UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Departamento de Ingeniería de Sistemas y Automática

Desarrollo de una consola virtual para el movimiento de robots

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

Autor: **Óscar Muñoz Menzinger**
Director: Jorge Juan Feliu Batlle
Codirectora: Alejandra Armenta Valadez

Cartagena, 13 de Julio de 2018

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 : Introducción

En este primer capítulo se hará una pequeña introducción a las consolas de robots y sus características, y después se presentarán los objetivos del trabajo, mostrándose las etapas en las que este se divide.

1.1. Consolas de robots

Un robot puede tener implantada una tecnología de inteligencia artificial que haga que este se pueda mover por sí mismo, o puede ser manejado por un operador de forma manual. Cuando este es manejado por un operador, el movimiento del robot es controlado con una teach box como la de la figura 1.1:

Figura 1.1. Teach box

Este aparato se conecta al armario de control mediante un cable de longitud suficiente para recorrer con él todo el espacio de trabajo y contiene un display y unos cuantos botones. Con estos últimos, un operador puede controlar la posición y orientación del extremo del robot o la posición de cada una de sus articulaciones, siendo estos valores indicados de forma instantánea a través del display, que generalmente es de pantalla de cristal líquido. Solo se puede controlar una variable a la vez. Un operador puede utilizar este aparato para mover el extremo del robot sobre una trayectoria deseada y, con solo pulsar un botón, solicitar a la computadora controladora que registre las posiciones a lo largo de esa trayectoria. A algunos robots, de hecho, se les insertan unos movimientos determinados para que sean efectuados una vez se pulse un botón de la teach box [1]. Este método es el que se utiliza, por ejemplo, en los robots industriales de pintura, como el de la figura 1.2:

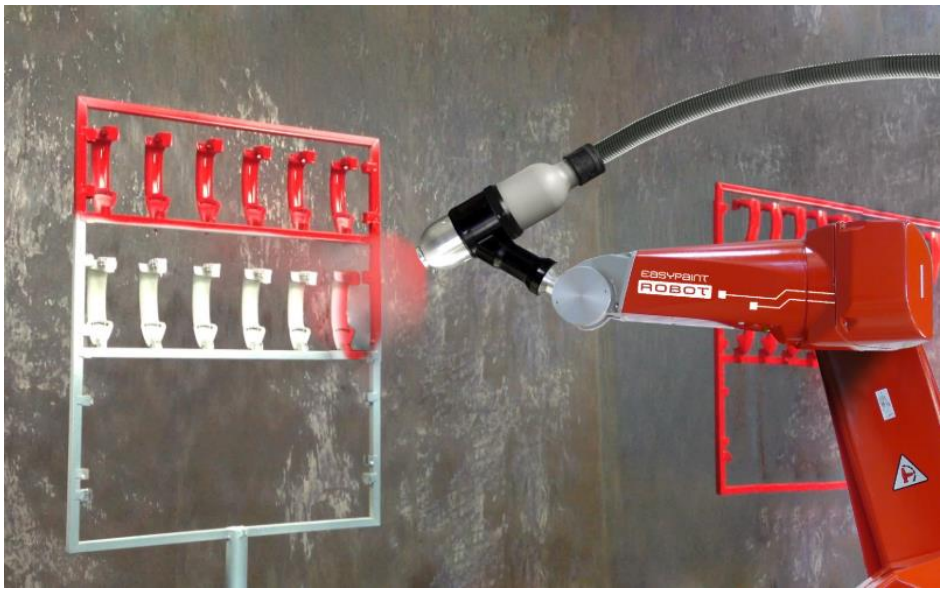

Figura 1.2. Robot industrial de pintura

La mayoría de los robots pueden ser programados utilizando un lenguaje de programación. Usualmente, el código del programa se introduce a través de un ordenador, aunque los investigadores están experimentando la generación automática de programas a partir de modelos de procesos. Esta programación que se implanta al aparato previamente a su uso para el manejo del robot, se llama programación off-line.

Los botones de esta teach box, que normalmente son de membrana, se pueden dividir en 4 grupos:

- Teclas multifunción. Suelen ser pocas, menos de 5. Se suelen situar justo debajo de la pantalla. Lo que hace cada una de ellas se suele indicar en la última línea del display.

- Teclado alfanumérico. Son siempre 10 teclas, las cuales van directamente asociadas a los números del 0 al 9. Para introducir caracteres, sin embargo, se suele utilizar alguna combinación asociada a las teclas de función.

