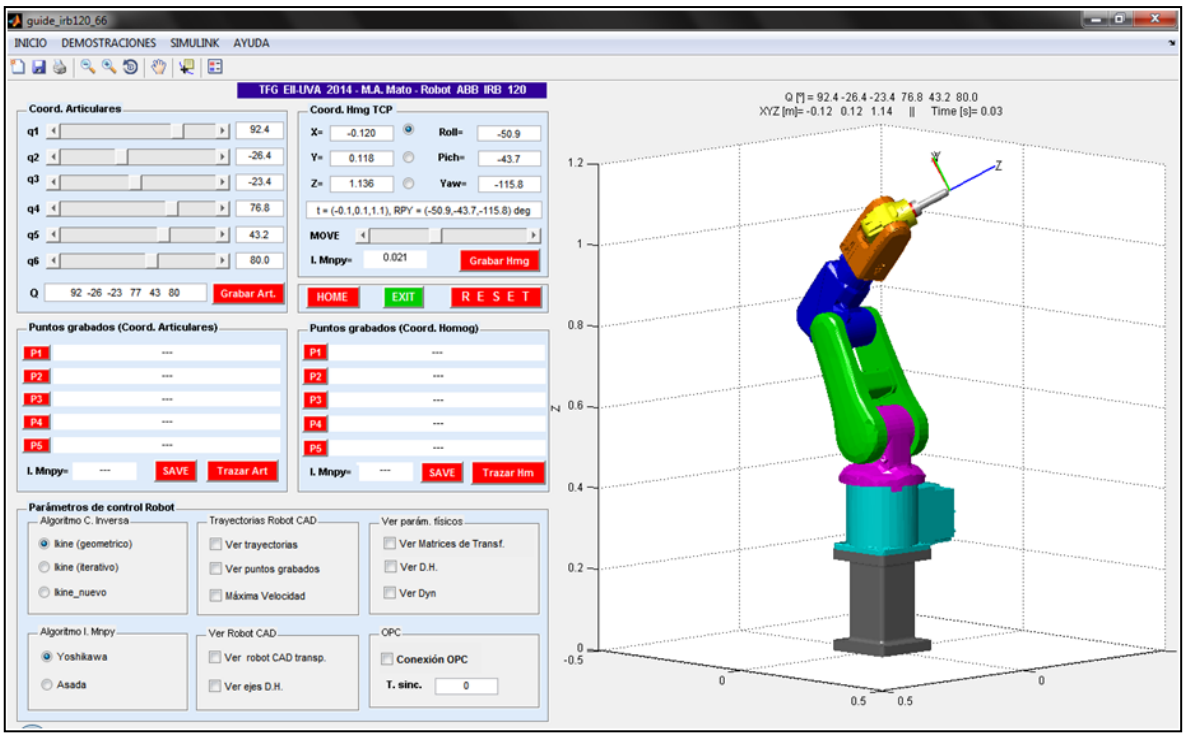


Figura 32. Interfaz gráfico de la aplicación

3.5.1. INICIO. CONFIGURACIONES DEL ROBOT

Desde la pestaña ***INICIO***, podremos cargar varias configuraciones del robot como las que se muestran en la Figura 33 :

- ***Robot 3D CAD***: modelo idéntico al real, con máximo realismo gráfico.
- ***Robot 3D CAD***, con puntero: idéntico al anterior al que se le añade un puntero (*tool* de 0,1 m).
- ***Robot 3D CAD***, con puntero y soporte-base: al anterior se le añade un soporte al anterior (Figura 33-izqda.)
- ***NO esférico***: modelo del toolbox, de tipo conceptual, idéntico al real (Figura 33-drcha.)
- ***NO esférico, con base y puntero***: igual que el anterior, pero con base y puntero.
- ***Esférico, con puntero***: modelo de tipo conceptual esférico, con muñeca esférica.

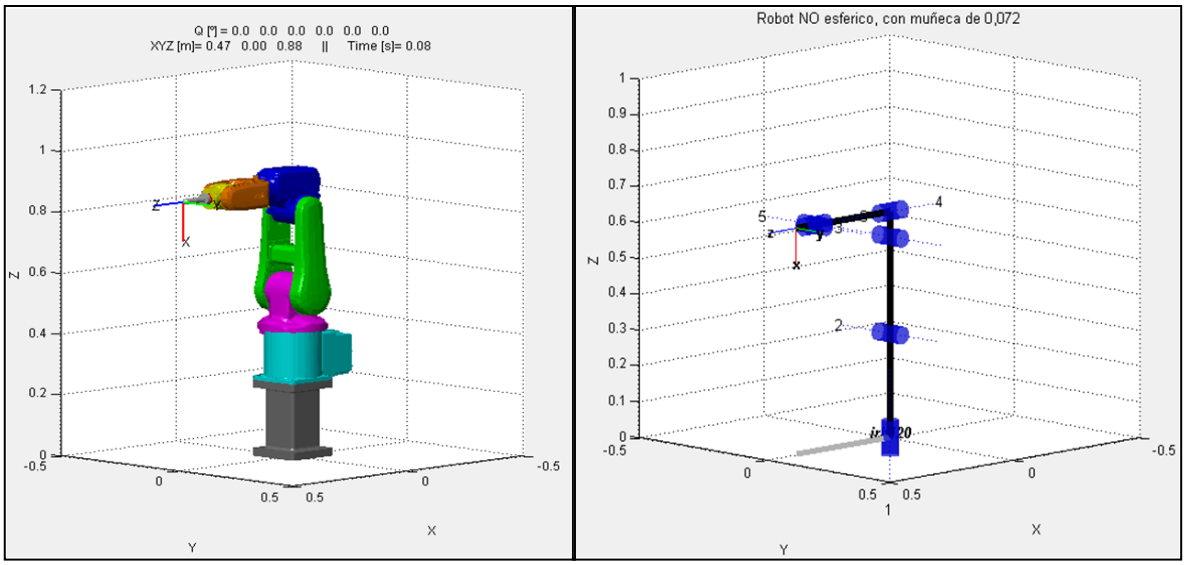


Figura 33. Dos de las seis posibles configuraciones del robot

3.5.2. DEMOSTRACIONES

Desde esta pestaña accedemos a varias funciones, que muestran el comportamiento del robot:

- ***Alcance máximo***: muestra los puntos de máximo alcance, en forma de superficie *3D* (Figura 34).
- ***Espacio de trabajo***: desde una posición determinada, muestra las posiciones alcanzables por las últimas tres articulaciones (Figura 35).

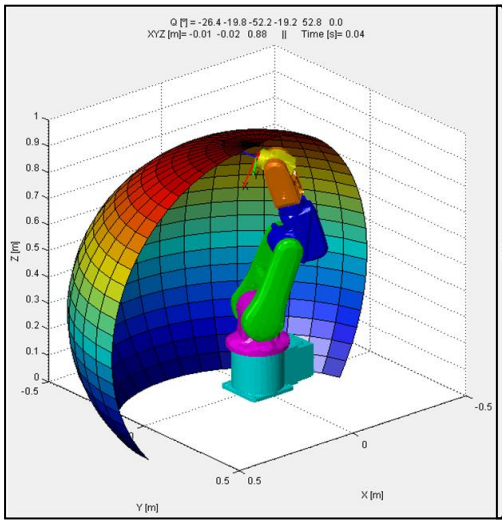


Figura 34. Alcance máximo

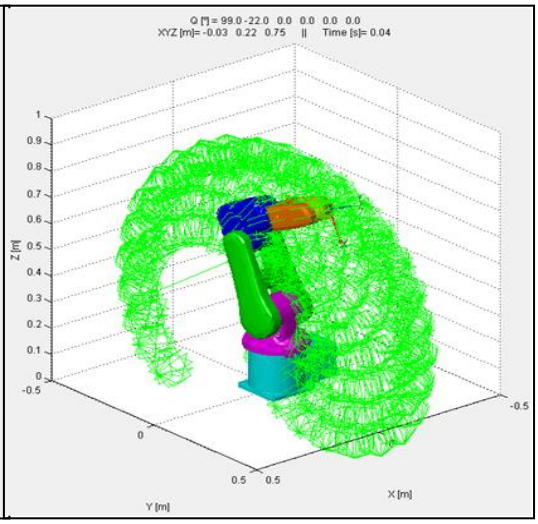


Figura 35. Espacio de trabajo

- ***Movimientos del robot***: muestra los movimientos del robot eslabón por eslabón.
- ***Trayectorias. Trazar Multipuntos:*** el robot traza una trayectoria predefinida por punto (Figura 36)
- ***Trayectorias. Trazar poligonal:*** el robot traza una trayectoria siguiendo una línea poligonal *3D* predefinida (Figura 37).
- ***Trayectorias. Trazar de Fichero***: lee los datos del fichero *datos.txt* en forma de matriz Q (mx6), donde m -filas-, es el número de puntos, y 6, son las variables articulares que definen un punto (q).
- ***Cinemática inversa. Multiples posiciones del punto:*** muestra las diferentes configuraciones posicionales de los eslabones, para alcanzar el punto donde se encuentre el robot, lo hace mediante la función *ikine_irb120*. Por tanto debe estar seleccionado esta opción en el algoritmo de cinemática inversa.

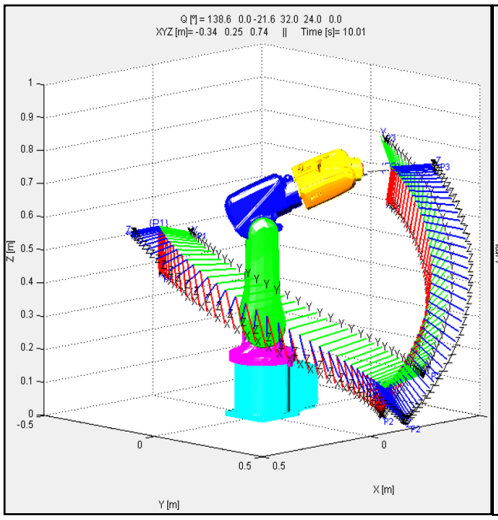


Figura 36. Trayectoria tres puntos

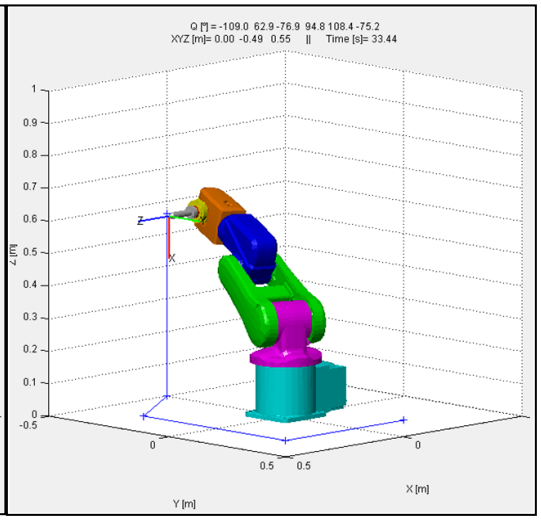


Figura 37. Trayectoria poligonal

3.5.3. SIMULINK

Esta pestaña nos lleva al entorno de ***Simulink***, en el cual nuestro modelo numérico está definido para trabajar con sus típicos bloques. El nuevo modelo, además de los parámetros físicos del toolbox de Corke, contiene nuevas funcionalidades de ***SimMechanic***

3.5.4. OPERACIONES CON EL ROBOT VIRTUAL

En la parte izquierda de la aplicación aparecen los dispositivos de manipulación y control del robot (Figura 38), desde la cual podremos:

- Mover el robot en coordenadas articulares mediante 6 ***slider*** que controlan el valor de las variables articulares. Para cada posición el slider recoge un valor.
- Para cada posición del robot, saber las coordenadas espaciales del ***TCP*** en coordenadas homogéneas, describiendo su posición en coordenadas cartesianas (**x**, **y** ,**z**) más la orientación de la muñeca con: ***Roll, Pich*** *y* ***Yaw.***
- En la zona de coordenadas homogéneas podemos seleccionar un valor ***X***, ***Y*** o ***Z***, y mover al robot sobre dicho eje seleccionado con el slider ***MOVER***, mantiene las otras dos coordenadas.
- En cada ventana de coordenadas aparece el valor del vector Q, o de la matriz T que definen la posición del robot.
- Todos los valores que aparecen en las ventanas, son modificables desde el teclado, el robot se moverá a la posición resultante.
- En la parte inferior de cada ventana de coordenadas, aparece un botón (***Grabar_***) que nos permite registrar la posición en la que está el robot en ese momento, en la ventana de ***Puntos grabados***.

Aparece también una pequeña ventana independiente, con:

- ***HOME***: coloca el robot en su posición inicial.
- ***RESET***: además de lo anterior, borra todos los registros almacenados.
- ***EXIT***: sale de la aplicación y de *Matlab.*

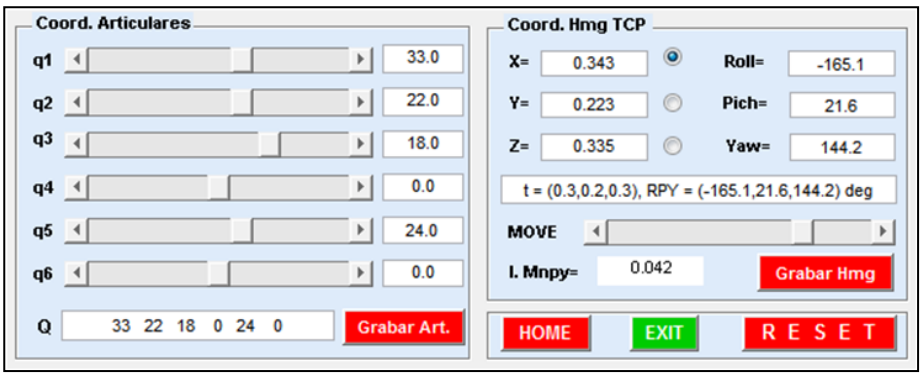


Figura 38. Ventanas de coordenadas

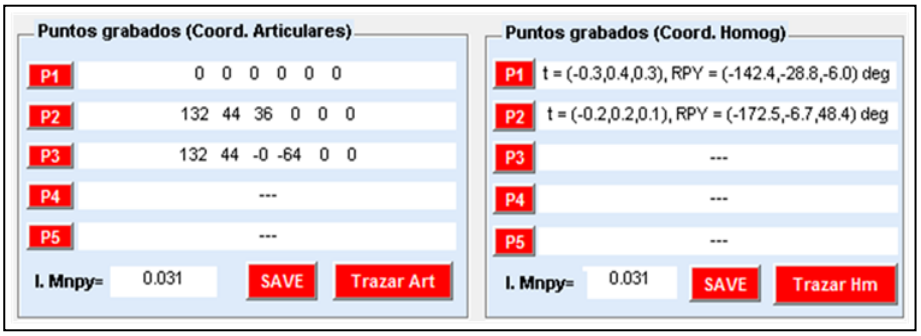


Figura 39. Ventanas de puntos grabados

En la zona intermedia, aparecen hasta cinco campos para almacenar puntos, en coordenadas articulares y homogéneas (Figura 39). La aplicación permite almacenar más de 5 puntos, aunque no aparezcan en pantalla. Una vez grabados los puntos, podremos:

- Ir al punto grabado pulsado el botón ***P1***, ***P2***, etc.
- Trazar la trayectoria definida por dichos puntos.
- Grabar los puntos en un fichero de nombre ***datos.txt***.
- Obtener información a través del índice de manipulabilidad, de la facilidad que tiene el robot para el trazado de la trayectoria, que definen los puntos grabados.

3.5.5. PARÁMETROS DE CONTROL y VISUALIZACIÓN

En la zona inferior aparecen los **Parámetros de control del Robot** (Figura 40) :

- Selección del algoritmo de cinemática inversa: ikine (geométrico), ikine (iterativo), ikine (nuevo).
- Selección del algoritmo de manipulabilidad: *Yoshikawa* ó *Asada*.
- Selección de la visualización de trayectoria: ver puntos grabados en el *Axis*, ver la trayectoria, aumentar la velocidad del robot.

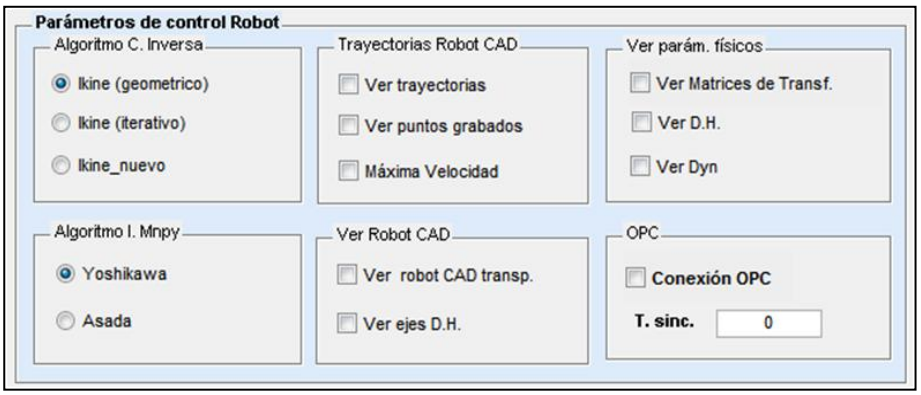


Figura 40. Parámetros de control del modelo

- Selección de visualización del robot *3D CAD*: ver modelo alámbrico, ver los ejes *D-H* del robot.
- Mostrar información del robot en ventana flash: parámetros DH, parámetros dinámicos, o las matrices de transformación para cada punto (del robot y de cada eslabón)
- Conectar la aplicación mediante ***OPC*** con el robot real. Tendremos que introducir un parámetro para sincronizar la velocidad de ambos.

3.5.6. VENTANA GRÁFICA (*AXIS*)

La ventana gráfica es un *Axis* de *Matlab* en la cual nos aparece el modelo numérico ejecutando las acciones que le requerimos. El modelo de mayor realismo, resulta computacionalmente más lento que el del toolbox, más conceptual.

En la parte superior, del *Axis* se suministra información acerca de su posición: variables articulares del robot, y posición del *TCP* (x, y z). Estos valores salen directamente del modelo en el axis, de forma que nos sirve para validar que las posiciones alcanzadas son correctas. Se ha añadido, también, un contador en tiempo real, útil para controlar la duración de las trayectorias.

Además sobre esta zona gráfica (***axis***), se pueden utilizar los comandos ***Zoom***, ***Perspectiva*** y ***Desplazar***, etc. de la zona de ***herramientas***, pudiendo obtener vistas detalladas de la simulación Figura 41.

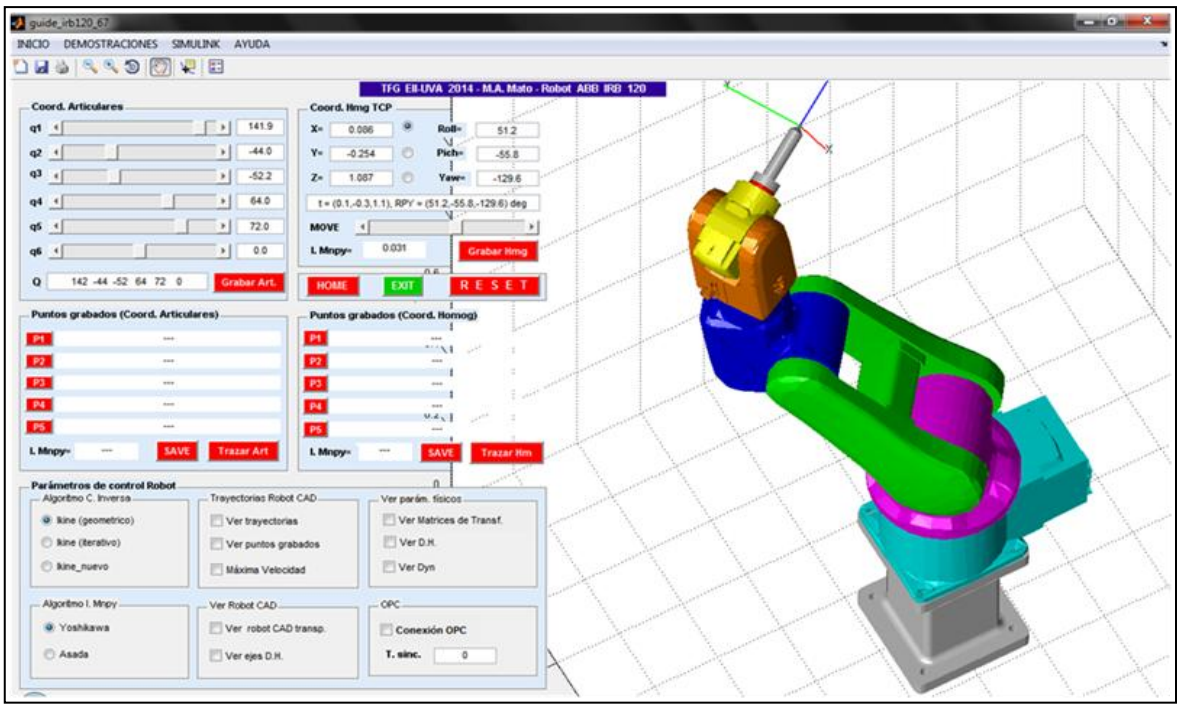


Figura 41. Zoom y perspectiva del *axis* del robot

CAPÍTULO IV

4. CONCLUSIONES

4.1. CONCLUSIONES

Con el trabajo realizado, concluimos que:

- ***Robotic Toolbox*** de ***Matlab*** es una herramienta válida para el estudio de los robots industriales, permitiendo conocer de forma sencilla, efectiva y gráfica, los principios físicos-matemáticos en los que se basa su control y simulación.
- La aplicación desarrollada sobre *Matlab,* con el *toolbox,* describe perfectamente el modelado numérico del robot manipulador ***IRB120*** de ***ABB***.
- El interfaz gráfico mejora el realismo gráfico del modelo gráfico del *toolbox*, gracias al software *CAD 3D*, ***SolidWorks,*** sobre un modelo obtenido de *ABB*.
- La arquitectura abierta del lenguaje de programación del robot real, permite conectar éste, mediante software ***OPC,*** con el modelo numérico de la aplicación, sincronizando ambos, y validando el resultado.
- El *toolbox* está en continuo proceso de mejora. Su autor, ***Peter Corke***, mantiene, a través de su web, un valioso flujo de información, que nos ha sido muy útil.
- El fabricante ***ABB*** suministra suficiente información para el estudio de sus robots. Posibilitando, además, a través de su web contactar con su personal.
- Para la simulación dinámica del robot, existe software basado en *Matlab*, como ***SimMechanic***, que mejora el *Toolbox.*
- Y, finalmente, la aplicación desarrollada, es un excelente **recurso didáctico**, que permite a cualquier usuario introducirse en el estudio y manejo de la robótica industrial.

4.2. LÍNEAS FUTURAS DE MEJORA

Queda abierta aquí esta línea de trabajo para todos aquellos interesados en el estudio de la robótica, y que, quieran:

- Conectar esta aplicación con ***RobotStudio***, a través de programas escritos en ***RAPID***.
- Profundizar en el estudio dinámico del robot, con ***SimMechanic***, añadiendo o sustituyendo componentes: actuadores, sensores, señales de entrada.

CAPÍTULO V

5. ESTUDIO ECONÓMICO

En este apartado desarrollamos el estudio económico necesario para la ejecución de este trabajo. Se desglosa la contabilidad por actividades, y se valoran los costes de cada una de ellas.

Apdo.	CONCEPTO	Ud.	Precio	TOTAL
1	HORAS DE TRABAJO (Ingeniero)			
1.1	Estudio y modelado de robot en *Matlab*	100		
1.2	Modelado del robot en SolidWorks	50		
1.3	Modelado del robot en SimMechanic	50		
1.4	Desarrollo interfaz gráfico Guide	100		
1.5	Reuniones con dirección del proyecto	20		
1.6	Sincronización del robot virtual con real	20		
1.7	Elaboración de documentación	60		
	TOTAL HORAS PROYECTO	400	50	20.000
2	SOFTWARE Y EQUIPOS:			
2.1	Software: *Matlab*, Simulink y *SimMechanic* (V.Educacional)	1	500	500
2.2	Software: *SolidWorks* Premium 2013 (V. Educacional)	1	500	500
2.3	Software: RobotStudio (V. Educacional)	1	500	500
2.4	Hardware: Portátil Toshiba Sattelite PRO 660C	1	600	600
2.5	Equipo robótico Robot rb120 con controlador y teach-pendant	1	20.000	20.000
2.6	Accesorios puntero robot	1	400	400
	TOTAL SOFTWARE Y EQUIPOS			22.500
	TOTAL PROYECTO			42.500

CAPÍTULO VI

6. BIBLIOGRAFIA

[1] **MATLAB (R.2010a)**. Software Mathworks. (Junio-214). http://www.mathworks.es

[2] *Corke, P. I.* (2013). "Robotic Toolbox for *Matlab*. Release 9.8". http://petercorke.com/Home.html

[3] ***Solidworks*** Premium 2013x64-SP3.0. (Jun-2014). http://www.solidworks.com

[4] ***Simulink***, Software *Mathworks*. (Jun-2014). http://www.mathworks.es/products/simulink/

[5] ***SimMechanic***. (Jun-2014). http://www.mathworks.es/es/help/physmod/smlink/index.html

[6] ***Guide***. Software *MathWorks* (Jun-2014). http://www.mathworks.es/discovery/matlab-gui.html

[7] ***OPC.*** Software *MathWorks* (Jun-2014). http://www.mathworks.es/products/opc/

[8] ***OPC***. Manual aplicación ABB. Documento ID: 3HAC023113-001.

[9] "Global robotics industry: Record beats".(Jun-2014). http://www.ifr.org/news/ifr-press-release/global-robotics-industry-record-beats-record-621

[10] *Marshall, D, Bredin, C*. (2008). "Historia de un éxito". Revista ABB 2/2008, pág. 56-62.

[11] *Zhenli Lu, y otros*. (2011). "A Brief Survey of Commercial Robotic Arms for Research on Manipulation". 3º International Conference on Computer and Network Technology. (ICCNT 2011)

[12] *Ivaldiy S., Padoisy V., Norix* F. (2014) ."Tools for dynamics simulation of robots: a survey based on user feedback". ArXiv:1402.7050v1 [cs.RO].

[13] ***Inventor***. ***Autodesk.*** (Jun-2014) .http://www.autodesk.es/products/autodesk-inventor-family/features/all/gallery-view

[14] ***RobotExpert***. Siemens PLM software. (Jun-2014). http://www.plm.automation.siemens.com/es_es/

[15] ***RobotWorks***. (Jun-2014). http://robotworks-eu.com/products/RBWabout.htm

[16] ***Kuka Sim***. Software simulación robots Kuka (Jun-2014). http://www.kuka-robotics.com/es/products/software/kuka_sim/kuka_sim_detail/PS_KUKA_Sim_Layout.htm

[17] ***RobotGuide.*** Software de simulación robot ***Fanuc*** (Jun-2014) http://www.fanucrobotics.es/es/products/software

[18] ***RobotStudio***. ABB robotic. Manual del operador. (Jun-2014). ID de documento: 3HAC032104-005. http://new.abb.com/products/robotics/robotstudio

[19] ***RAPID***. Manual del operador. ABB robotic. (RobotWare 5) (2013). ID de documento: 3HAC029364-005.

[20] *Bonev I.* (Jun-2014) ***RokiSim*** (Robot Kinematic Simulation), Escuela de Superior de Tecnología (Quebec, Canadá). http://en.etsmtl.ca/Unites-de-recherche/CoRo/Accueil

[21] *Gil Aparicio, A.* (Jun-2014). ***ARTE*** (A Robotic Toolbox for Education) Universidad Miguel Hernández (Elche, España), http://arvc.umh.es/arte/index_en.html

[22] ***ARVC.*** *Dpto. Automática Universidad Miguel Hernández (Elche-España).* (Jun-2014). http://arvc.umh.es/index

[23] ***HEMERO***. (Jun-2014). http://grvc.us.es/hemero/descargar.html

[24] Manual del producto: IRB 120-3/0.6. ID de documento: 3HAC035728-005. (Jun-2014). http://new.abb.com/products/robotics/industrial-robots/irb-120/irb-120-data

[25] *Barrientos, A*. (2007). Fundamentos de Robótica (2ª edición), MacGraw-Hill.

[26] Modelo CAD, IRB 120, Ver 3-58, IRC5, rel0. *Solidworks*, (Jun-2014). http://www.abb.es/product/seitp327/26a64e26204118e5c12576a4004a8497.aspx?productLanguage=es&country=ES&tabKey=7

[27] *Hartenberg, R.S., Denavit, J.,* (1955), "A kinematic notation for lower pair mechanisms based on matrices". Journal of Applied Mechanics, vol. 77, pp. 215-221.

[28] *Corke, P*. (2013). Robotics, Vision & Control. Springer.

[29] ***V-Realm Builde***r. (Jun-2014). http://www.mathworks.es/es/help/sl3d/install-v-realm-builder.html.

[30] *Martínez J., Sabater J.M.* (2012). "Guía Docente para el Diseño de Robots de Servicio". (Jun-2014). https://sites.google.com/a/goumh.umh.es/hidma/home

[31] *Gouasmi, M*, y otros. 2008. "Kinematic Modelling and Simulation of a 2-R Robot Using SolidWorks and Verification by *MATLAB*/Simulink". ISSN 1729-8806 DOI: 10.5772/50203. "International Journal of Advanced Robotic Systems".

[32] *Perez M.A., Cuevas E.V., Zaldivar D.* (2014) Fundamentos de Robótica y Mecatronica con *Matlab* y Simulink. Editorial RA-MA.

[33] Manual del operador IRC5 con FlexPendant. ABB robotic. ID de documento: 3HAC16590- 5.