- Teclas relacionadas con el movimiento manual. Estas a su vez se dividen en teclas para modificación de parámetros de movimiento, con las que selecciona el sistema de referencia o la velocidad de ejecución; y en teclas de movimiento, con las que se consiguen efectos de desplazamiento y giro en ambos sentidos de movimiento en un espacio 3D, por lo que el número de estas teclas puede llegar hasta 12. Algunas teach box llevan implantadas un joystick para mover el brazo y así no tener que usar los botones.

- Teclas de ejecución del programa. Con estas teclas se arranca o para el programa, y estas suelen tener un color o tamaño que las hacen destacar sobre el resto.

Además de estas teclas, estos aparatos también suelen tener implantados un botón de parada de emergencia, LED indicadores del estado del robot, y un interruptor de parada

de hombre muerto, el cual sirve para que el robot solo funcione cuando se esté pulsando este interruptor, y, de esta manera, si el operador se desmayase o le pasara algo de ese estilo, el robot dejaría de funcionar.

El uso combinado de la programación off-line y la programación manual (mediante la teach box), constituyen el método más habitual de programación de robots. Esto dota al robot de una de sus grandes características: la flexibilidad. Además, las características de la programación y las del robot están directamente relacionadas, ya que su programación determina el uso que se le puede dar al robot. [2]

Pues bien, las mismas acciones que se pueden hacer con una teach box como la de la figura 1.1 se pueden efectuar a través de una interfaz manejable desde un ordenador, lo cual es el objetivo de este trabajo.

1.2. Objetivos del trabajo

El objetivo de este proyecto trata de manejar un brazo robótico situado en el departamento de Ingeniería de Sistemas y Automática de la UPCT a través de una interfaz que debe ser creada previamente con ayuda de un lenguaje de programación. En este caso se hará uso de Matlab, habiéndoseme ofrecido previamente el hacer el trabajo con lenguaje C o con Matlab. El proyecto se divide en varias etapas:

1. Diseño de la interfaz con la que posteriormente se manejará el robot con ayuda del lenguaje de programación Matlab, el cual cuenta con la función GUI, que permite diseñar de forma fácil e intuitiva una interfaz que pueda servir de herramienta de conexión entre una máquina y un usuario. Matlab es un lenguaje de programación que se utiliza mucho en robótica, ya que en esta se hacen cálculos con matrices, ámbito para el cual Matlab está especialmente preparado (de hecho, Matlab es una abreviatura de MATrix LABoratory).

2. Familiarizarse con la arquitectura del robot y con sus posibles funciones, es decir, con su hardware y con su software. El manejo del robot se efectuará con Matlab a través de funciones de la librería m5apiw32, la cual está proporcionada para controlar los módulos de PowerCube, programa que permite controlar y configurar cada una de las articulaciones del robot.

3. Resolución del problema de la cinemática directa del robot. Con ayuda de esta, siendo conocida la posición de todas las articulaciones, se podrá conocer la posición y orientación del extremo del robot con respecto a su base, que se mostrará en la interfaz creada e irá variando conforme se muevan una o varias articulaciones del robot, puesto que la resolución del problema se implantará en el programa.

4. Resolución del problema de la cinemática inversa del robot. Se abordará el problema de saber qué valor necesita tener cada coordenada articular para poder llegar el extremo del robot a una posición determinada con una orientación determinada. Esta resolución se implantará en el programa para que la muñeca

del robot pueda generar trayectorias rectilíneas en una de las 3 direcciones de los 3 ejes coordenados, manteniendo su extremo su orientación.

5. Obtención de la matriz Jacobiana del robot. Se obtendrá la matriz Jacobiana directa del robot para poder conocer la velocidad de su extremo a partir de la velocidad conocida de cada una de sus articulaciones.

6. Asociar cada elemento de la interfaz con la acción que debe hacer el robot o el programa a la hora de interactuar sobre él. Se comprobará que todo funciona sin ningún problema y que el desarrollo de la consola virtual para el manejo del brazo robot se ha efectuado correctamente.

En el transcurso del proyecto se aprenderá a utilizar un lenguaje de programación del que no se hace mucho uso en el grado de GITI en la UPCT y que puede resultar interesante conocer en mediana profundidad como es Matlab. Además, también servirá para familiarizarse con varios conceptos básicos de robótica, un ámbito interesante del que no se habla nada en el grado de GITI.