CAPÍTULO VII

7. ANEXOS

7.1. CODIGO DEL INTERFAZ GRÁFICO

7.1.1. FUNCIÓN guide_irb120

```
function varargout = guide_irb120(varargin)
% GUIDE_IRB120 M-file for guide_irb120.fig
%      Simulación y control de robot ABB irb120
%      Miguel A. Mato. UVA-TFG Ingeniero Mecánico- 2014
%      H = GUIDE_IRB120 returns the handle to a new GUIDE_IRB20_31 or the handle
%      the existing singleton*.
%      GUIDE_IRB120('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUIDE_IRB120.M with the given input arguments.
%
%      GUIDE_IRB120('Property','Value',) creates a new GUIDE_IRB120 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before guide_irb120_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to guide_irb120_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES
% Edit the above text to modify the response to help guide_irb120
% Last Modified by GUIDE v2.5 05-Jul-2014 14:41:36
% Begin initialization code - DO NOT EDIT

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @guide_irb120_OpeningFcn, ...
                   'gui_OutputFcn',  @guide_irb120_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guide_irb120 is made visible.
function guide_irb120_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guide_irb120 (see VARARGIN)
```

```
%Situar el guide en el centro de la pantalla del ordenador
scrsz=get(0,'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3)-pos_act(3);
xp=round(xr/2);
yr=scrsz(4)-pos_act(4);
yp=round(yr/2);
set (gcf,'Position', [xp,yp,pos_act(3), pos_act(4)]);
%****
handles.output = hObject;
%figure(handles.figure1);
%axes(handles.axes_robot);

% Configuración del axes (donde aparece el robot)
xlabel('X [m]'); ylabel('Y [m]'); zlabel('Z [m]');
axis ([-0.5 0.5 -0.5 0.5 0 1]);
campos([2 2 1]); % posición de la camara
daspect([1 1 1]);
camproj orthographic; %perspectiva
grid on
hold on

% Definimos e iniciales valor de las variables
%cla;
q0=zeros(1,6); % q0 articulares
handles.q0=q0;
handles.q=q0;
handles.Q=[];% Matriz coordenas articulares
handles.T=[];%Matriz coordenadas homogeneas
handles.rpy=[];% Angulos del TCP
handles.mnpQ=[]; %
handles.mnpT=[];
handles.velocidad=50; % velocidad máxima
handles.met_mnpy='yoshikawa';
handles.mov_eje = handles.man_x;
handles.tipo_robot=14; %cargo con defecto el robot 3DCAD
handles.irb120=mdl_irb120(handles.tipo_robot); %definimos el robot físicamente
handles.t=handles.irb120.fkine(q0);% Matriz del punto en coord. homogeneas, en
cada posición
handles.puntos_grabados=[]; %Puntos grabados
handles.demos =[]; % dibujos de las demos
% Parametros visión robot CAD
handles.ver_puntos_grabados=false;
handles.ver_transp=false; % Sin transparencia
handles.ver_ejes=false; % ver ejes en estado cero
handles.ver_trayec=false;
handles.alg_cin_inv=0; % Algoritmo de cinemática inversa por defecto el
geométrico%handles.text_on_opc='OFF OPC';
handles.time_opc=0; % tiempo de sincronización del reloj
%handles.punto=[];
%texto de coordenas articulares (a la derecha del slider)
for i=1:5
t=sprintf('handles.text_pa%g', i);
set(eval(t),'String', '---');
end

% Texto de coordenadas homogeneas
for i=1:5
t=sprintf('handles.text_ph%g', i);
set(eval(t),'String', '---');
end

 %Actualizao valor del slider mover
  set(handles.marcar_coord, 'SelectedObject', handles.man_x);
```

```
  set (handles.text_time_opc, 'String', '0');
%get(gcf,'CurrentAxes') ; % miro en que axes estoy
%isa(handles.irb120, 'function_handle'); % para saber lo que lleva el robot
update (hObject, handles, handles.q0);

%guidata(hObject, handles);

% --- Executes on button press in RESET ---> limpio todo
function RESET_Callback(hObject, eventdata, handles)
% hObject    handle to RESET (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

cla; clc; %borro pantalla command y figures
%xlabel('X'); ylabel('Y'); zlabel('Z');
axis ([-0.5 0.5 -0.5 0.5 0 1]);
campos([2 2 1]); % posición de la camara
daspect([1 1 1]);
camproj orthographic; %perspectiva
%grid on

q0=zeros(1,6); % q0 articulares
handles.q0=q0;
handles.q=q0;
handles.Q=[];% Matriz coordenas articulares
handles.T=[];%Matriz coordenadas homogeneas
handles.t=[]; % Matriz del punto en coord. homogeneas
handles.rpy=[];% Angulos del TCP
handles.mnpQ=[]; %
handles.mnpT=[];
handles.velocidad=50; % velocidad máxima
handles.met_mnpy='yoshikawa';
handles.alg_cin_inv=0; % por defecto el algoritmo es el geométrico
handles.mov_eje = handles.man_x;
handles.puntos_grabados=[]; % puntos grabados para dibujarlos
handles.ver_puntos = false;
handles.ver_transp=false; % NO ver la transparencia del robot
handles.ver_ejes=false; % ver ejes en estado cero
handles.ver_trayec=false;
%handles.tipo_robot=14; %cargo con defecto el robot 3DCAD
handles.irb120=mdl_irb120(handles.tipo_robot); %definimos el robot físicamente
handles.ver_puntos_grabados=false;
handles.time_opc=0;

% valores de las posiciones grabadas en articulares
for i=1:5
t=sprintf('handles.text_pa%g', i);
set(eval(t),'String', '---');
end

% valores de las posiciones grabadas en CH
for i=1:5
t=sprintf('handles.text_ph%g', i);
set(eval(t),'String', '---');
end

set(handles.text_mnp_Q,'String', '---'); % indice manipulación de Coor Q
set(handles.text_mnp_H,'String', '---'); % indice manipulación de Coor H

    set (handles.text_para_robot, 'Visible', 'off'); %ocultar parámetros
    set(handles.panel_alg_mnpy, 'SelectedObject', handles.yoshikawa);
    % Restaurar estado de botones
    set(handles.boton_ver_dh,'Value', 0);% Botón ver dh
```

```
    set(handles.boton_ver_dyn,'Value', 0);
    set(handles.boton_ver_MT,'Value', 0);   % Ver matríz de transformación
    set(handles.boton_max_vel,'Value', 0);
    set(handles.boton_ver_ejes,'Value', 0);
    set(handles.boton_ver_transp,'Value',0); % ver el robot NO transparente
    set(handles.boton_max_vel,'Value', 0);
    set(handles.boton_ver_puntos_grabados,'Value', 0);
    set(handles.boton_ver_trayec,'Value', 0);
    set(handles.boton_conx_opc,'Value', 0); %conx_opc
    %Actualizao valor del slider mover
    set(handles.marcar_coord, 'SelectedObject', handles.man_x);
   % set(handles.slider_mover, 'Value',handles.t(1,4));

    set (handles.text_time_opc, 'String', '0');
    set (handles.text_on_opc, 'Visible', 'off'); %ocultar parámetros
    set (handles.text_para_robot, 'Visible', 'off'); %ocultar parámetros
   %{
    set (handles.robot_14, 'Checked', 'on');
    set (handles.robot_15, 'Checked', 'off');
    set (handles.robot_16, 'Checked', 'off');
    set (handles.robot_NO_esferico_11, 'Checked', 'off');
    set (handles.robot_NO_esferico_12, 'Checked', 'off');
    set (handles.robot_esferico_13, 'Checked', 'off');
    %}

update (hObject, handles, handles.q0);

% **** Executes on button press in Home.
function Home_Callback(hObject, eventdata, handles)
% hObject    handle to home (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
% hObject    handle to home (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

update (hObject,handles,handles.q0)

% --- Outputs from this function are returned to the command line.
function varargout = guide_irb120_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% Función UPDATE oontrol en tiempo real del robot desde los sliders
% Sólo actualiza puntos
function update(hObject,handles,punto)
%Función que recoge todos los datos entrada y se los envía al robot
rad= pi/180;
% handles.q --> viene en radianes

% Con la segunda variable mando al robot a esa posición
    if nargin==3, %1 Entro si viene con un punto
        if ishomog(punto),  %1.1 El punto es homogeneo
            if handles.alg_cin_inv==0, %1.1.1 Tengo seleccion geometrico
            q8=ikine_irb120 (punto, handles.tipo_robot); %geométrica
            q_punto=q8(1,:);
            %{
            %compruebo los valores que me entrega
```

```
            for i=1:8
              for j=1:6
               %display (j);
               if q8(i,j)<handles.irb120.qlim (j,1) ||
q8(i,j)>handles.irb120.qlim (j,2);
               break
               else q_punto=q8(i,:);
                   %fprintf('El valor de la articulación %d de la variante %d:\n
%4.1f < %4.1f  < %4.1f \n', j, i,...
                   %rad2deg(handles.irb120.qlim (j,1)),rad2deg(q8(i,j)),
rad2deg(handles.irb120.qlim (j,2)));
                   display (q8(i,:));
               end
                end
               display (q_punto);
               end
            %}
        % fin de ikine_irb120 geometrico
        display (q_punto);
        elseif handles.alg_cin_inv==1,% 1.1.2 Por interpolación (si no es 0)
         q_punto=handles.irb120.ikine (punto,handles.q0,'tol', 0.05);
        else % 1.1.2 Por interpolación (si no es 0, ni 1)
        q_punto=ikine_nuevo (handles.irb120, punto); %interpolación ikine_alberto
        % fin de entrada de punto homogeno
        % Compruebo si el punto que me devuelve no tiene NAN
        end
        if isnan(q_punto),
            return %salgo aparecen NAN
        end
        handles.t=punto; % Punto en coordenadas homogeneas
        punto = q_punto; %Valor del punto en coord. articulares
    %2 El punto entra en articulares
    else
        %1.2 Si el punto es coordenadas articulares
        handles.t =handles.irb120.fkine (punto);
        %hadles.q=punto;
         % Fin de control entrada de puntos
   end

   for i=1:6
         if punto(i)<handles.irb120.qlim (i,1)|| punto(i)>handles.irb120.qlim
(i,2);
            texto=sprintf('El valor de la articulación %d:\n %4.1f < %4.1f  <
%4.1f', i,...
                rad2deg(handles.irb120.qlim (i,1)),rad2deg(punto(i)),
rad2deg(handles.irb120.qlim (i,2)));
            errordlg(texto,'Error: fuera de angulos');
            return;
        end
   end
    handles.q = punto;
    handles.q=handles.q/rad;
    % Muevo los slider a sus valores correspondientes
    for i=1:6
    tx=sprintf('handles.slider%g', i);
    set(eval(tx),'Value', handles.q(i));
    end
     % Coloco el valor junto al slider
    for i=1:6
    tx=sprintf('handles.text_q%g', i);
    set(eval(tx), 'String', sprintf('%.1f', handles.q(i)));
    end
    % set(handles.text_q1, 'String', sprintf('%.1f', handles.q(1)));

else % Si Recojo el valor de cada pulso de slider
```

```
    for i=1:6
    tx=sprintf('handles.slider%g', i);
    handles.q(i)= get(eval(tx), 'Value');
    end
    %handles.q=[q1,q2,q3,q4,q5,q6]; % recojo el valor en grados
    %Este valor viene bien al robot y a los slider
    handles.t= handles.irb120.fkine((handles.q)*rad);

end
    % Actualizo el valor del resto de las casillas y parámetros
    handles.q=handles.q*rad;
    % Coloco el valor de Q bajo los slider
    set(handles.text_Q,'String', num2str(handles.q/rad,'% 5.0f')); % variables
articulares de cada posición

    %display('estoy en el 2 control');
    %display (handles.q) %Asigno valores a coordenadas homogeneas

    %Asignamos valores a las coordenadas del TCP
    set(handles.textx, 'String', sprintf('%.3f', handles.t(1,4)));
    set(handles.texty, 'String', sprintf('%.3f', handles.t(2,4)));
    set(handles.textz, 'String', sprintf('%.3f',handles.t(3,4)));

    % valor del slider mover sobre un eje
    if (handles.mov_eje== handles.man_x)
        valor=handles.t(1,4);
    elseif (handles.mov_eje== handles.man_y)
        valor=handles.t(2,4);
    else
        valor=handles.t(3,4);
    end
    set(handles.slider_mover, 'Value',valor);

    handles.rpy = tr2rpy(handles.t,'deg'); % angulos del TCP
    set(handles.text_roll, 'String', num2str(handles.rpy(1),'%.1f'));
    set(handles.text_pich, 'String', num2str(handles.rpy(2),'%.1f'));
    set(handles.text_yaw, 'String', num2str(handles.rpy(3),'%.1f'));

    % Actualizamos las coordenas del TCP
    set(handles.text_tcp,'String', trprint(handles.t,'fmt', '%.1f'));
    mnp_t=handles.irb120.maniplty(handles.q, handles.met_mnpy); %Manipulabilidad
del punto
    set(handles.text_mnp_t,'String', sprintf('%.3f', mnp_t));
 % Dibujo el robot para cada nueva posición
    if handles.tipo_robot>=14
        plot_irb120(handles.q,handles);
        %handles.irb120.plot(handles.q); % ***** OJO !!!!
    else handles.irb120.plot(handles.q);
    end;
guidata(hObject,handles)

% --- Executes on button press in grabar_articulares.
function grabar_articulares_Callback(hObject, eventdata, handles)
% hObject    handle to grabar_articulares (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes on button press in home.
% Cazo el punto handles.q

handles.Q=[handles.Q;handles.q];% Construyo la matriz Q, con los sucesivos q
f=size (handles.Q,1); % Filas de la matriz Q
%handles.ejes_coord=g; % Indico que hay punto grabados
t=handles.irb120.fkine(handles.q);
```

```
handles.puntos_grabados=cat (3,handles.puntos_grabados,t); % Le añado el nuevo
punto
if ~isempty (handles.Q)
     handles.mnpQ=handles.irb120.maniplty(handles.Q, handles.met_mnpy);
     %display(handles.mnpQ)
     set(handles.text_mnp_Q,'String', sprintf('%.3f', min(handles.mnpQ)));
end
switch f; %opcion el numero de case
case 1;
   set(handles.text_pa1,'String',num2str(rad2deg(handles.q),'% 5.0f'));
case 2;
   set(handles.text_pa2,'String', num2str(rad2deg(handles.q),'% 5.0f'));
case 3;
   set(handles.text_pa3,'String', num2str(rad2deg(handles.q),'% 5.0f'));
case 4;
   set(handles.text_pa4,'String',num2str(rad2deg(handles.q),'% 5.0f'));
case 5;
   set(handles.text_pa5,'String', num2str(rad2deg(handles.q),'% 5.0f'));
otherwise;
   warndlg('Se registró el punto','Información');
end
update(hObject, handles, handles.q)

% --- Grabar en coordenadas homogeneas
function grabar_homogeneas_Callback(hObject, eventdata, handles)
% hObject    handle to grabar_homogeneas (see GCBO)
%t=handles.irb120.fkine(handles.q);% Saco el punto t0 en MH con cinematica
directa
%x =t(1,4); % Añado el valor x,y,z de la matriz homogenea t0
%y =t(2,4);
%z =t(3,4);
%hold on;
%plot3 ( x , y , z, 'b+' ); % Dibujo el punto (está en la matriz x)
handles.T=cat (3,handles.T,handles.t);
handles.puntos_grabados=cat (3,handles.puntos_grabados,handles.T);
p=size (handles.T,3); % El tamaño es el numero del punto
%display (t);
    %manipulabilidad de la trayectoria
if ~isempty (handles.q)
     mnpT=handles.irb120.maniplty(handles.q, handles.met_mnpy);
     handles.mnpT=[handles.mnpT, mnpT];
     %display(handles.mnpT)
     set(handles.text_mnp_H,'String', sprintf('%.3f', min(handles.mnpT)));
end
switch p; %opcion el numero de case
case 1;
   set(handles.text_ph1,'String', trprint(handles.t,'fmt', '%.1f'));
case 2;
    set(handles.text_ph2,'String', trprint(handles.t,'fmt', '%.1f'));
case 3;
    set(handles.text_ph3,'String', trprint(handles.t,'fmt', '%.1f'));
case 4;
   set(handles.text_ph4,'String', trprint(handles.t,'fmt', '%.1f'));
case 5;
   set(handles.text_ph5,'String', trprint(handles.t,'fmt', '%.1f'));
otherwise;
  warndlg('Se registró el punto','Información');
end
guidata(hObject, handles)

% --- Movimiento de los SLIDERS
function slider1_Callback(hObject, eventdata, handles)
% hObject    handle to slider1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.text_q1, 'String', sprintf('%.1f', get(hObject, 'Value')));
update(hObject, handles)

function slider2_Callback(hObject, eventdata, handles)
% hObject    handle to slider2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.text_q2, 'String', sprintf('%.1f', get(hObject, 'Value')));
update(hObject, handles)

function slider3_Callback(hObject, eventdata, handles)
% hObject    handle to slider3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.text_q3, 'String', sprintf('%.1f', get(hObject, 'Value')));
update(hObject, handles)

function slider4_Callback(hObject, eventdata, handles)
% hObject    handle to slider4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.text_q4, 'String', sprintf('%.1f', get(hObject, 'Value')));
update(hObject, handles)

function slider5_Callback(hObject, eventdata, handles)
% hObject    handle to slider5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
set(handles.text_q5, 'String', sprintf('%.1f', get(hObject, 'Value')));
update(hObject, handles)

function slider6_Callback(hObject, eventdata, handles)
% hObject    handle to slider5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider

set(handles.text_q6, 'String', sprintf('%.1f', get(hObject, 'Value')));
% con el valor del sliders muevo el robot
update(hObject, handles)

% Valor de angulo de las sucesivas articulaciones
function text_q1_Callback(hObject, eventdata, handles)
% hObject    handle to q1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 q = str2double(get(hObject, 'String')); % lo cojo en grados
```

```
 qr=q*pi/180; % lo paso a radianes
 if qr<handles.irb120.qlim (1,1)|| qr>handles.irb120.qlim (1,2);
     errordlg('El valor está fuera de límites (OUT-Limits) -->q previo','Error')
     set(handles.slider1, 'Value',(handles.q(1)*180/pi));
     set(handles.text_q1, 'String', sprintf('%.1f', (handles.q(1)*180/pi)));
     return;
 end;
 set(handles.slider1, 'Value',q);
 handles.q(1)=q;
 update(hObject, handles)


function text_q2_Callback(hObject, eventdata, handles)
% hObject    handle to Y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 q = str2double(get(hObject, 'String')); % lo cojo en grados
 qr=q*pi/180; % lo paso a radianes
 if qr<handles.irb120.qlim (2,1)|| qr>handles.irb120.qlim (2,2);
     errordlg('El valor está fuera de límites (OUT-Limits) -->q previo','Error')
     set(handles.slider2, 'Value',(handles.q(2)*180/pi));
     set(handles.text_q2, 'String', sprintf('%.1f', (handles.q(2)*180/pi)));
     return;
 end;
 set(handles.slider2, 'Value',q);
 handles.q(2)=q;
 update(hObject, handles)

function text_q3_Callback(hObject, eventdata, handles)
% hObject    handle to X (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 q = str2double(get(hObject, 'String')); % lo cojo en grados
 qr=q*pi/180; % lo paso a radianes
 if qr<handles.irb120.qlim (3,1)|| qr>handles.irb120.qlim (3,2);
     errordlg('El valor está fuera de límites (OUT-Limits) -->q previo','Error')
     set(handles.slider3, 'Value',(handles.q(3)*180/pi));
     set(handles.text_q3, 'String', sprintf('%.1f', (handles.q(3)*180/pi)));
     return;
 end;
 set(handles.slider3, 'Value',q);
 handles.q(3)=q;
 update(hObject, handles)

function text_q4_Callback(hObject, eventdata, handles)
% hObject    handle to Z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 q = str2double(get(hObject, 'String')); % lo cojo en grados
 qr=q*pi/180; % lo paso a radianes
 if qr<handles.irb120.qlim (4,1)|| qr>handles.irb120.qlim (4,2);
     errordlg('El valor está fuera de límites (OUT-Limits) -->q previo','Error')
     set(handles.slider4, 'Value',(handles.q(4)*180/pi));
     set(handles.text_q4, 'String', sprintf('%.1f', (handles.q(4)*180/pi)));
     return;
 end;
 set(handles.slider4, 'Value',q);
 handles.q(4)=q;
 update(hObject, handles)

 function text_q5_Callback(hObject, eventdata, handles)
% hObject    handle to q2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
q5 = str2double(get(hObject, 'String'));
if isnan(q5) %si no es número,
    errordlg('Debe introducir un numero (grados)','Error');
    set(hObject, 'String', 0);%Lo pone en cero
end
  set(handles.slider5, 'Value',q5);
  q5=q5*pi/180;
  handles.q(5)=q5;
update(hObject, handles)

function text_q6_Callback(hObject, eventdata, handles)
% hObject    handle to q3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

 q = str2double(get(hObject, 'String')); % lo cojo en grados
 qr=q*pi/180; % lo paso a radianes
 if qr<handles.irb120.qlim (6,1)|| qr>handles.irb120.qlim (6,2);
     errordlg('El valor está fuera de límites (OUT-Limits) -->q previo','Error')
     set(handles.slider6, 'Value',(handles.q(6)*180/pi));
     set(handles.text_q6, 'String', sprintf('%.1f', (handles.q(6)*180/pi)));
     return;
 end;
 set(handles.slider6, 'Value',q);
 handles.q(6)=q;
 update(hObject, handles)

 % Introduzco valores por teclado de la posición angular Q
function text_Q_Callback(hObject, eventdata, handles)
% hObject    handle to text_Q (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of text_Q as text
%         returns contents of text_Q as a double

q=(get(hObject, 'String')); % Recoge la matriz del string
display (q);
if isempty(q)  %si no es array vacio,
    errordlg('Introducir los 6 angulos de cada articulación [°])','Error');
    set(hObject, 'String', [0 0 0 0 0 0]);%Lo pone en cero
end
q=str2num(q); %Lo cojo en grados
display (q);
punto= q*pi/180; %(lo paso a radianes)
%set(handles.text_Q, 'Value',q);
update(hObject, handles, punto)


 % Coger el valor de x desde el teclado
function textx_Callback(hObject, eventdata, handles)
% hObject    handle to textx (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

x = str2double(get(hObject, 'String'));

if x<-0.65 || x >0.65 || isnan(x);
     errordlg('El valor está fuera de límites (OUT-Limits) -->x previo','Error')
     set(handles.textx, 'String', sprintf('%.1f', (handles.t (1,4))));
     return;
 end;
handles.t(1,4)=x;
```

```
update(hObject, handles, handles.t)

function texty_Callback(hObject, eventdata, handles)
% hObject    handle to texty (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

y = str2double(get(hObject, 'String'));

if y<-0.65 || y >0.65 || isnan(y);
     errordlg('El valor está fuera de límites (OUT-Limits) -->x previo','Error')
     set(handles.texty, 'String', sprintf('%.1f', (handles.t (2,4))));
     return;
end;

handles.t(2,4)=y;
update(hObject, handles, handles.t)

function textz_Callback(hObject, eventdata, handles)
% hObject    handle to textz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of textz as text
%        str2double(get(hObject,'String')) returns contents of textz as a double
 z = str2double(get(hObject, 'String'));
 display (z);
if z<-0.2 || z >1.1 || isnan(z);
     errordlg('El valor está fuera de límites (OUT-Limits) -->x previo','Error')
     set(handles.textz, 'String', sprintf('%.1f', (handles.t (3,4))));
     return;
 end;
 handles.t(3,4)=z;
update(hObject, handles, handles.t)

% --- Executes on button press in exit.
function exit_Callback(hObject, eventdata, handles)
% hObject    handle to exit (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
quit;

function text_yaw_Callback(hObject, eventdata, handles)
% hObject    handle to text_yaw (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% DEMOSTRACIÓN trazar poligono
function demo_trazar_poligonal_Callback(hObject, eventdata, handles)
% hObject    handle to demo_trazar_poligonal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1=[-.2, .3, 0];
p2=[.3, .3, 0];
p3=[.3, -.3, 0];
p4=[.0, -.5, 0];
p5=[0, -.5, .56];
%p6=[.3, -.3, .6];
%p7=[.5, 0, .6];
%p8=[.3, .3, 0.6];
%p9=[.0, .0, 0.8];
%p10=p1;
%p10=[0.1, .3, .7];
```

```
P=[p1; p2; p3; p4; p5]; %p5]; %; p6;p7;p8;p9;p10];
handles.demos=P;
line (P(:,1), P(:,2), P(:,3),'LineWidth', 1, 'Marker','+');%dibujo linea 3D
%display (P);
n=size (P,1);
Q=[];
for i=1:n-1 % bucle para cada par de puntos
        %p1=P(i,:); % punto 1
        %p2=P((i+1),:); % punto 2
        if i>3, %&& i~=n,
             angulo=pi/2;
        else angulo=pi;
        end;
        t1= transl(P(i,:)) * troty (angulo); % giro el punto sobre eje y
(funciona mejor)
        t2= transl(P(i+1,:)) * troty (angulo);% cojo los puntos para dibujarlos
despues (ver puntos grabados)
        t = ctraj (t1, t2, 5); % variables art. para alcanzar t1
        for j=1:5
        %display (t);
         if handles.alg_cin_inv==0;
            q=ikine_irb120 (t(:,:,j), handles.tipo_robot); %geométrica
             q=q(1,:); %Cojo la primera, que es la mejor
            %display (q_punto);
        elseif handles.alg_cin_inv==1% Por interpolación (si no es 0)
        q=handles.irb120.ikine (t(:,:,j),handles.q0,'tol', 0.02);
        else q=ikine_nuevo (handles.irb120, punto); %interpolación ikine_alberto
        end
        Q=[Q;q(1,:)];
        end;
        %handles.puntos_grabados=cat (3,handles.puntos_grabados,t1);
end
%display(Q);
trazar_Q(Q, handles)

% --------------------------------------------------------------------
function demo_trazar_multipuntos_Callback(hObject, eventdata, handles)
% hObject    handle to demo_trazar_multipuntos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
p1=[-.3, .3, 0];
p2=[.3, .3, 0];
p3=[.3, -.3, 0];
p4=[-.2, -.5, 0];
p5=[-.2, -.5, .4];
p6=[.3, -.3, .6];
p7=[.5, 0, .6];
p8=[.3, .3, 0.6];
p9=[.0, .2, 0.8];
p10=p1;
%p10=[0.1, .3, .7];
P=[p1; p2; p3; p4; p5; p6;p7;p8;p9;p10];
handles.demos=P;
line (P(:,1), P(:,2), P(:,3),'LineWidth', 1, 'Marker','+');%dibujo linea 3D
%display (P);
n=size (P,1);
Q=[];
for i=1:n % bucle para cada par de puntos
        %p1=P(i,:); % punto 1
        %p2=P((i+1),:); % punto 2
        if i>4 && i~=n,
             angulo=pi/2;
        else angulo=pi;
        end;
```

```
        t= transl(P(i,:)) * troty (angulo); % giro el punto sobre eje y (funciona
mejor)
        handles.puntos_grabados=cat (3,handles.puntos_grabados,t);
        %t(i+1)= transl(P(i+1,:)) * troty (angulo); % giro el punto sobre eje y
        q = ikine_irb120 (t,handles.tipo_robot); % variables art. para alcanzar
t1
        %q2 = ikine_irb120 (t2);
        %Q_tray= jtraj(q1(1,:),q2(1,:),2); %Trayectoria en variables articulares
        %display (Q_tray);
        %Q_tray=[q1(1,:);q2(1,:)];
        Q=[Q;q(1,:)];% Construyo la matriz Q, con los sucesivos q
end
%display(Q);
trazar_Q (Q, handles);

% DEMOSTRACIÓN: Multiples posiciones de un punto con ikine geométrico
function demo_cin_inversa_punto_Callback(hObject, eventdata, handles)
% hObject    handle to demo_cin_inversa_punto (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%p1 = get(handles.t, 'Value'); % le cojo
if handles.alg_cin_inv~=0; % solo si tengo seleccionado ikine geometrico
    errordlg('Sólo con cinemática inversa geométrica--> seleccione Ikine
geometrico','Error demo')
else
    q=handles.q;
    q8=ikine_irb120 (handles.t,handles.tipo_robot); % Cojo el punto en
coordenadas homogeneas
    display (q8);
    %k=0;
    for i=1:8 % las 8 posiciones de un punto
        %display (rad2deg(q8(i,:)));
        if handles.tipo_robot>=14
            plot_irb120(q8(i,:),handles);
        else handles.irb120.plot(q8(i,:));
        end
        %p= repmat(' ', [30,40]); %Alberto, dimensión array de texto
        for j=1:6 % miro los valores que obtengo
        if q8(i,j)<handles.irb120.qlim (j,1)|| q8(i,j)>handles.irb120.qlim (j,2);
          %k=k+1;
          fprintf('Articulación %d, variante %d :\n %4.1f < %4.1f  < %4.1f, fuera
de rango\n', j, i,...
                  rad2deg(handles.irb120.qlim (j,1)),rad2deg(q8(i,j)),
rad2deg(handles.irb120.qlim (j,2)));
        end;
        end
        pause(1.5);
    end
end
pause (5);
%set (handles.text_para_robot, 'Visible', 'off');
update(hObject, handles, q);
%vuelve donde estaba

 % DESMOSTRACIÓN: trazar círculo
function demo_trazar_circulo_Callback(hObject, eventdata, handles)
% hObject    handle to demo_trazar_circulo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
C=[0,0,0.3]; %centro del cículo
R=0.4; %radio del circulos;
P=circle(C, R, 'n', 6); % handles.velocidad);  % 30 el número de puntos
plot_circle(C, R); %'edgecolor','b'); % Tengo los xyz
display (P);
%saco la orientación de la muñeca
```

```
%t=handles.t(1:3,1:3);
display (t);
T=[];
Q=[];
n=size(P,2);%numero de puntos de la circunferencia
G=[-0.33 0 0.94 0.5; 0 1 0 0.5; -0.94 0 -0.33 0.5; 0 0 0 1 ]; % giro de la muñeca
%unos=ones(1,4);
for i=1:n % numero de puntos de la circunferencia
    T(:,:,i)=G*transl(P(:,1))';
    q8= ikine_irb120 (T(:,:,i), handles.tipo_robot);
    %display(q8);
    Q=[Q;q8(1,:)];
    %display (Q);
    %T(:,:,i)=* troty (pi);% *trotz (i*0,25*pi)
end
for i=1:n-1 %el punto 1 es el q0
     Q_tray = jtraj(Q(i,:),Q(i+1,:),handles.velocidad); %Trayectoria en variables
articulares
     if handles.tipo_robot>=14 %||handles.tipo_robot==15 ;  % Si le robot es
CAD3d
          plot_irb120(Q_tray,handles);
     else handles.irb120.plot(Q_tray);%Cuando el robot sea de palillo
     end
end

%for i=1:n-1;
%T=handles.irb120.jtraj(T(:,:,i), T(:,:,i+1), 2);
%trazar_Q(Q, handles);
%end

%***DEMOSTRACIÓN Movimiento del robot sólo con una coordenada
function demo_mov_robot_Callback(hObject, eventdata, handles)

QLR=handles.irb120.qlim; %valores de la posiciónes límites
%QLR=Q; % Q es la matriz de varias posiciones del robot
%end
rad= pi/180;
% Muevo el robot, articulación por articulación, entre valores límites
n=20; % divido el angulo de las  articulaciones
% Bucles para mover el robot y dibujar los puntos
for j=1:6 % Desde el eje 1 hasta el eje 6
    q0=handles.q0;
    x=[]; y=[]; z=[]; % Declaro las variables vacias
    for qi=QLR(j,1):rad*n:QLR(j,2)
    q0(j)=qi;
    if handles.tipo_robot>=14;
        plot_irb120(q0,handles);
        else handles.irb120.plot(q0);
    end
    t0=handles.irb120.fkine(q0);% Saco el punto t0 en MH con cinematica directa
    x =[x, t0(1,4)]; % Añado el valor x,y,z de la matriz homogenea t0
    y =[y, t0(2,4)];
    z =[z, t0(3,4)];
    title('Movimientos del robot matriz QLR');
    end
    %hold on;
    plot3 ( x , y , z, '-mx' ); % Dibujo la trayectoria (está en la matriz x)
    pause(1);
end

% DEMOSTRACION: Espacio de trabajo
function demo_esp_trabajo_Callback(hObject, eventdata, handles)
% hObject    handle to demo_esp_trabajo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
espacio_trabajo_lin(handles.q,handles.irb120);

% DEMOSTRACIÓN: Alcance máximo
function demo_alc_max_Callback(hObject, eventdata, handles)
% hObject    handle to demo_alc_max (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% QLG= [-165, 165;-110, 110; -110, 70; -160, 160; -120, 120; -400, 400];
n=20; % n° de particiones
for i= 1:n+1,
    q1=deg2rad(-165+ (i-1)*( 165/n)); % Empiezo en el ángulo más pequeño
    for j= 1:n+5,
      q2=deg2rad(120-(j-1)*(110/n));
      %for k=1:2,
          %if k==1, q3=deg2rad(-30);
      q3=deg2rad(-60);
          %end
      %for e3=QLR(3,1):rad*n:QLR(3,2)% cuanto mayor numero, menos tarda
      TR = handles.irb120.fkine([q1 q2 q3  0 0 0]);% los ejes 4 y 6 no aumentan
      TR(:,1:3)=[];% Elimino todos los elementos de la columnas de 1 a 3
      TR(4,:)=[]; % elimino todos los elementos de la fila 4
      puntos(i,j,:) = TR;
      %end
    end

end
surf(puntos(:,:,1), puntos(:,:,2), puntos(:,:,3)); % todos

function simulink_Callback(hObject, eventdata, handles)
% hObject    handle to simulink (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
    % --------------------------------------------------------------------

function simulink_cargar_modelo_Callback(hObject, eventdata, handles)
% hObject    handle to simulink_cargar_modelo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if  isempty (find_system('Name','irb120_simulink')),
open_system('irb120_simulink'); %cargo el modelo
set_param(gcs,'SimulationCommand','Start'); % Inicio la simulación
set_param('irb120_simulink/Constant6','Value', hadles.text_pa1);
set_param('irb120_simulink/Constant7','Value', hadles.text_pa2);
set_param('irb120_simulink/Constant1','Value', hadles.text_pa3);

%set_param('f14/Controller/Gain','Position',[275 14 340 56])
    %figure(handles.F14ControllerEditor)
    % Put  values of Kf and Ki from the GUI into the Block dialogs
    %set_param('f14/Controller/Gain','Gain',...
     %                       get(handles.KfCurrentValue,'String'))
    %set_param('f14/Controller/Proportional plus integral compensator',...
     %       'Numerator',...
    %get(handles.KiCurrentValue,'String'))

  %model_open(handles)

  % Get the new value for the Kf Gain from the slider
  %NewVal = get(hObject,'Value');

  end

function robot_opc_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to robot_opc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% --- Executes during object creation, after setting all properties.
 handles.time_opc= str2double(get(hObject, 'String')); % lo cojo de la etiqueta
 display (handles.time_opc);
 update(hObject, handles)

function text_time_opc_Callback(hObject, eventdata, handles)
% hObject    handle to text_time_opc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 handles.time_opc= str2double(get(hObject, 'String')); % lo cojo de la etiqueta
 display (handles.time_opc);
 update(hObject, handles)

% Sucesiva funciones para cargar el ROBOT
function robot_NO_esferico_11_Callback(hObject, eventdata, handles)
% hObject    handle to robot_NO_esferico_11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.robot_NO_esferico_11, 'Checked', 'on');
set (handles.robot_NO_esferico_12, 'Checked', 'off');
set (handles.robot_esferico_13, 'Checked', 'off');
set (handles.robot_14, 'Checked', 'off');
set (handles.robot_15, 'Checked', 'off');
set (handles.robot_16, 'Checked', 'off');
handles.tipo_robot=11;
cla;
handles.irb120=mdl_irb120(handles.tipo_robot);
update(hObject, handles)

function robot_NO_esferico_12_Callback(hObject, eventdata, handles)
% hObject    handle to robot_NO_esferico_12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.robot_NO_esferico_12, 'Checked', 'on');
set (handles.robot_NO_esferico_11, 'Checked', 'off');
set (handles.robot_esferico_13, 'Checked', 'off');
set (handles.robot_14, 'Checked', 'off')
set (handles.robot_15, 'Checked', 'off');
set (handles.robot_16, 'Checked', 'off');
handles.tipo_robot=12;
cla;
handles.irb120=mdl_irb120(handles.tipo_robot);
update(hObject, handles)

function robot_esferico_13_Callback(hObject, eventdata, handles)
% hObject    handle to robot_esferico_13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.robot_esferico_13, 'Checked', 'on');
set (handles.robot_14, 'Checked', 'off');
set (handles.robot_15, 'Checked', 'off');
set (handles.robot_16, 'Checked', 'off');
set (handles.robot_NO_esferico_12, 'Checked', 'off');
set (handles.robot_NO_esferico_11, 'Checked', 'off');
handles.tipo_robot=13;
cla;
handles.irb120=mdl_irb120(handles.tipo_robot);
update(hObject, handles)


function robot_14_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to robot_14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.robot_14, 'Checked', 'on');
set (handles.robot_15, 'Checked', 'off');
set (handles.robot_16, 'Checked', 'off');
set (handles.robot_NO_esferico_11, 'Checked', 'off');
set (handles.robot_NO_esferico_12, 'Checked', 'off');
set (handles.robot_esferico_13, 'Checked', 'off');
handles.tipo_robot=14;
cla;
handles.irb120=mdl_irb120(handles.tipo_robot);
update(hObject, handles)

function robot_15_Callback(hObject, eventdata, handles)
% hObject    handle to robot_15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.robot_15, 'Checked', 'on');
set (handles.robot_16, 'Checked', 'off');
set (handles.robot_14, 'Checked', 'off');
set (handles.robot_esferico_13, 'Checked', 'off');
set (handles.robot_NO_esferico_12, 'Checked', 'off');
set (handles.robot_NO_esferico_11, 'Checked', 'off');
handles.tipo_robot=15;
cla;
handles.irb120=mdl_irb120(handles.tipo_robot);
update(hObject, handles)

function robot_16_Callback(hObject, eventdata, handles)
% hObject    handle to robot_16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
set (handles.robot_16, 'Checked', 'on');

set (handles.robot_15, 'Checked', 'off');
set (handles.robot_14, 'Checked', 'off');
set (handles.robot_esferico_13, 'Checked', 'off');
set (handles.robot_NO_esferico_12, 'Checked', 'off');
set (handles.robot_NO_esferico_11, 'Checked', 'off');
handles.tipo_robot=16;
cla;
handles.irb120=mdl_irb120(handles.tipo_robot);
% handles.irb120=mdl_irb120(12); % OJO, para verle transparente !!!
update(hObject, handles)

function text_roll_Callback(hObject, eventdata, handles)
% hObject    handle to text_roll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

function text_pich_Callback(hObject, eventdata, handles)
% hObject    handle to text_pich (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%yaw = str2double(get(hObject, 'String'));
%if isnan(yaw) %si no es número,
 %   errordlg('Debe introducir un número [°]','Error');
  %  set(hObject, 'String', 0);%Lo pone en cero
%end
%handles.rpy(1)=yaw;
%handles.t=rpy2tr(handles.rpy);
%update(hObject, handles,handles.t)
```

```
% TRAZAR en coordenadas articulares
function trazar_art_Callback(hObject, eventdata, handles)
% hObject    handle to trazar_art (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%display (handles.Q)
%tray_puntos (irb120,tipo_robot, Q)
trazar_Q (handles.Q, handles);

function trazar_Q (Q, handles)
%** Trazar trayectorias
n=size(Q,1); % Admite más puntos de los que visualizo
if n<=1
    errordlg('Tiene que haber 2 puntos grabados -->Grabar Art','Error');
else
%Dibujo puntos para trazar
TQ=handles.irb120.fkine (Q);
if ~handles.ver_puntos_grabados %Si no estoy viendo los puntos grabados
    for i=1:n % Dibujo los puntos
    trplot(TQ(:,:,i), 'frame', ['P',num2str(i)], 'length',0.15,...
        'width', 1.2 , 'rgb', 'thick', 1.2); %punto que tiene q
    end
pause (1);
end
% Trazado de la trayectoria por los  puntos
for i=1:n-1 %el punto 1 es el q0
     Q_tray = jtraj(Q(i,:),Q(i+1,:),handles.velocidad); %Trayectoria en variables
articulares
     if handles.tipo_robot>=14 %||handles.tipo_robot==15 ;  % Si le robot es
CAD3d
          plot_irb120(Q_tray,handles);
     else handles.irb120.plot(Q_tray);%Cuando el robot sea de palillo
     end
end
end

% TRAZAR en coordenadas homogeneas
function trazar_hm_Callback(hObject, eventdata, handles)
% hObject    handle to trazar_hm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%display (handles.T)
%tray_puntos (irb120,tipo_robot, Q)
n=size(handles.T,3);
if n<=1
    errordlg('Tiene que haber 2 puntos grabados -->Grabar Hom','Error');
else trazar_T(handles, handles.T)
end