Capítulo 2 : Implementación de la consola virtual

En este capítulo se hablará tanto de la función GUI de Matlab, la cual permite crear la interfaz que se necesita crear para el manejo del robot, como de las características de la interfaz creada para el manejo de este. Además, se hablará de algunas funciones de la librería m5apiw32 que se ejecutan a la hora de pulsar sobre algún botón de la interfaz.

2.1. Función GUI de Matlab

El lenguaje de programación Matlab cuenta con la función GUI (Grafical User Interface), que es un editor de interfaces de usuario. Con esta función se puede diseñar de forma fácil una interfaz gráfica que permita una fácil comunicación persona-máquina, es decir, poder hacer que un programa o, en este caso, un robot, haga algo con tan solo tocar un botón o introducir algo por teclado, además de poder mostrar en la misma interfaz resultados útiles de forma clara para el usuario.

Gracias a esta interfaz gráfica, será el usuario quien pueda elegir el orden de ejecución de un código, previamente desarrollado y asociado a esta interfaz, de manera intuitiva (o al menos es lo que se pretende, que sea intuitivo). Es decir, el código no se ejecutará de la manera ordenada en la que lo haría si no existiera esta interfaz.

Para poder acceder a la función GUI, tan solo hay que escribir “guide” en la ventana de comandos de Matlab, lo que da la posibilidad de crear una nueva interfaz, para lo cual se parte de algo parecido a la figura 2.1:

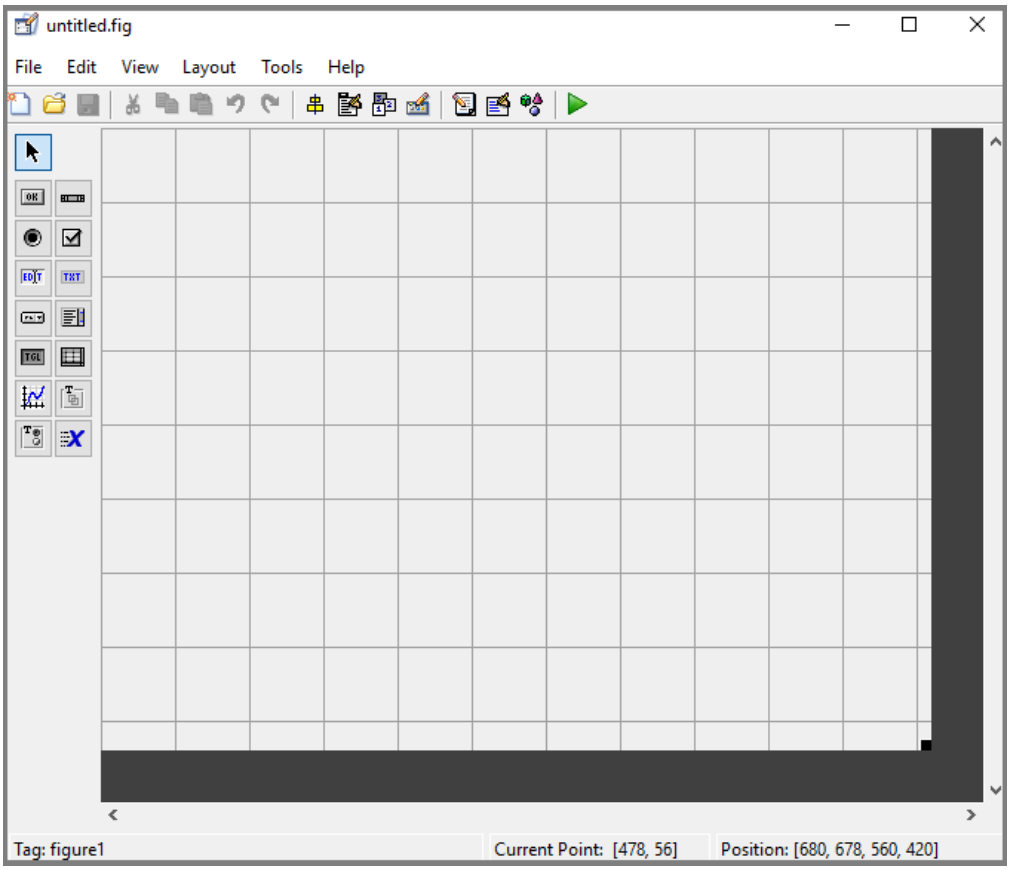

Figura 2.1. Función GUI de Matlab

Partiendo de esto, se pueden arrastrar desde el panel de la izquierda diversos elementos, de los que va a estar compuesta la interfaz. Los diferentes elementos que se pueden utilizar se pueden ver representados en la figura 2.2, los cuales van numerados y se van a describir en la tabla 2.1:

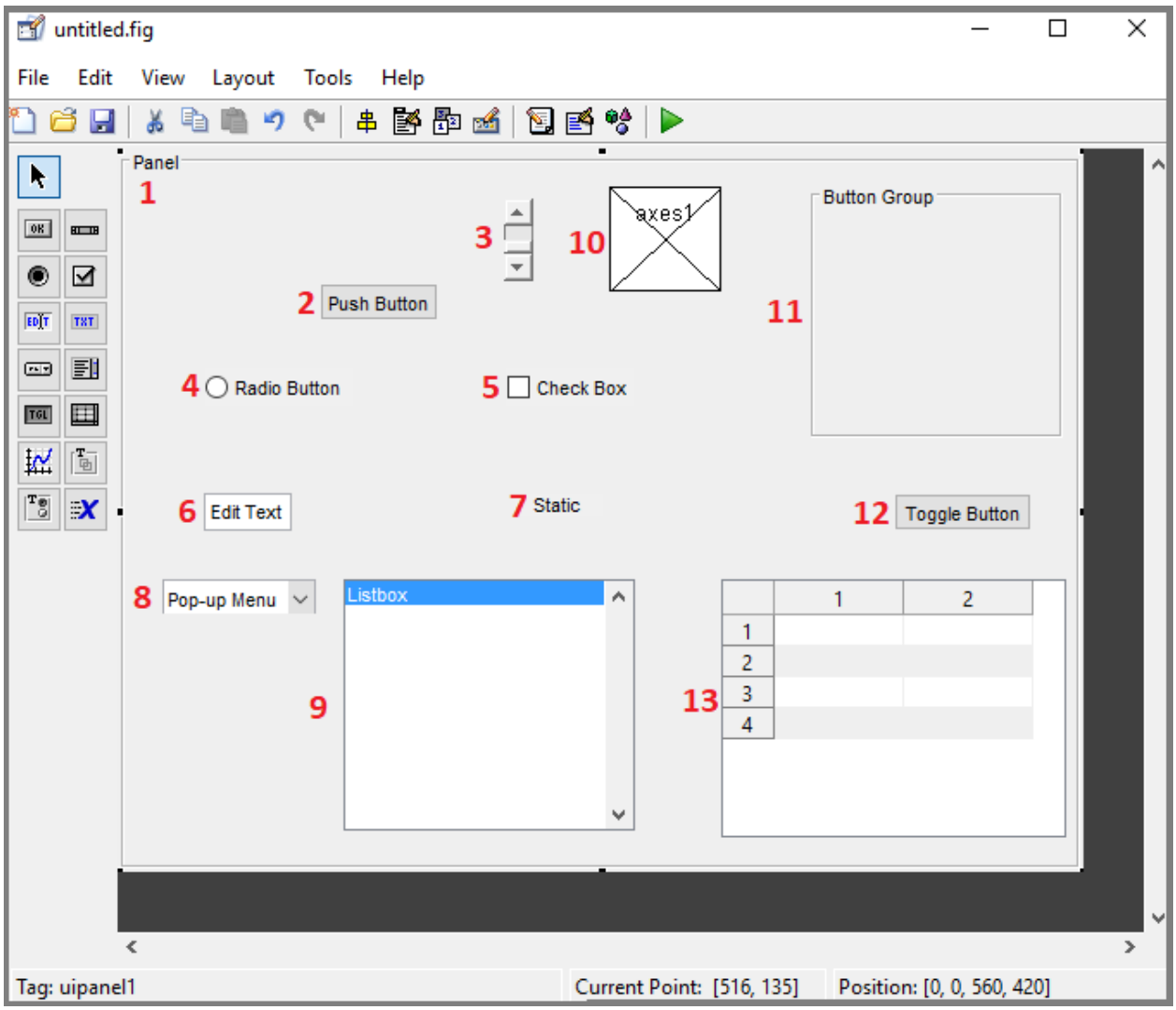

Figura 2.2. Posibles elementos de la interfaz

Elemento	Nombre	Descripción
1	Panel	Es un cuadro en el que se agrupan varios elementos.
2	Push Button	Se ejecuta la acción especificada al hacer click con el ratón sobre él. Si no se modifica, el botón aparece presionado mientras se mantiene presionado el ratón, y se levanta cuando se deja de pulsar el ratón.
3	Slider	Es una barra deslizante, la cual se puede mover pulsando sobre ella y moviendo el ratón hacia arriba o hacia abajo, o clicando sobre las flechas.