% --- Trazo la trayectoria desde puntos dados en coordenadas homogeneas
function trazar_T(handles,T)
% hObject    handle to trazar_hm (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%display (handles.T)
%tray_puntos (irb120,tipo_robot, Q)
n=size(T,3);
if n<=1
    errordlg('Tiene que haber 2 puntos grabados -->Grabar Art','Error');
else
if ~handles.ver_puntos_grabados
for i=1:n %Marco los puntos a los que tiene que ir
   trplot(T(:,:,i), 'frame', ['P',num2str(i)], 'length',0.15,'width', 1 , 'rgb',
'thick', 1);
end
```

```
  pause(1)
end
    %*****Dibujo el robot
for i=1:(n-1) %el punto 1 es el q0
    % Con JTRJ sale lenta y con errores
    Q_tray =handles.irb120.jtraj(T(:,:,i),T(:,:,i+1),handles.velocidad);
      if handles.tipo_robot>=14; plot_irb120(Q_tray,handles);
            else handles.irb120.plot(Q_tray);
      end
end
end

% TRAZAR desde fichero
function demo_trazar_fichero_Callback(hObject, eventdata, handles)
% hObject    handle to demo_trazar_fichero (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%helpdlg('Se trazará el fichero -datos.txt-', 'demo_trazar_fichero');
   %nom_fich=inputdlg('Nombre de fichero con trayectoria -sin extension-',
'(datos.txt)');%, 1, 'datos.txt','tex');%,('%s.%s',nom_fich,'txt');
   % if isempty(nom_fich);
 nom_fich = 'datos.txt';
    %else nom_fich=sprintf('%s.%s',nom_fich,'txt');
fid=fopen (nom_fich); % Guardar en un fichero nombre.txt
Q_imp = importdata(nom_fich);
disp (Q_imp);

if fclose ( fid )==-1;
     errordlg('Se ha producido un error -->Repita SAVE','Error');
     %else fprintf ('...Grabación correcta en fichero: %s. Pulse una tecla..\n',
nomb);
else
    trazar_Q(Q_imp, handles);% le paso la trayectoria grabada
end

% Ir al punto guardado en coordenadas articulares
function pa1_Callback(hObject, eventdata, handles)
% hObject    handle to pa1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.Q,1);
if n>=1; q=handles.Q(1,:);
    t=handles.irb120.fkine(q);
    trplot(t, 'frame', 'P1', 'length',0.15,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(1);
    update(hObject,handles,q);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

function pa2_Callback(hObject, eventdata, handles)
% hObject    handle to PA3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.Q,1);
if n>=2; q=handles.Q(2,:);
    t=handles.irb120.fkine(q);
     trplot(t, 'frame', 'P2', 'length',0.15,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
      pause(1);
    update(hObject,handles,q);
else
```

```
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

function pa3_Callback(hObject, eventdata, handles)
% hObject    handle to pa3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.Q,1);
if n>=3; q=handles.Q(3,:);
    t=handles.irb120.fkine(q);
    trplot(t, 'frame', 'P3', 'length',0.15,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(1);
    update(hObject,handles,q);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end


function pa4_Callback(hObject, eventdata, handles)
% hObject    handle to pa4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.Q,1);
if n>=4; q=handles.Q(4,:);
    t=handles.irb120.fkine(q);
    trplot(t, 'frame', 'P1', 'length',0.15,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(1);
    update(hObject,handles,q);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

function pa5_Callback(hObject, eventdata, handles)
% hObject    handle to pa5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.Q,1);
if n>=5; q=handles.Q(5,:);
    t=handles.irb120.fkine(q);
    trplot(t, 'frame', 'P5', 'length',0.15,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(1);
    update(hObject,handles,q);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

% SAVE_Q: función salvar la trayectoria en coordenadas articulares
function save_Q_Callback(hObject, eventdata, handles)
% hObject    handle to save_Q (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if isempty(handles.Q)
    errordlg('No existen puntos grabados!','Error');% Si no hay datos en
handles.Q
else
    f=save_trayec(handles.Q);
    if f==-1;
     errordlg('Se ha producido un error -->Repita SAVE','Error');
     %else fprintf ('...Grabación correcta en fichero: %s. Pulse una tecla..\n',
nomb);
    end
```

```
end

% Ir a los puntos de la coordenadas Homogeneas guardados
function ph1_Callback(hObject, eventdata, handles)
% hObject    handle to ph1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
t=handles.T(:,:,1);
%display(handles.q);
%display(t);
if isempty (t);
    errordlg('No hay punto grabado -->Grabar Hom','Error');
else
    trplot(t, 'frame', 'P2', 'length',0.17,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(0.5);
    update(hObject,handles,t);
end

function ph2_Callback(hObject, eventdata, handles)
% hObject    handle to ph2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.T,3);
if n>=2;
    t=handles.T(:,:,2);
     trplot(t, 'frame', 'P2', 'length',0.17,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(.5);
    update(hObject,handles,t);
else
   errordlg('No hay punto grabado -->Grabar Hom','Error');
end

function ph3_Callback(hObject, eventdata, handles)
% hObject    handle to ph3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.T,3);
if n>=3; t=handles.T(:,:,3);
     trplot(t, 'frame', 'P3', 'length',0.17,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(0.5);
    update(hObject,handles,t);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

function ph4_Callback(hObject, eventdata, handles)
% hObject    handle to ph4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.T,3);
if n>=4;
    t=handles.T(:,:,4);
    trplot(t, 'frame', 'P4', 'length',0.17,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(0.5);
    update(hObject,handles,t);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

function ph5_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to ph5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
n=size(handles.T,3);
if n>=5;
    t=handles.T(:,:,5);
    trplot(t, 'frame', 'P5', 'length',0.17,'width', 1 , 'rgb', 'thick', 1);
%punto que tiene que
    pause(1);
    update(hObject,handles,t);
else
    errordlg('No hay punto grabado -->Grabar Hom','Error');
end

% SAVE_T: salvar en coordenadas articulares
function save_T_Callback(hObject, eventdata, handles)
% hObject    handle to save_T (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if isempty(handles.T)
    errordlg('No existen puntos grabados!','Error');% Si no hay datos en
handles.Q
else
    f=save_trayec(handles.T);
    if f==-1;
     errordlg('Se ha producido un error -->Repita SAVE','Error');
     %else fprintf ('...Grabación correcta en fichero: %s. Pulse una tecla..\n',
nomb);
    end
end

%SLIDER mover (cinemática inversa)
function slider_mover_Callback(hObject, eventdata, handles)
% hObject    handle to slider_mover (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

   punto=(get(hObject, 'value'));% cojo el valor de slider mover en punto

if (handles.mov_eje == handles.man_x) % Si muevo sobre el eje X
    handles.t(1,4)=punto; % le doy el valor de x
    %handles.tx=handles.t;
    punto= handles.t; %Lo convierto en un punto homogeneo
    %display ('punto homog');
    %display (handles.tx);
    set(handles.slider_mover,'Min', -0.65) ;
    set(handles.slider_mover,'Max',0.65);
    set(handles.slider_mover, 'Value',handles.t(1,4));

elseif (handles.mov_eje == handles.man_y)
    handles.t(2,4)=punto;
    %handles.ty=handles.t;
    punto= handles.t;
    set(handles.slider_mover,'Min', -0.65) ;
    set(handles.slider_mover,'Max',0.65);
    set(handles.slider_mover, 'Value',handles.t(2,4));
else
    handles.t(3,4)=punto;
    %handles.tz=handles.t;
    punto= handles.t;
    set(handles.slider_mover,'Min', -0.2) ;
    set(handles.slider_mover,'Max',1.1);
    set(handles.slider_mover, 'Value',handles.t(3,4));
end
```

```
update(hObject, handles, punto)


% Marcar coordenada movimiento.
function marcar_coord_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in marcar_coord
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
if (hObject == handles.man_x)
    handles.mov_eje = handles.man_x;
    set(handles.slider_mover,'Min', -0.65) ;
    set(handles.slider_mover,'Max',0.65);
    set(handles.slider_mover, 'Value',handles.textx);
elseif (hObject == handles.man_y)
    handles.mov_eje = handles.man_y;
    set(handles.slider_mover,'Min', -0.65) ;
    set(handles.slider_mover,'Max',0.65);
    set(handles.slider_mover, 'Value',handles.texty);
elseif (hObject == handles.man_z)
     handles.mov_eje= handles.man_z;
     set(handles.slider_mover,'Min', -0.2) ;
     set(handles.slider_mover,'Max',1);
     set(handles.slider_mover, 'Value',handles.textz);
end
update (hObject, handles)

% --- Panel de selección de algoritmo cinemática inversa
function panel_alg_cin_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in panel_alg_cin
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
if (hObject == handles.ikine_geo)
    handles.alg_cin_inv=0;
    display (handles.alg_cin_inv);
elseif  (hObject == handles.ikine_ite)
    handles.alg_cin_inv=1;
    display (handles.alg_cin_inv);
else
     handles.alg_cin_inv=2;
     display (handles.alg_cin_inv);
end
update (hObject, handles)

% Panel selección de algoritmo de manipulación
function panel_alg_mnpy_SelectionChangeFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in panel_alg_mnpy
% eventdata  structure with the following fields (see UIBUTTONGROUP)
%   EventName: string 'SelectionChanged' (read only)
%   OldValue: handle of the previously selected object or empty if none was
selected
%   NewValue: handle of the currently selected object
% handles    structure with handles and user data (see GUIDATA)
if (hObject == handles.yoshikawa)
    handles.met_mnpy = 'yoshikawa';
else
```

```
    handles.met_mnpy = 'asada';
end

update (hObject, handles)


% VER PARÁMETROS: matrices de transformación
function boton_ver_MT_Callback(hObject, eventdata, handles)
% hObject    handle to boton_ver_MT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value') == get(hObject,'Max'))
    set (handles.text_para_robot, 'Visible', 'on');
    set(handles.boton_ver_dh,'Value', 0);
    set(handles.boton_ver_dyn,'Value', 0);
    [Tn, T] = handles.irb120.fkine(handles.q);
    p= repmat(' ', [30,60]);
    texto='Matríz de Transformación Homogenea del ROBOT:';
    p(1,1:length(texto))=texto;
    for i=1:4;
    texto= sprintf('%10.2f', Tn(i,:));
    p(i+1,1:length(texto))=texto;
    end %Me llego en el cinco
    k=5;
    for j=1:size(T,3);
        %display (size(T,3));
        k=k+2;
        texto= sprintf('Matríz de Transformación del %i° ESLABÓN del robot: ',
j);
        p(k,1:length(texto))=texto;
        for i=1:4;
        k=k+1;
        texto= sprintf('%10.2f', T(i,:,j));
        p(k,1:length(texto))=texto;
        end %Me llego en el cinco
    end
    %MT = ['Matríz de Transformación Homogenea del robot', MT] ;
    set(handles.text_para_robot,'String',p);
    set(handles.boton_ver_MT,'Value', 1)
   elseif (get(hObject,'Value') == get(hObject,'Max'))
       set(handles.boton_ver_dyn,'Value', 0)
       set(handles.boton_ver_dh,'Value', 0)
       set(handles.boton_ver_MT,'Value', 0)
  else
    set (handles.text_para_robot, 'Visible', 'off');
    %  set(handles.text_para_robot,'String','Parámetros físicos del robot') ;
end
update (hObject, handles)

% VER PARÁMETROS: valores Denavit-Hatenver
function boton_ver_dh_Callback(hObject, eventdata, handles)
% hObject    handle to boton_ver_dh (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of boton_ver_dh
if (get(hObject,'Value') == get(hObject,'Max'))
    dh=handles.irb120.char; % Es la pone en pantalla los valores
    set(handles.boton_ver_dyn,'Value', 0);
    set(handles.boton_ver_MT,'Value', 0);
    set (handles.text_para_robot, 'Visible', 'on');
    set(handles.text_para_robot,'String',dh);
    set(handles.boton_ver_dh,'Value', 1)
   elseif (get(hObject,'Value') == get(hObject,'Max'))
```

```
        set(handles.boton_ver_dyn,'Value', 0)
        set(handles.boton_ver_dh,'Value', 0)
        set(handles.boton_ver_MT,'Value', 0)
  else
    set (handles.text_para_robot, 'Visible', 'off');
    %  set(handles.text_para_robot,'String','Parámetros físicos del robot') ;
end

update (hObject, handles)


% VER PARÁMETROS: valores dynámicos
function boton_ver_dyn_Callback(hObject, eventdata, handles)
% hObject    handle to boton_ver_dyn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of boton_ver_dyn
if (get(hObject,'Value') == get(hObject,'Max'))
    set(handles.boton_ver_dh,'Value', 0);
    set(handles.boton_ver_MT,'Value', 0);
    % Create and open a new example file:
  % Create and open a new example file:
    delete datos_dyn.txt;
    diary on;
    diary('datos_dyn.txt');
    handles.irb120.dyn; %Imprimo los datos
    diary off;
    type ('datos_dyn.txt');
    fich= fopen ( 'datos_dyn.txt' , 'r' ) ;
    set (handles.text_para_robot, 'Visible', 'on');
    dyn='Parámetros Dinámicos';
    while feof(fich) == 0       % test for end of file, if not then do stuff
    tline = fgetl(fich); % Lee línea por linea el fichero
    tline=sprintf('%s\n ', tline);
    dyn = [dyn, tline] ;
    end;
    set(handles.text_para_robot,'String',dyn);
    fclose ( fich ) ;
    set(handles.boton_ver_dyn,'Value', 1)
 elseif (get(hObject,'Value') == get(hObject,'Max'))
       set(handles.boton_ver_dyn,'Value', 0);
       set(handles.boton_ver_dh,'Value', 0);
       set(handles.boton_ver_MT,'Value', 0);
 else
      %set(handles.text_para_robot,'String','Parámetros físicos del robot')
      set (handles.text_para_robot, 'Visible', 'off');
end

update (hObject, handles)

% Botón ver trayectoria
function boton_ver_trayec_Callback(hObject, eventdata, handles)
% hObject    handle to boton_ver_trayec (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value') == get(hObject,'max'))
    handles.ver_trayec = true;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    handles.ver_trayec=false;
    %set(handles.boton_ver_trayec,'Value', 0)
end
update (hObject, handles)
```

```
% Botón máxima velocidad
function boton_max_vel_Callback(hObject, eventdata, handles)
% hObject    handle to boton_max_vel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of boton_max_vel
if (get(hObject,'Value') == get(hObject,'max'))
    handles.velocidad = 20;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    handles.velocidad=50;
    set(handles.boton_max_vel,'Value', 0)
end

update (hObject, handles)

 % Botón ver ejes
function boton_ver_ejes_Callback(hObject, eventdata, handles)
% hObject    handle to boton_ver_ejes (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% %%%%%%
if (get(hObject,'Value') == get(hObject,'max'))
    handles.ver_ejes = true;
elseif (get(hObject,'Value') == get(hObject,'Min'))
    handles.ver_ejes=false;
    %set(handles.boton_ver_ejes,'Value', 0)
end

update (hObject, handles)

% Botón ver transparencias
function boton_ver_transp_Callback(hObject, eventdata, handles)
% hObject    handle to boton_ver_transp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value') == get(hObject,'max'))
      handles.ver_transp =true;
elseif (get(hObject,'Value') == get(hObject,'Min'))
     handles.ver_transp = false;
end
update (hObject, handles)

function boton_conx_opc_Callback(hObject, eventdata, handles)
% hObject    handle to boton_conx_opc (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% Hint: get(hObject,'Value') returns toggle state of boton_max_vel
if (get(hObject,'Value') == get(hObject,'max'))
    display ('conecto OPC');
    %set(handles.text_on_opc, 'String', 'ON OPC');
    %set (handles.panel_opc, 'BackgroundColor', 'g');
    set (handles.text_on_opc, 'Visible', 'on');
elseif (get(hObject,'Value') == get(hObject,'Min'))
     set(handles.boton_conx_opc,'Value', 0)
     set (handles.text_on_opc, 'Visible', 'off');
end

update (hObject, handles)

% --- Executes on button press in boton_ver_puntos_grabados.
function boton_ver_puntos_grabados_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to boton_ver_puntos_grabados (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject,'Value') == get(hObject,'max'))
      handles.ver_puntos_grabados =true;
elseif (get(hObject,'Value') == get(hObject,'Min'))
     handles.ver_puntos_grabados = false;
end
update (hObject, handles)

 %{
function model_open(handles)
% Make sure the diagram is still open
 % if  isempty(find_system('Name','f14')),
    open_system('f14'); open_system('f14/Controller')
    set_param('f14/Controller/Gain','Position',[275 14 340 56])
    figure(handles.F14ControllerEditor)
    % Put  values of Kf and Ki from the GUI into the Block dialogs
    set_param('f14/Controller/Gain','Gain',...
                            get(handles.KfCurrentValue,'String'))
    set_param('f14/Controller/Proportional plus integral compensator',...
              'Numerator',...
              get(handles.KiCurrentValue,'String'))

%set_param('simu/Signal_Generator','Wave','sine');
%set_param('simu/Signal_Generator','frequency','5');
%set_param(gcs,'SimulationCommand','Start');
end
%}

%********* Código manipulación axes
    %get(rob);
    %display (ver_faces);
    % 1 opaco, 0 transparente
    %pepe=findprop(rob.patch,'Facecolor','none');
    %display (pepe);
    % = get(gca,'Children');
    %object_handles = findall(rob,'FaceAlpha');
    %display (rob);
    %rob=gca;
    %set(rob,'FaceAlpha',0); % oculta las faces
    %display (handles.rob);
    %handles.rob.= 0 ;
% --- Executes on button press in boton_conx_opc.

% MENU: ***** AYUDA
function web_autor_Callback(hObject, eventdata, handles)
% hObject    handle to web_autor (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 web ('http://mecanotronica.blogspot.com.es/','-new','-browser')

function web_corke_Callback(hObject, eventdata, handles)
% hObject    handle to web_corke (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
web (' http://www.petercorke.com','-new','-browser')

% --------------------------------------------------------------------
function web_abb_Callback(hObject, eventdata, handles)
% hObject    handle to web_abb (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
web ('http://new.abb.com/products/robotics/industrial-robots/irb-120','-new','-
browser')
 % --------------------------------------------------------------------
function web_eii_Callback(hObject, eventdata, handles)
% hObject    handle to web_eii (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
web ('http://www.eii.uva.es/','-new','-browser')

% --------------------------------------------------------------------
function web_matworks_Callback(hObject, eventdata, handles)
% hObject    handle to web_matworks (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
web ('http://www.mathworks.com/es/','-new','-browser')
```