4	Radio Button	Se puede mantener activado o desactivado al pulsar sobre él.
5	Check Box	No presenta diferencia alguna con respecto al Radio Button, excepto si se encuentran varios en un Button Group, lo cual se explicará en su respectivo apartado.
6	Edit Text	Es una casilla de texto que puede ser editada directamente por el usuario. En ella se puede escribir cualquier cadena de caracteres, con la condición de que solo se puede escribir en una línea.
7	Static Text	Es una casilla de texto no editable directamente por el usuario. Es como un texto permanente de la interfaz.
8	Pop-up Menu	Es un menú desplegable en el que el usuario elige la opción que necesita.
9	Listbox	Es parecido al Pop-up Menu, con la diferencia de que es un menú no desplegable, apareciendo todas las opciones a vista del usuario.
10	Axes	En este elemento se puede tanto representar una gráfica como representar una imagen guardada.
11	Button Group	Es un panel en el que, si se colocan varios Radio Buttons y Toggle Buttons, solo puede quedar activado uno de ellos.
12	Toggle Button	Es como un Push Button, con la diferencia de que, una vez se cliquea sobre él, se mantiene presionado hasta que se vuelve a cliquear sobre él, momento en el que se vuelve a quedar despulsado.
13	Table	Es una tabla no editable directamente por el usuario.

Tabla 2.1. Descripción de los elementos de la interfaz

Además, cada uno de estos elementos tiene varios eventos asociados posibles (o Callbacks). Estos eventos se pueden definir como algo que puede ocurrir sobre un elemento de la interfaz. Una vez se invoque un evento asociado a un elemento, ocurrirá algo especificado por el programador. Cada uno de los eventos tiene un nombre determinado, y algunos ejemplos de estos son:

- Callback: Este evento se invoca una vez se pulsa sobre el elemento al que está asociado el evento.

- ButtonDownFcn: Este evento se invoca cuando se pulsa un botón del ratón sobre su elemento asociado. No hace falta que se despulse para que el evento tenga lugar.

- WindowButtonDownFcn: Este evento se invoca cuando se pulsa un botón del ratón en cualquier parte de la interfaz. Es un evento asociado a la propia interfaz gráfica.
- WindowButtonUpFcn: Este evento se invoca cuando se despulsa un botón del ratón en cualquier parte de la interfaz. Es un evento asociado a la propia interfaz gráfica.

- CloseRequestFcn: Este evento se invoca cuando se pulsa el botón de cerrar la ventana de su elemento asociado, por ejemplo el de la propia interfaz.

Sabiendo esto, una vez se crea la interfaz gráfica, se elige con qué eventos de cada elemento se va a trabajar dando clic derecho sobre el elemento y pulsando sobre los que sean de interés, como se muestra en la figura 2.3. Hay que tener en cuenta que el evento Callback siempre va a aparecer por defecto en el programa asociado a la interfaz.