7.2. MODELADO Y SIMULACIÓN DEL ROBOT

7.2.1. FUNCIÓN mdl_irb120

```
function irb120= mdl_irb120(tipo_robot)
% Función: mdl_irb120.m
% Modelado del Robot ABB irb120 con Toolbox rcvt de Peter Corke
% varin:tipo de robot: 11,12,13,14,15,16
% varout:robot generado

% Parámetros geométricos (DH) y dinámicos (masa, cgd, inercias, etc)
%... Fecha: 20-02-14
%... Autor: Miguel A. Mato
% ** EII-UVA-Valladolid
%  Nombre fichero:mdl_irb120
% tipo= tipo de robot

%global irb120 q0
if nargin==0; tipo_robot=11; % sin variable de entrada devuelve el robot ESFÉRICO
end

%{
%****Definición de L brazos de robot con parámetros de D-H       ---> OK!
... método antiguo
... Sigma 0 = tipo de articulación circular ...
... Offset= última columna -> desfase inicial articulación
... Limit= limites de las articulaciones  ----->    NO FUNCIONA
% Defino cada uno de los eslabones con las parámetros del robot
%L1.qlim = [-150*pi/180,150*pi/180] **
% sintaxis antigua           --->            funcionaba, OK!
L(1) = Link([0 0.29 0 -pi/2 0 0]); %
L(2) = Link([0 0 0.27 0 0 -pi/2]);
L(3) = Link([0 0 0.07 -pi/2 0 0]);
L(4) = Link([0 0.302 0 pi/2 0 0]);
L(5) = Link([0 0 0 -pi/2 0 0]);
L(6) = Link([0 0.072 0 0 0 +pi]);
%}
 rad= pi/180; % es lo mismo que degtorad(360)%* conversor a radianes radianes

% Defino los  Li eslabones con el camando REVOLUTE -es lo mismo-
% El parametro Theta (DH) es 0 entonces es de revolucion -por defecto- no lo
pongo
% 'offset' desfase inicial de la articulación
% 'I' momentos de inercia desde los cdg respecto de los ejes cooradenados
% 'r' cdg considerados desde la base del robot en m
% 'm' la masa de cada eslabón
```

```
% 'qlim' valores límites de las articulaciones
% 'G', G motor gear ratio (default 0)
% 'B', B friccion articulación, motor referenced (default 0)
% 'Jm', J motor inertia, motor referenced (default 0)
% 'Tc', T Coulomb friction, motor referenced (1 _ 1 or 2 _ 1), (default [0 0])

L(1) = Revolute('d', 0.29, 'a', 0, 'alpha', -pi/2, ...
    'offset', 0,...
    'I', [0.0142,   -1.29e-5, -2.31e-5; -1.29e-5, 0.0144, 1.93e-5;...
    -2.31e-5, 1.93e-5, 0.01050], ...
    'r', [0, 0, 0.238], ...
    'm', 8.617, ...
    'Jm', 200e-6, ...
    'G', -62.6111, ...
    'B', 1.48e-3, ...
    'Tc', [0.395 -0.435], ...
    'qlim', [-165 165]*rad ); % límites de la articulación

L(2) = Revolute('d', 0, 'a', 0.27, 'alpha', 0, ...
    'offset', -pi/2,...
    'I', [0.0603, 9.92e-6, 5.72e-5; 9.92e-6, 0.0416, -0.000505;...
    5.72e-5, -0.000505, 0.026], ...
    'r', [-0.000778, -0.00212, 0.391], ...
    'm', 3.91, ...
    'Jm', 200e-6, ...
    'G', 107.815, ...
    'B', .817e-3, ...
    'Tc', [0.126 -0.071], ...
    'qlim', [-110, 110]*rad );

L(3) = Revolute('d', 0, 'a', 0.07, 'alpha', -pi/2,  ...
    'offset', 0,...
    'I', [0.00836, -8.01e-5, 0.00143; -8.01e-5, 0.0167, -0.000182;...
    0.00143, -0.000182, 0.0127], ...
    'r', [0.0228, 0.00106, 0.618], ...
    'm', 2.94, ...
    'Jm', 200e-6, ...
    'G', -53.7063, ...
    'B', 1.38e-3, ...
    'Tc', [0.132, -0.105], ...
    'qlim', [-110, 70]*rad );

L(4) = Revolute('d', 0.302, 'a', 0, 'alpha', pi/2,  ...
    'offset', 0,...
    'I', [0.00285, -2.13e-5, -1.64e-5; -2.13e-5, 0.00401, 1.31e-5;...
    -1.64e-5, 1.31e-5, 0.00525], ...
    'r', [0.225, 0.000153, 0.63], ...
    'm', 1.33, ...
    'Jm', 33e-6, ...
    'G', 76.0364, ...
    'B', 71.2e-6, ...
    'Tc', [11.2e-3, -16.9e-3], ...
    'qlim', [-160, 160]*rad );

L(5) = Revolute('d', 0, 'a', 0, 'alpha', -pi/2,  ...
    'offset', 0,...
    'I', [0.000405, 1.62e-6, 0; 1.62e-6, 0.000893, 0; 0, 0, 0.000815], ...
    'r', [0.301, 0, 0.63], ...
    'm', 0.547, ...
    'Jm', 33e-6, ...
    'G', 71.923, ...
    'B', 82.6e-6, ...
    'Tc', [9.26e-3, -14.5e-3], ...
    'qlim', [-120, 120]*rad );
```

```
% Si queremos que la muñeca del robot sea esférica
if tipo_robot==13; Ld6=0 ; %muñeca ESFÉRICA,
else Ld6=0.072; % Muñeca NO esférica (casos 11, 12, 14 y 15)
end
% Si pongo el parametro d=0.072 el robot no tiene la muñeca esférica

L(6) = Revolute('d', Ld6 , 'a', 0, 'alpha', 0,  ...
    'offset', pi,...
    'I', [0, 0, 0; 0, 0, 0; 0,0, 0], ...
    'r', [0, 0, 0.7], ...
    'm', 0.019, ...
    'Jm', 33e-6, ...
    'G', 76.686, ...
    'B', 36.7e-6, ...
    'Tc', [3.96e-3, -10.5e-3], ...
    'qlim', [-400, 400]*rad );

%{
% Para dibujar el puntero
    if tipo_robot>=15
L(7) = Revolute('d', 0.1 , 'a', 0, 'alpha', 0,  ...
    'offset', pi,...
    'I', [0, 0, 0; 0, 0, 0; 0,0, 0], ...
    'r', [0, 0, 0.7], ...
    'm', 0.3, ...
    'Jm', 33e-6, ...
    'G', 76.686, ...
    'B', 36.7e-6, ...
    'Tc', [3.96e-3, -10.5e-3], ...
    'qlim', [0, 0]*rad );
end
%}
% Ensamblamos los brazos con Seriallink y le damos el nombre de irb120
irb120 = SerialLink(L,'name', 'irb120', 'manufacturer', 'ABB', 'comment',
'MAMato');


% Valores máximos de alcance de las articulaciones (las tengo ya metidas)
% QLG= [-165, 165;-110, 110; -110, 70; -160, 160; -120, 120; -400, 400];
% QLR=rad*QLG; %Paso los elementos de la matriz a radianes --> QLR

q0=zeros(1,6); % hago cero todos los angulos (1 fila, 6 columnas: q0=[0 0 0 0 0
0];
% *** Añadir elementos al robot (tool y base) -------->OK!

%Opciones del axes
axis ([-0.5 0.5 -0.5 0.5 0 1]);
%campos([2 2 1]); % posición de la camara
%daspect([1 1 1]);
%camproj orthographic; %perspectiva
%cla;
%opts = irb120.plot({'workspace', W', 'opt2', ... });
% Selección del tipo de robot
switch tipo_robot; %opcion el numero de case
case 11;% Muñeca NO ESFERICA, (0.072)
    titulo= 'Robot NO esferico, con muñeca de 0,072';
    irb120.plot(q0); % robot palillo
case 12; % La muñeca tiene los 0,72, le añado el puntero y la base
    irb120.tool=transl(0,0,0.1); % Añadimos el puntero 0,1 ********
    irb120.base=transl(0,0,0.25);
    axis ([-0.5 0.5 -0.5 0.5 0 1.2]);
    titulo= 'Robot NO esférico, con muñeca (0,072), base (0,25) y puntero (0,1)';
    irb120.plot(q0); % robot palillo
```

```
case 13 % 13 Muñeca ESFERICA le añado el puntero (0.072)
    irb120.tool=transl(0,0,0.072); %%%%%% oJO!
    titulo= 'Robot ESFERICO, con puntero (0,072)';
    irb120.name =( 'irb120sF'); % renombro el robot con irb120sd al ser esférico
    irb120.plot(q0); % robot palillo
case 14; % robot irb120 3D cad
    titulo= 'Robot 3D CAD';
case 15; % robot irb120 3D cad
    titulo= 'Robot 3D CAD con puntero de 0,1 m';
    irb120.tool=transl(0,0,0.1);
case 16; % robot irb120 3D cad
    titulo= 'Robot 3D CAD, con puntero de 0,1 y base de 0,25 m';
    irb120.base=transl(0,0,0.25);
    axis ([-0.5 0.5 -0.5 0.5 0 1.2]);
    irb120.tool=transl(0,0,0.1);
end

title(titulo,'FontSize',10);

return
```