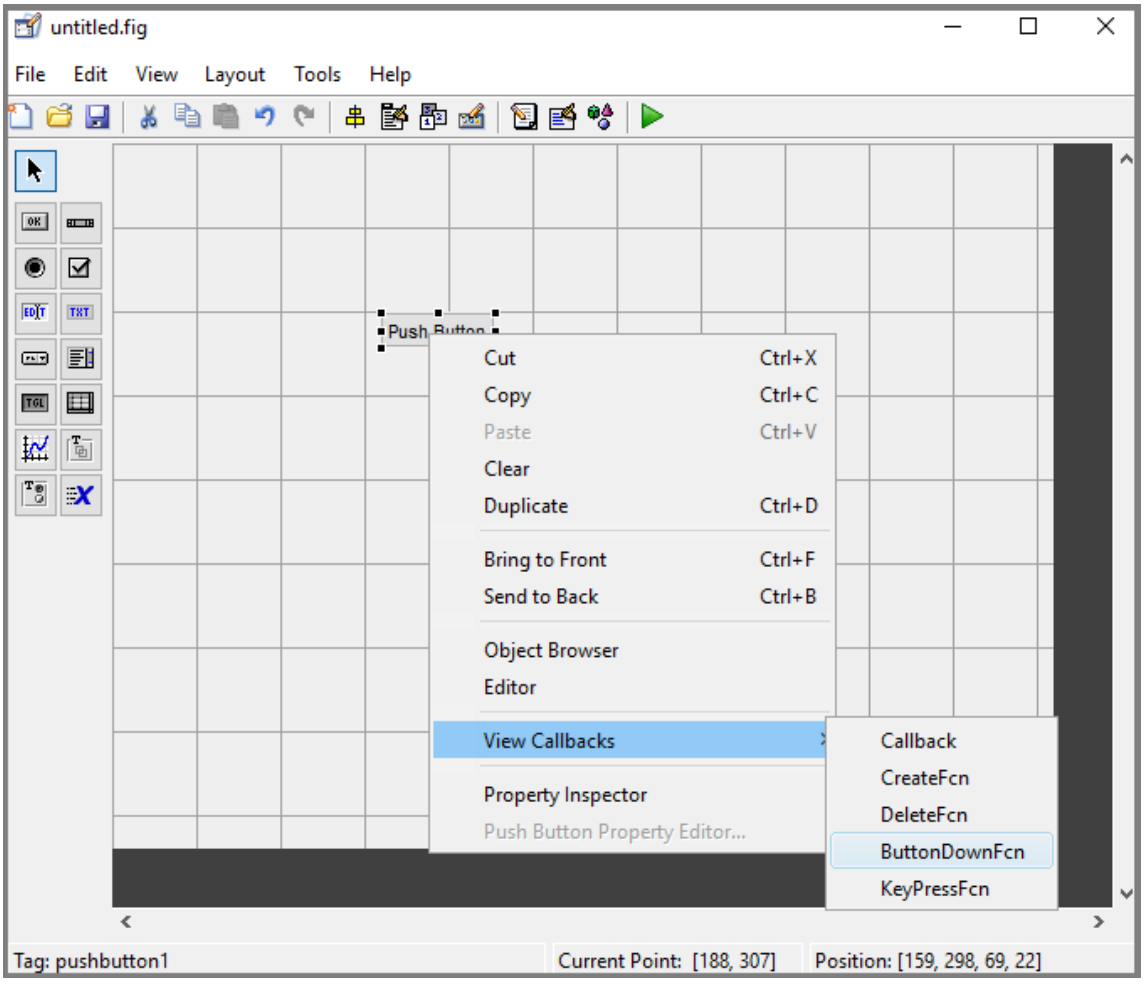

Figura 2.3. Posibles eventos de un control

Una vez seleccionados los eventos a utilizar, se trabajará con el programa asociado a la interfaz y, como se ha dicho previamente, se elegirá qué es lo que pasará cuando se invoque a cada uno de los eventos seleccionados de cada elemento.

Por último, falta nombrar algo bastante importante: el Inspector de propiedades, al cual se puede invocar dando doble clic en un elemento de la interfaz o pulsando clic derecho sobre él y seleccionando la opción “Property Inspector”. Este tiene la apariencia de la figura 2.4:

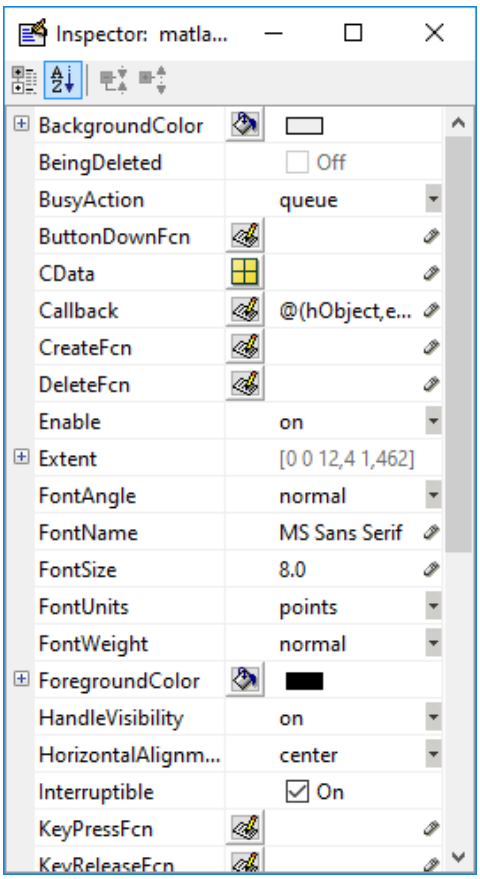

Figura 2.4. Inspector de propiedades

Desde esta opción se pueden modificar varias características de cada elemento de la interfaz. Algunas de las más importantes son:

- BackgroundColor: Se selecciona el color de fondo del elemento. El color que se da por defecto es el gris.