7.2.2. FUNCIÓN plot_irb120

```
function plot_irb120 (q,handles,items)
% Plot del robot en 3D CAD
% Simulación y control de robot ABB irb120
% Miguel A. Mato. UVA-TFG Ingeniero Mecánico- 2014
% q: puede ser un punto q o una trayectoria Q
% handles: contiene la estructura de la aplicación
% items: variables para la comunicación OPC

persistent homogPoints robot faces colors f time rob

if nargin<3,
    items=[];
end

if isnan(q) ;
     errordlg('No se puede acceder a dichas posiciones (NAN) -->q0','Error en
plot_3D1');
     % Mantengo el robot en posición q0
return; % Salgo de la función
else %ejecuta todo el resto de la función

robot = handles.irb120;
%q0=zeros(1,6);
%{
if nargin <3
    ver=0;
end %sin ejes por defecto
if nargin<4
    transp=0;
end % sin transparencia
if nargin<5
    trayec=0;
end

if isempty(q);
q=q0;
end;
    rad=pi/180;
 %}
    rad=pi/180;
```

```
    files =
{'link1','link2','link3','link4','link5','link6','link_puntero','link_soporte','l
ink_base'}; %,
     f = length(files); % número de componentes del robot
% Coordenadas de los patch
   if isempty(homogPoints)% || ver==1; % Si ya está cargado,no los vuelve a coger
       homogPoints = cell(1,f);
       faces = cell(1,f);
           for i = 1: f ; % para cada elemento del robot (eslabones)
           %[ithPoints, ithFaces] = stlread([files{i},'.stl']);
           [ithFaces, ithPoints,ithColors] = rndread([files{i},'.stl']);
           homogPoints{i} = [ithPoints'; ones(1, size(ithPoints,1))];
           faces{i} = ithFaces;
           colors(i,:)= ithColors;
           end
    end
    % fin de cargar las coordenas del patch

%    Control de transparencia del robot CAD 3D
    if handles.ver_transp %(si transp=true,  hay trasparencia )
        FAlpha=0;
        EAlpha=1;
        %colors='none';
    else  % Transp =0, sin no hay transparencia (robot normal)
        %display('sin transparencia');
        FAlpha=1; % faces 0
        EAlpha=0;  % Vertices 1
        %colors; % Los vertices tienen color
    end
    %}
%***** Aquí empieza el dibjo del robot

    qs=size(q,1);
    for row = 1:qs  % row=numero de puntos (ejecutar trayectorias)
        % Control del tiempo, cuando entre más de una q
        pause (0.00001)
        if qs==1;
            time =0;
            cla; % si entra sólo un q, borro el axis
        elseif ~handles.ver_trayec;
            cla;
        else delete(rob);
        end
        ti=tic;
        light('Position', [1 0.5 1]);
        material shiny;
        %hold(bas,soporte);
        % cojo el tiempo al empezar
        % lo puedo interpretar como una velocidad de refresco
        %if exist dib_rob; cla (dib_rob); end;
        %end %type = exist(rob)

        %T es de cada uno de los eslabones, Tn es la matriz de
        [Tn, T] = robot.fkine(q(row,:));

        % *****Dibujo del ROBOT **************
        % SOPORTE DE LA BASE (gris oscuro), si lo hay
        if robot.base(1,4)~=0 || robot.base(2,4)~=0 || robot.base(3,4)~=0, %zeros
(3,1); % si tiene  SOPORTE
             % si está vacio en persistente lo cargo
              pts = transl (-0.09, -0.09, 0)*homogPoints{8};% El dibujo de la
malla está desplazado
              pts = pts(1:3,:)'; %puntos de los vértices
```

```
                patch('Faces', faces{8}, 'Vertices', pts, 'FaceColor',
colors(8,:),'FaceAlpha', FAlpha , ...
                 'Edgecolor',  colors(8,:),'EdgeAlpha', EAlpha); %dibujo el
robot,con los vertices ocultos
        end %Fin dibujo de la base del robot

       % Dibujo BASE con conexiones (azul claro)
                pts = homogPoints{9}; % Cojo los puntos del último link -base-
                pts=(robot.base)*pts;
                pts = pts(1:3,:)'; %puntos de los vértices de la base
                patch('Faces', faces{f}, 'Vertices', pts, 'FaceColor', [0 1
1],'FaceAlpha', FAlpha, ...
                     'Edgecolor',  [0 1 1] ,'EdgeAlpha', EAlpha);
            % Dibujo los ejes de la base (no debería)
            if handles.ver_ejes; % si el parámetro ver está activado, veo los
ejes de la base
            trplot([1 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1], 'frame', '0','color',
'k','length',0.22,...
                'width', 1.4 , 'rgb', 'thick', 1.3); % Ejes del robot
            end %fin dibujo de la base con conexiones


                for link = 1:6; % Dibujo del resto de eslabones (los 6 eslabones)
                    pts = T(:,:,link) * homogPoints{link};
                    pts = pts(1:3,:)';
                    rob(link)=patch('Faces', faces{link}, 'Vertices', pts,
'FaceColor',  colors(link,:),'FaceAlpha', FAlpha , ...
                        'Edgecolor',  colors(link,:),'EdgeAlpha', EAlpha);
%dibujo el robot,con los vertices ocultos
                 if handles.ver_ejes; % si el parámetro VER está activado, veo
los ejes de cada articulación
                 trplot(T(:,:,link), 'frame', num2str(link),'color',
'k','length',0.22,'width', 1.4 ,...
                     'rgb', 'thick', 1.3); % Ejes del robot
                 end %Fin dibujo del robot
                end
          % ** Dibujo del puntero
          if robot.tool(1,4)~=0 || robot.tool(2,4)~=0 || robot.tool(3,4)~=0,  %
zeros (3,1); % si tiene puntero
                %display (irb120.tool(1:3,4));
                % t2 = transl(p2); % t2 "
                % t1= t1 * troty (pi); % giro el punto sobre eje y (funciona
mejor)
                pts = T(:,:,6) *transl (-0.02, -0.02, 0);
                pts= pts* homogPoints{7};
                pts = pts(1:3,:)';
                rob (7)=patch('Faces', faces{7}, 'Vertices', pts, 'FaceColor',
colors(7,:),'FaceAlpha', FAlpha , ...
                     'Edgecolor',  colors(7,:),'EdgeAlpha', EAlpha);
          end %Fin dibujo del robot
        % Dibujo del XYZ del TCP del robot
         trplot(Tn, 'color', 'k','length',0.13,'width', 1.2 , 'rgb', 'thick',
1.3); %XYZ del TCP


        %**** Valor de las coordenadas del TCP
        coord= num2str(Tn(1:3,4)','% 7.2f' ); % convierto los números en texto
X,Y,Z
        tf=toc(ti); %tiempo final
        time=time+tf;
        espacios='      ||      '; % espacios en blanco
        title({['Q [°] = ',num2str(q(row,:)/rad, '% 6.1f')];['XYZ [m]=
',coord,espacios ,'Time [s]= ', num2str((time),'%4.2f')]});
```

```
         %** Ver gráficos de DEMO guardadas
        if ~isempty (handles.demos),
          %En este caso es una trayectoria de puntos
          line (handles.demos(:,1), handles.demos(:,2),
handles.demos(:,3),'LineWidth', 1, 'Marker','+');%dibujo linea 3D
        %set(gca,'CurrentAxes',handles.demos)
        end

         % Puntos guardados/ ralentiza el movimiento del robot
         if handles.ver_puntos_grabados
         if ~isempty (handles.puntos_grabados);
              j=size (handles.puntos_grabados,3);
              for k=1:j
              trplot(handles.puntos_grabados(:,:,k), 'frame', ['P',num2str(k)],
'length',0.15,'width', 1, 'rgb', 'thick', 1);
              end
         end
         end

         % Retraso para sincronización con elrobot real TIME_OPC
         pause(handles.time_opc);

          %{
         %get(tcp);
         if ~isempty (handles.punto);
             n=size(TQ,3);
             for i=1:n % Dibujo los puntos
                 trplot(TQ(:,:,i), 'frame', ['P',num2str(i)],
'length',0.17,'width', 1 ,...
                     'rgb', 'thick', 1); %punto que tiene que
             end
         end
         %}
         %str(3) = {'With the values:'};
         %set(gco,'CurrentAxes',h)
         %text(.025,.6,str,'FontSize',12)
         %coord=strcat('XYZ= ', coord);
         %uicontrol('Style', 'edit', 'String', coord ,'Position', [600 500 100
30])
         %rob=gca; % Recojo el robot en la variable rob
         %get (rob); % Miro lo que lleva

    end  %Fin dibujo del robot y multiples posiciones
    %cla (rob)
    %rob = get(gca,'Children');

 end % si no tiene NAN
 end

 % Extracted from cad2matdemo
function [fout, vout, cout] = rndread(filename)
% Reads CAD STL ASCII files, which most CAD programs can export.
% Used to create Matlab patches of CAD 3D data.
% Returns a vertex list and face list, for Matlab patch command.
%

fid=fopen(filename, 'r'); %Open the file, assumes STL ASCII format.
if fid == -1
    error('File could not be opened, check name or path.')
end

% Render files take the form:
%solid BLOCK
```

```
%  color 1.000 1.000 1.000
%  facet
%      normal 0.000000e+00 0.000000e+00 -1.000000e+00
%      normal 0.000000e+00 0.000000e+00 -1.000000e+00
%      normal 0.000000e+00 0.000000e+00 -1.000000e+00
%    outer loop
%      vertex 5.000000e-01 -5.000000e-01 -5.000000e-01
%      vertex -5.000000e-01 -5.000000e-01 -5.000000e-01
%      vertex -5.000000e-01 5.000000e-01 -5.000000e-01
%    endloop
% endfacet
%
% The first line is object name, then comes multiple facet and vertex lines.
% A color specifier is next, followed by those faces of that color, until
% next color line.
%
%CAD_object_name = sscanf(fgetl(fid), '%*s %s');  %CAD object name, if needed.
%                                                 %Some STLs have it, some don't.
       %Vertex number counter.
%report_num=0; %Report the status as we go.
%vcolor = 0; % color para los eslabones del robot

vnum=0;
while feof(fid) == 0                % test for end of file, if not then do stuff
    tline = fgetl(fid);             % reads a line of data from file.
    fword = sscanf(tline, '%s ');   % make the line a character string
% Check for color
    if strncmpi(fword, 'c',1) == 1;% Si en una línea hay "C"olor, como 1°caracter
       vcolor = sscanf(tline, '%*s %f %f %f');% & if a C, get the RGB color data
of the face
    end                          % Keep this color, until the next color is used.
    % Checking if a "V"ertex line, as "V" is 1st char.
    if strncmpi(fword, 'v',1) == 1;
       vnum = vnum + 1;                % If a V we count the # of V's
       %report_num = report_num + 1;    % Report a counter, so long files show
status
       %if report_num > 249; % Numero máximo de vertices recogidos
           %fprintf('Reading vertix num: %d.',vnum);
       %    report_num = 0;
       %end
       v(:,vnum) = sscanf(tline, '%*s %f %f %f'); % & if a V, get the XYZ data of
it.
       % c(:,vnum) = VColor;              % A color for each vertex, which will
color the faces.
    end                                 % we "*s" skip the name "color" and get
the data.
end
%   Build face list; The vertices are in order, so just number them.

fnum = vnum/3;      %Number of faces, vnum is number of vertices.  STL is
triangles.
flist = 1:vnum;     %Face list of vertices, all in order.
F = reshape(flist, 3,fnum); %Make a "3 by fnum" matrix of face list data.
%
%   Return the faces and vertexs.
%
fout = F';  %Orients the array for direct use in patch.
vout = v';  % "
cout = vcolor'; % colores

fclose(fid);
end
```

7.2.3. FUNCIÓN espacio_trabajo_lin

```
function T=espacio_trabajo_lin(q,irb120)
% Función
% Dibuja nebulosa de puntos de las posiciones límites del robot
%global irb120
rad= pi/180;

% QLG= [-165, 165;-110, 110; -110, 70; -160, 160; -120, 120; -400, 400];

%pasoq1 = 330*rad/10;
pasoq2 = 220*rad/10;
pasoq3 = 180*rad/10;
pasoq4= 320*rad/6;
pasoq5= 240*rad/4;
x=[]; y=[]; z=[];
%Volumen mal calculado pero con mejor surface
display (q(1));
for q1=q(1)%:pasoq1:165*rad,
    for q2=-110*rad:pasoq2:110*rad,
        for q3=-110*rad:pasoq3:70*rad;
            for q4=-160*rad:pasoq4:160*rad;
                for q5=-120*rad:pasoq5:120*rad;
                T = irb120.fkine([q1 q2 q3 q4 q5 0]);
                x =[x, T(1,4)];
                y =[y, T(2,4)];% abs()
                z= [z, T(3,4)];
                end
            end
         end
    end
end

%title('Espacio de trabajo del robot','FontSize',10);
hold on;
plot3(x,y,z,'g');

return;
```

7.2.4. FUNCIÓN graf_puntos

```
function graf_puntos(T1,T2,t)
%% Gráfica de la desviación entre los puntos alcanzados entre trayectorias
% Imput:
% ** T1: Matriz homogenea (3 dim) de los valores dinámica directa --> ctraj
% ** T2: Matris homogenea de los valores dinámica inversa
% ** t: Array de los valores calculados (intervalo tiempo)

% Si no me pasan t
if nargin < 3
    n = size(T1,3); % numero de puntos calculados en TY1
    if n==1; % si TY1 solo contiene un punto
         t1x=T1(1,4); t1y=T1(2,4);t1z=T1(3,4); % Puntos para la gráfica
         n= size(T2,3); % Tomo el numero de puntos TY2
         t2x=T2(1,4); t2y=T2(2,4);t2z=T2(3,4); % Puntos para la gráfica
         t=(0:n)/10;
    end
else n=length(t);
end

figure(); % Gráfica 2D (x,y,z) que alcanza el robot en línea recta
```

```
hold on;
%grid on; % desplegamos la red -- emborrona el gráfico
xlabel('Eje t[s]','fontname', 'Comic Sans Ms', 'fontsize',10);
ylabel('Eje posiciones alzanzadas(x,y,z)[m]','fontname', 'Comic Sans Ms',
'fontsize',10);
title('Comparativa posiciones alcanzadas');

for i=1:n % nº de elementos de t, puntos que ha descrito el robot
    if  size(T1,3)~=1
    t1x= T1(1,4,i);
    t1y= T1 (2,4,i);
    t1z= T1(3,4,i);
    end
    if  size(T2,3)~=1
    t2x= T2(1,4,i);
    t2y= T2 (2,4,i);
    t2z= T2(3,4,i);
    end
    plot (t(i),t1x,'bo'); % trayectoria calculada por ctraj
    plot (t(i),t2x,'bx'); % trayectoria
    plot (t(i),t1y, 'ro'); % trayectoria calculada por ctraj
    plot (t(i),t2y, 'rx'); % trayectoria
    plot (t(i),t1z, 'go'); % trayectoria calculada por ctraj
    plot (t(i),t2z, 'gx'); % trayectoria
    %plot (T_recta_inv(1,4,i), T_recta_inv(2,4,i), T_recta_inv(3,4,i), 'ro');
    % puntos a los que llega realmente el robot
end

% Etiquetamos las series
label1= inputname(1);
label2= inputname(2);
lt1x=strcat('X ', label1);
lt2x=strcat('X ',label2);
lt1y=strcat('Y ', label1);
lt2y=strcat('Y ', label2);
lt1z=strcat('Z ', label1);
lt2z=strcat('Z ', label2);

legend(lt1x,lt2x,lt1y,lt2y,lt1z,lt2z,'location', 'NorthEastOutside');
end
```

7.2.5. FUNCIÓN save_trayec

```
function fid=save_trayec(datos)
% Función para grabar datos en un fichero,
% datos: tipo de datos a grabar
% t: identifica el tipo. 1= v. arti. 0 Mmatriz homogénea)
%nom_fich = impdiag('Se salvará la trayectoria en el fichero: datos.txt: ',
'Información grabación datos');
%if isempty(nom_fich); % nombre por defecto datos.txt
if isempty(datos);
    errordlg('No existen puntos grabados!','Error');% Si no hay datos en
handles.Q
    fid=-1;
else
    msgbox('Se salvará la trayectoria en el fichero: datos.txt ', 'Información
grabación datos');
    nom_fich = 'datos.txt';
    %nom_fich=sprintf('%s.%s',nom_fich,'txt');
    %end
    fid=fopen ( nom_fich, 'w' ); % Guardar en un fichero nombre.txt
    if size(datos,2)==6
        fprintf ( fid ,'%.4g %.4g %.4g %.4g %.4g %.4g\n', datos') ;%Formato
v.art.
```

```
    elseif size(datos,2)==4
        fprintf ( fid ,'%.4g %.4g %.4g %.4g\n', datos) ;%Formato disp ('no está
hecha');
        %fprintf (fid, datos) ;%OJO!! Formato coor.H.
    else
        errordlg('Error en la grabación de datos, repita!','Error');
        fid=-1;
    end
fclose (fid);
end


 %{
fid = -1;
msg = '';
while fid < 0
   disp(msg);
   filename = input('Abre el fichero: ', 's');
   [fid,msg] = fopen(filename);
end

% Create and open a new example file:
fid = fopen('exampleFile.txt','w+');
% Write something to the file and the console simultaneously:
multidfprintf([1 FID],'% s % d % d % d% Close the file:
fclose(FID);
%}
```

7.2.6. FUNCIÓN ikine_nuevo

```
function punto=ikine_nuevo (irb120,t)
% Función ikine_nueva optimizada por Alberto Herreros (25-03-14)
% Transformada inversa por tramos

%%
%{
if nargin<2;
    pto=[.3, -.2, 0.4];
else pto=punto;
end
%}

display (t);
q=zeros(1,6);
deg= pi/180; rad= 180/pi;

%display (q);

%% Optimización multi-start

F= @(q) FncIkine(q, irb120, t); % Paso a la nueva función q

N= 5; % N intentos para mejorar
fopt= Inf;
for i= 1:N,
   x0= rand(1,6);
   [x, f]= fminsearch(F, x0);
   if f<fopt,
       fopt= f; xopt= x;
   end
end
```

```
% Convergencia final
for i= 1:2,
   [xopt]= fminsearch(F, xopt);
end

% Resultado
[fopt, qopt]= F(xopt);
%qopt*rad;
%q*rad;
%irb120.plot(q);
%plot_3D1(q,irb120)
%hold on;
%plot3 (pto(1), pto(2),pto(3), 'ro');
%trplot(T, 'frame', 'P', 'color', 'g', 'thick', 1.1); %punto que tiene que
% alcanzar
%plot_3D1(q,irb120)
%irb120.plot(q);
%plot_3D1(qopt,irb120)
%irb120.plot(qopt); %Solución optimizada
punto=qopt;
%trplot(Topt, 'frame', 'Top', 'color', 'c', 'thick', 1); %punto que tiene que
%display (Topt);
%display(['Partiendo de la posición articular (q0): ', num2str(q*rad)]);
%display (['Solución encontrada (qopt):  ', num2str(qopt*rad)]);
%punto=
%display(['Función de coste: P2: ', num2str(fopt*rad)]);
%disp('<<<< Pulsa una tecla para volver menu principal....');
%pause;
%disp('>>>> 1. Gráfica de comparativa de precisión ... ');
%graf_puntos(T,T_inv);
return

function [f, q]= FncIkine(q, irb120, Tref)

rad= pi/180;
lim= [-165, -110, -110, -160, -120, -400;
       165,  110,   70,  160,  120,  400]*rad;
pos_ref= transl(Tref); orient_ref= Quaternion(Tref);
N= size(q,1);
f= zeros(N,1);

for i= 1:N,
    q(i,:)= q(i,:)- floor(q(i,:));
    q(i,:)= lim(1, :)+ q(i,:).*(lim(2,:)- lim(1,:));
    That= irb120.fkine(q(i,:));
    pos_hat= transl(That); orient_hat= Quaternion(That);

    pos_err= pos_ref- pos_hat;
    orient_err= orient_ref- orient_hat;
    f(i)= norm(pos_err)+ norm(orient_err);
end
return
```

7.2.7. FUNCIÓN ikine_irb120 (ARTE)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Q = ikine_IRB120(robot, T)
%   Solves the inverse kinematic problem for the ABB IRB 140 robot
%   where:
%   robot stores the robot parameters.
%   T is an homogeneous transform that specifies the position/orientation
%   of the end effector.
```

```
%
%   A call to Q=INVERSEKINEMATIC_IRB6620 returns 8 possible solutions, thus,
%   Q is a 6x8 matrix where each column stores 6 feasible joint values.
%
%   Example code:
%
%   abb=load_robot('abb', 'IRB120');
%   q = [0 0 0 0 0 0];
%   T = directkinematic(abb, q);
%   %Call the inversekinematic for this robot
%   qinv = inversekinematic(abb, T);
%   check that all of them are feasible solutions!
%   and every Ti equals T
%   for i=1:8,
%        Ti = directkinematic(abb, qinv(:,i))
%   end
%   See also DIRECTKINEMATIC.
%
%   Author: Arturo Gil Aparicio
%           Universitas Miguel Hernï¿½ndez, SPAIN.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Copyright (C) 2012, by Arturo Gil Aparicio
%
% This file is part of ARTE (A Robotics Toolbox for Education).
%
% ARTE is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% ARTE is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU Lesser General Public License for more details.
%
% You should have received a copy of the GNU Leser General Public License
% along with ARTE.  If not, see <http://www.gnu.org/licenses/>.
function q= ikine_irb120(T, tipo_robot)

%Control de herramienta
if nargin<2, tl=0; bs=0;
else
    switch tipo_robot; %opcion el numero de case
    case {11,14},
     bs=0;
     tl=0;
     case {12,16},
     bs=0.25;
     tl=0.1;
     case {13,15},
     bs=0;
     tl=0.1;
    end;
end

robot= parameters;
%initialize q,
%eight possible solutions are generally feasible
q= zeros(6,8);

% %Evaluate the parameters
% theta = eval(robot.DH.theta);
d= eval(robot.DH.d);
L6=d(6);
```

```
%display (d);

%T= [ nx ox ax Px;
%     ny oy ay Py;
%     nz oz az Pz];
Px=T(1,4);
Py=T(2,4);
Pz=T(3,4);

%Compute the position of the wrist, being W the Z component of the end effector's
system
W = T(1:3,3);

% Pm: wrist position
Pm = [Px Py (Pz-bs)]' - (L6+tl)*W;

%first joint, two possible solutions admited:
% if q(1) is a solution, then q(1) + pi is also a solution
q1=atan2(Pm(2), Pm(1));


%solve for q2
q2_1=solve_for_theta2(robot, [q1 0 0 0 0 0 0], Pm);
%the other possible solution is q1 + pi
q2_2=solve_for_theta2(robot, [q1+pi 0 0 0 0 0 0], Pm);

%solve for q3
q3_1=solve_for_theta3(robot, [q1 0 0 0 0 0 0], Pm);
%solver for q3 for both cases
q3_2=solve_for_theta3(robot, [q1+pi 0 0 0 0 0 0], Pm);


%Arrange solutions, there are 4 possible solutions so far, being
% each column repeated twice. For each triplet (theta1, theta2, theta3),
% there exist two possible solutions for the last three joints, generally
% called wrist up and wrist down solutions
% NOTE: so far there exist 4 possible solutions
% q = [q1    q1     q1+pi  q1+pi;
%      q2_1(1) q2_1(2)   q2_2(1)  q2_2(2);
%      q3_1(1) q3_1(2)  q3_2(1)  q3_2(2);
%      0          0           0       0;
%      0          0           0       0;
%      0          0           0       0];

%the next matrix doubles each column. For each two columns, two different
%configurations for theta4, theta5 and theta6 will be computed. These
%configurations are generally referred as wrist up and wrist down solution
q = [q1         q1         q1         q1        q1+pi    q1+pi    q1+pi    q1+pi;
     q2_1(1)    q2_1(1)    q2_1(2)    q2_1(2)   q2_2(1)  q2_2(1)  q2_2(2)  q2_2(2);
     q3_1(1)    q3_1(1)    q3_1(2)    q3_1(2)   q3_2(1)  q3_2(1)  q3_2(2)  q3_2(2);
     0          0          0          0          0       0        0        0;
     0          0          0          0          0       0        0        0;
     0          0          0          0          0       0        0        0];

 %normalize q to [-pi, pi]
 q = normalize(q);

% solve for the last three joints
% for any of the possible combinations (theta1, theta2, theta3)
for i=1:2:size(q,2),
    qtemp = solve_spherical_wrist(robot, q(:,i), T, 1,'geometric'); %wrist up
    q(:,i)=qtemp;

    qtemp = solve_spherical_wrist(robot, q(:,i), T, -1, 'geometric'); %wrist up
    q(:,i+1)=qtemp;
```

```
end
%display(q);
   q= q';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% solve for second joint theta2, two different
% solutions are returned, corresponding
% to elbow up and down solution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function q2 = solve_for_theta2(robot, q, Pm)

%Evaluate the parameters
theta = eval(robot.DH.theta);
d = eval(robot.DH.d);
a = eval(robot.DH.a);
alpha = eval(robot.DH.alpha);

%See geometry
L2=a(2);
L3=d(4);
A2=a(3);

%See geometry of the robot
%compute L4
L4 = sqrt(A2^2 + L3^2);

%The inverse kinematic problem can be solved as in the IRB 140 (for example)

%given q1 is known, compute first DH transformation
T01=dh(robot, q, 1);

%Express Pm in the reference system 1, for convenience
p1 = inv(T01)*[Pm; 1];

r = sqrt(p1(1)^2 + p1(2)^2);

beta = atan2(-p1(2), p1(1));
gamma = real(acos((L2^2+r^2-L4^2)/(2*r*L2)));

%return two possible solutions
%elbow up and elbow down
%the order here is important and is coordinated with the function
%solve_for_theta3
q2(1) = pi/2 - beta - gamma; %elbow up
q2(2) = pi/2 - beta + gamma; %elbow down


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% solve for third joint theta3, two different
% solutions are returned, corresponding
% to elbow up and down solution
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function q3 = solve_for_theta3(robot, q, Pm)

%Evaluate the parameters
theta = eval(robot.DH.theta);
d = eval(robot.DH.d);
a = eval(robot.DH.a);
alpha = eval(robot.DH.alpha);

%See geometry
L2=a(2);
L3=d(4);
A2=a(3);
```

```
%See geometry of the robot
%compute L4
L4 = sqrt(A2^2 + L3^2);

%the angle phi is fixed
phi=acos((A2^2+L4^2-L3^2)/(2*A2*L4));

%given q1 is known, compute first DH transformation
T01=dh(robot, q, 1);

%Express Pm in the reference system 1, for convenience
p1 = inv(T01)*[Pm; 1];

r = sqrt(p1(1)^2 + p1(2)^2);

eta = real(acos((L2^2 + L4^2 - r^2)/(2*L2*L4)));

%return two possible solutions
%elbow up and elbow down solutions
%the order here is important
q3(1) = pi - phi- eta;
q3(2) = pi - phi + eta;

%Subfunción normalize

function q = normalize(q)

for i=1:size(q,2)
   q(:,i)=atan2(sin(q(:,i)),cos(q(:,i)));
end

%Subfunción parameters
function robot = parameters()

robot.name= 'ABB_IRB120';

robot.DH.theta= '[q(1) q(2)-pi/2 q(3) q(4) q(5) q(6)+pi]';
robot.DH.d='[0.290 0 0 0.302 0 0.072]';
robot.DH.a='[0 0.270 0.070 0 0 0]';
robot.DH.alpha= '[-pi/2 0 -pi/2 pi/2 -pi/2 0]';
%robot.DH.tl='[0.1]';
%robot.DH.bs='[0.25]';

robot.J=[];

robot.inversekinematic_fn = 'inversekinematic_irb120(robot, T)';

%number of degrees of freedom
robot.DOF = 6;

%rotational: 0, translational: 1
robot.kind=['R' 'R' 'R' 'R' 'R' 'R'];

%minimum and maximum rotation angle in rad
robot.maxangle =[deg2rad(-165) deg2rad(165); %Axis 1, minimum, maximum
                 deg2rad(-110) deg2rad(110); %Axis 2, minimum, maximum
                 deg2rad(-110) deg2rad(70); %Axis 3
                 deg2rad(-160) deg2rad(160); %Axis 4:
                 deg2rad(-120) deg2rad(120); %Axis 5
                 deg2rad(-400) deg2rad(400)]; %Axis 6:

%maximum absolute speed of each joint rad/s or m/s
%robot.velmax = [deg2rad(250); %Axis 1, rad/s
%                deg2rad(90); %Axis 2, rad/s
%                deg2rad(90); %Axis 3, rad/s
```

```
%                deg2rad(150); %Axis 4, rad/s
%                deg2rad(120); %Axis 5, rad/s
%                deg2rad(190)];%Axis 6, rad/s

%robot.accelmax=robot.velmax/0.1; % 0.1 is here an acceleration time
            % end effectors maximum velocity
%robot.linear_velmax = 0.0; %m/s, unavailable from datasheet

%base reference system
robot.T0 = eye(4);

function q = inversekinematic(T)

robot= parameters;
%if robot.debug
%    fprintf('\nComputing inverse kinematics for the %s robot\n', robot.name);
%    fprintf('\nCall to: inversekinematic/%s\n', robot.inversekinematic_fn);
%end
%Call specific inversekinematic function
q = eval(robot.inversekinematic_fn);

function T = directkinematic(q)

robot= parameters;

%In the case of a parallel robot, switch to its particular direct kinematic
%function
if isfield(robot, 'parallel')
    %Call specific direct kinematic function for parallel robots
    T = eval(robot.directkinematic_fn);
    return;
end

%compute direct kinematics for serial robotics
theta = eval(robot.DH.theta);
d = eval(robot.DH.d);
a = eval(robot.DH.a);
alfa = eval(robot.DH.alpha);

n=length(theta); %number of DOFs

%if robot.debug
%    fprintf('\nComputing direct kinematics for the %s robot with %d
DOFs\n',robot.name, n);
%end
%load the position/orientation of the robot's base
T = robot.T0;

for i=1:n,
    T=T*dh(theta(i), d(i), a(i), alfa(i));
end
```

7.2.7. 1) *FUNCIÓN dh*

```
%   DENAVIT Compute an homogeneous transform matrix DH in terms of
%   Denavit-Hartenberg's parameters of the robot
%
%   A = DH(TETA, D, A, alpha) returns a 4 x 4 homogeneous
%   transformation matrix  as a function of the the Denavit-Hartenberg's
%   parameters D, alpha, A and THETA for link i.
%
%   A = DH(robot, q, i) is an abbreviated call to return a 4x4
%   homogeneous transformation matrix as a function of the robot parameters
%   robot.DH.theta, d, a, alpha the joint values q and the transformation i.
```

```
%   For i = 1, the function returns the transformation matrix T01, for i=2,
%   T02..., etc.
%   See also DIRECTKINEMATIC.
%   Author: Arturo Gil. Universidad Miguel Hernández de Elche.
%   email: arturo.gil@umh.es date:   01/01/2012

% Copyright (C) 2012, by Arturo Gil Aparicio
%
% This file is part of ARTE (A Robotics Toolbox for Education).
%
% ARTE is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% ARTE is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU Lesser General Public License for more details.
%
% You should have received a copy of the GNU Leser General Public License
% along with ARTE.  If not, see <http://www.gnu.org/licenses/>.
function A=dh(theta, d, a, alpha)
switch nargin
    case 3 %abbreviated call to denavit
        %arrange arguments
        robot = theta;
        q = d;
        i = a;

        theta = eval(robot.DH.theta);
        d = eval(robot.DH.d);
        a = eval(robot.DH.a);
        alpha = eval(robot.DH.alpha);

        theta = theta(i);
        d = d(i);
        a = a(i);
        alpha = alpha(i);
        A=[cos(theta)  -cos(alpha)*sin(theta)   sin(alpha)*sin(theta)
a*cos(theta);
           sin(theta)   cos(alpha)*cos(theta)  -sin(alpha)*cos(theta)
a*sin(theta);
           0              sin(alpha)             cos(alpha)             d;
           0                      0                      0             1];
    case 4 %full 4 argumen call to denavit
        A=[cos(theta)  -cos(alpha)*sin(theta)   sin(alpha)*sin(theta)
a*cos(theta);
           sin(theta)   cos(alpha)*cos(theta)  -sin(alpha)*cos(theta)
a*sin(theta);
           0              sin(alpha)             cos(alpha)             d;
           0         0                    0              1];
    otherwise
        disp('ERROR:denavit: uncorrect number of arguments')
end
```

7.2.7. 2) *FUNCIÓN solve_spherical_wrist*

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   q = solve_spherical_wrist(robot, q, T, wrist)
%   Solves the inverse kinematic problem for a spherical wrist
%   robot: robot structure.
%   q: vector containing the values of the joints 1, 2 and 3.
```

```
%   T: orientation of the last reference system.
%   wrist: select -1 or 1 for two possible solutions (wrist up, wrist down)
%   See also DIRECTKINEMATIC.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Copyright (C) 2012, by Arturo Gil Aparicio
% This file is part of ARTE (A Robotics Toolbox for Education).
% ARTE is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
% ARTE is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU Lesser General Public License for more details.
%
% You should have received a copy of the GNU Leser General Public License
% along with ARTE.  If not, see <http://www.gnu.org/licenses/>.

function q = solve_spherical_wrist(robot, q, T, wrist, method)

switch method
     %algebraic solution
    case 'algebraic'
        T01=dh(robot, q, 1);
        T12=dh(robot, q, 2);
        T23=dh(robot, q, 3);

        Q=inv(T23)*inv(T12)*inv(T01)*T;
        %detect the degenerate case when q(5)=0, this leads to zeros
        % in Q13, Q23, Q31 and Q32 and Q33=1

        %detect if q(5)==0
        % this happens when cos(q5) in the matrix Q is close to 1
        if abs(Q(3,3)-1)>0.00001
            %normal solution
            if wrist==1 %wrist up
                q(4)=atan2(-Q(2,3),-Q(1,3));
                q(6)=atan2(-Q(3,2),Q(3,1));
                %q(5)=atan2(-Q(3,2)/sin(q(6)),Q(3,3));
            else %wrist down
                q(4)=atan2(-Q(2,3),-Q(1,3))-pi;
                q(6)=atan2(-Q(3,2),Q(3,1))+pi;
                %q(5)=atan2(-Q(3,2)/sin(q(6)),Q(3,3));
            end
            if abs(cos(q(6)+q(4)))>0.00001
                cq5=(Q(1,1)+Q(2,2))/cos(q(4)+q(6))-1;
            end
            if abs(sin(q(6)+q(4)))>0.00001
                cq5=(-Q(1,2)+Q(2,1))/sin(q(4)+q(6))-1;
            end
            if abs(sin(q(6)))>0.00001
                sq5=-Q(3,2)/sin(q(6));
            end
            if abs(cos(q(6)))>0.00001
                sq5=Q(3,1)/cos(q(6));
            end
            q(5)=atan2(sq5,cq5);

        else %degenerate solution, in this case, q4 cannot be determined,
             % so q(4)=0 is assigned
            if wrist==1 %wrist up
                q(4)=0;
                q(5)=0;
```

```
                q(6)=atan2(-Q(1,2)+Q(2,1),Q(1,1)+Q(2,2));
            else %wrist down
                q(4)=-pi;
                q(5)=0;
                q(6)=atan2(-Q(1,2)+Q(2,1),Q(1,1)+Q(2,2))+pi;
            end
        end
        %geometric solution
    case 'geometric'
        % T is the noa matrix defining the position/orientation of the end
        % effector's reference system
        vx6=T(1:3,1);
        vz5=T(1:3,3); % The vector a z6=T(1:3,3) is coincident with z5

        % Obtain the position and orientation of the system 3
        % using the already computed joints q1, q2 and q3
        T01=dh(robot, q, 1);
        T12=dh(robot, q, 2);
        T23=dh(robot, q, 3);
        T03=T01*T12*T23;

        vx3=T03(1:3,1);
        vy3=T03(1:3,2);
        vz3=T03(1:3,3);

        % find z4 normal to the plane formed by z3 and a
        vz4=cross(vz3, vz5);    % end effector's vector a: T(1:3,3)
        % in case of degenerate solution,
        % when vz3 and vz6 are parallel--> then z4=0 0 0, choose q(4)=0 as
solution
        if norm(vz4) <= 0.00001
            if wrist == 1 %wrist up
                q(4)=0;
            else
                q(4)=-pi; %wrist down
            end
        else
            %this is the normal and most frequent solution
            cosq4=wrist*dot(-vy3,vz4);
            sinq4=wrist*dot(vx3,vz4);
            q(4)=atan2(sinq4, cosq4);
        end
        %propagate the value of q(4) to compute the system 4
        T34=dh(robot, q, 4);
        T04=T03*T34;
        vx4=T04(1:3,1);
        vy4=T04(1:3,2);
        % solve for q5
        cosq5=dot(vy4,vz5);
        sinq5=dot(-vx4,vz5);
        q(5)=atan2(sinq5, cosq5);

        %propagate now q(5) to compute T05
        T45=dh(robot, q, 5);
        T05=T04*T45;
        vx5=T05(1:3,1);
        vy5=T05(1:3,2);

        % solve for q6
        cosq6=dot(vx6,vx5);
        sinq6=dot(vx6,vy5);
        q(6)=atan2(-sinq6,-cosq6); % Modificado valores negativos
       % display(q(6)); %%% mato
    otherwise
```

```
        disp('no method specified in solve_spherical_wrist');
end
```

7.3. CÓDIGO COMUNICACIÓN OPC

7.3.1. FUNCIÓN OPCitem

```
function SenItem= OPCitem(names)

   if nargin==0,
      error('Cell name of the variables')
   end

   opcreset;
   Nvar= length(names);
   SenItem= cell(1,Nvar);
   da= opcda('localhost', 'ABB.IRC5.OPC.Server.DA');
   connect(da); grp= addgroup(da);

   for i= 1:Nvar,
      item= serveritems(da, ['*', names{i}, '*']);
      if length(item)~=1,
          error('Nombre variable erroneo');
      end
      SenItem{i}= additem(grp, item);
   end
end
```

7.3.2. FUNCIÓN OPCJoint

```
function items= OPCJoint()
% OPC the de sensor variables
names= {'Joint_rax1', 'Joint_rax2', 'Joint_rax3', ...
        'Joint_rax4', 'Joint_rax5', 'Joint_rax6'};
items= OPCitem(names);
end
```

7.3.3. FUNCIÓN Joint2OPC

```
function Joint2OPC(q, items, deltaT)

if nargin<2,
    error('Joint2OPC(q, items, [deltaT])');
elseif nargin==2,
    deltaT=0;
end
MaxJ= [-165,165;-110,110;-70,70;-160,160;-120,120;-400,400]*pi/180;
MaxInc= MaxJ(:,2)-MaxJ(:,1);
Nj= size(q,1);

for j= 1:Nj,
   % Manda posición de ejes
   for i= 1:6,
      write(items{i}, q(j,i)/MaxInc(i)*200);
   end
   pause(deltaT);
end
 end
```