- Enable: Puede quedar marcado como on, off o inactive. La diferencia entre ellos es la siguiente:

    - on: El elemento es operacional, es decir, su evento asociado “Callback” si se invoca cuando se hace clic sobre él.

    - off: El elemento se ve gris y no se puede operar sobre él, es decir, su evento asociado “Callback” no se invoca cuando se hace clic sobre él. Sin embargo, sí funcionan los eventos “ButtonDownFcn” y “WindowButtonDownFcn”, los cuales no tienen efecto cuando “Enable” está marcado como “on”.

    - inactive: Tiene las mismas propiedades que “off”, con la diferencia de que el elemento no se ve gris, sino que tiene la misma apariencia que “on”.

- FontName: Se selecciona el tipo de letra del texto que aparece en el elemento.
- FontSize: Se selecciona el tamaño de letra del texto que aparece en el elemento.
- ForegroundColor: Se selecciona el color del texto que aparece en el elemento.
- Position: Es un vector compuesto por 4 números: los dos primeros indican la posición x e y del elemento con respecto a la esquina inferior izquierda de la interfaz, el tercero el ancho del elemento, y el cuarto su altura.
- String: Se especifica el texto que aparece en el elemento.

En el Inspector de propiedades se pueden ver también los eventos disponibles asociados a cada elemento.

Un punto importante que hay que recalcar sobre esto es que se puede hacer que alguna de estas propiedades de algún elemento se modifique cuando se invoque un evento de cualquier elemento, lo que da un gran juego a la hora de programar la interfaz. Por ejemplo, se puede hacer que la propiedad “String” de los “Edit text” o de los “Static Text” varíe por ejemplo cada vez que se pulse un botón, lo que daría lugar a los resultados que se quieren obtener con esa interfaz.

2.2. Características de la consola creada

Con ayuda de la función GUI de Matlab se ha desarrollado una interfaz a través de la cual se puede manejar el robot del trabajo. Esta interfaz puede entenderse como una consola virtual que cuenta con diversas funciones, las cuales se van a desarrollar en este subcapítulo. El resultado final de la interfaz se puede ver en la figura 2.5:

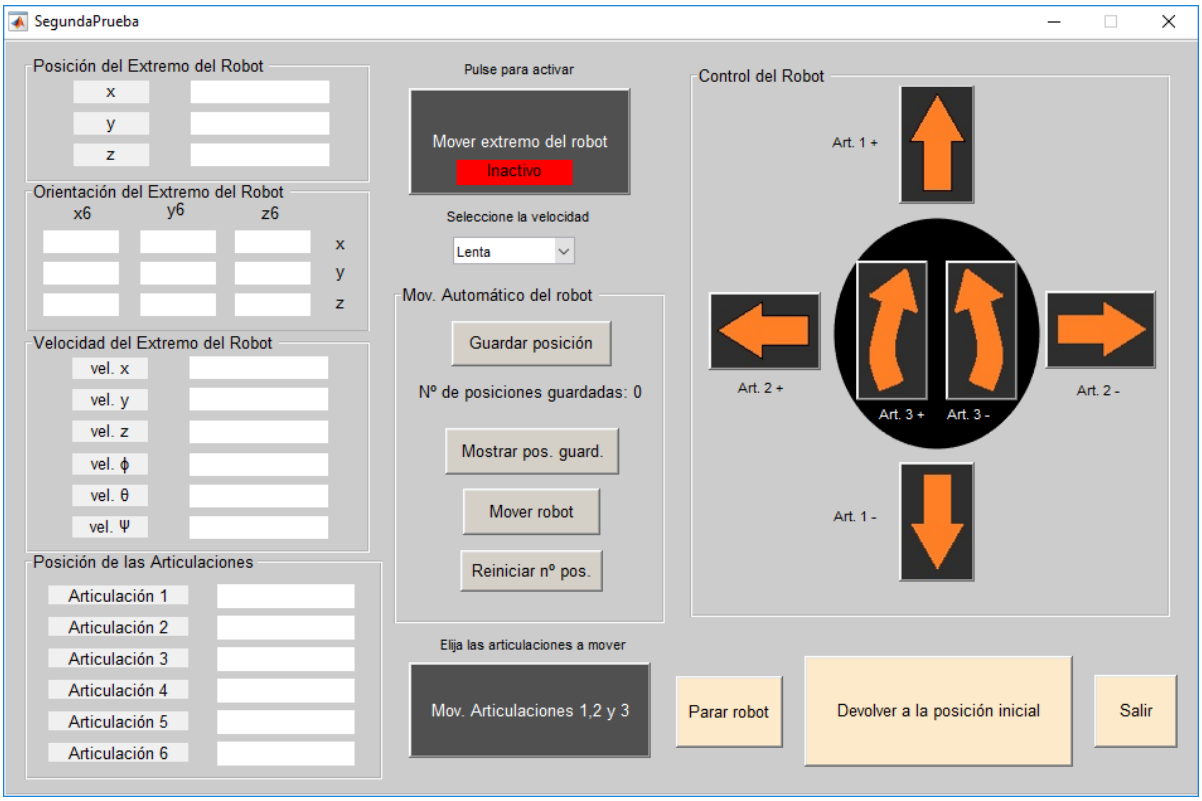

Figura 2.5. Interfaz del robot

Donde se pueden observar diversos controles como botones o cajas de texto, estando varios de ellos metidos dentro de paneles.

Al iniciarse la interfaz, se carga automáticamente la librería m5apiw32, y seguidamente se habilita el funcionamiento del robot con la siguiente función:

```
calllib('m5apiw32','PCube_openDevice', devID, 'ESD:0,1000')
```

Donde se especifica el ID del dispositivo (devID), el Puerto (0), y los kBit/s (1000).

Se procede ahora a explicar el funcionamiento de cada elemento de la interfaz.

2.2.1. Joystick

En el panel llamado “Control del Robot” de la interfaz (arriba a la derecha) se encuentran 6 botones que ejercen la función de un joystick. Cada uno de estos botones tiene insertado un dibujo parecido al de la figura 2.6, dependiendo de la posición del joystick en la que se encuentren:

Figura 2.6. Botón del joystick despulsado

En el caso de los botones del joystick se ha hecho uso del evento de “ButtonDownFcn”, explicado en el subcapítulo 2.1, y no del evento de pulsado (“Callback”), ya que estos botones deben ejercer su función solo cuando se encuentre pulsada una tecla del ratón sobre ellos, debiendo parar cuando se levanta. Esta función va precisamente asociada al evento “ButtonDownFcn”, mientras que su evento “Callback” solo hace efecto cuando se termina de pulsar el ratón sobre el botón, por lo que en este caso no se hace uso de él.

Como se debe hacer uso del evento “ButtonDownFcn”, no se puede dejar su propiedad “Enable” (también explicada en el subcapítulo 2.1) activada (“on”), y se debe dejar en “off” o en “inactive” (en esta interfaz está puesta como “inactive”), por lo que no se podría apreciar el efecto de pulsado sobre el botón. Para arreglar esto, se ha hecho que, cada vez que se pulse sobre el botón, el dibujo de este cambie, quedando como el de la figura 2.7:

Figura 2.7. Botón del joystick pulsado

Y, cuando el ratón se despulse, el botón volverá a estar como el de la figura 2.6. Esto hace que se cumpla de manera satisfactoria el efecto visual de parecer que el botón está pulsado, puesto que el color gris oscuro de fondo y el color naranja de la flecha de la figura 2.6 pasan a ser de los colores negro y rojo, respectivamente, representados en la figura 2.7. Es decir, los colores pasan a ser más oscuros de lo que eran previamente, lo que representa un efecto de pulsado.

Hay que tener en cuenta que las imágenes de las flechas deben estar en el directorio de Matlab, o en el que se esté trabajando para que estas puedan ser invocadas, ya que, si no lo estuvieran, habría un error y no se podrían invocar cada una de las imágenes. Lo mismo pasa con la librería m5apiw32: la carpeta en la que se encuentre tiene que estar en el directorio en el que se esté trabajando, ya que si no sus funciones no podrían ser invocadas y el programa no funcionaría correctamente.

Cerca de cada botón del joystick se puede apreciar una etiqueta que indica la articulación que se va a mover y el sentido en el que lo hará (positivo o negativo). Como se puede comprobar, en principio solo se pueden mover las tres primeras articulaciones de manera individual. Esto se arregla pulsando uno de los dos botones rodeados en azul en la figura 2.8:

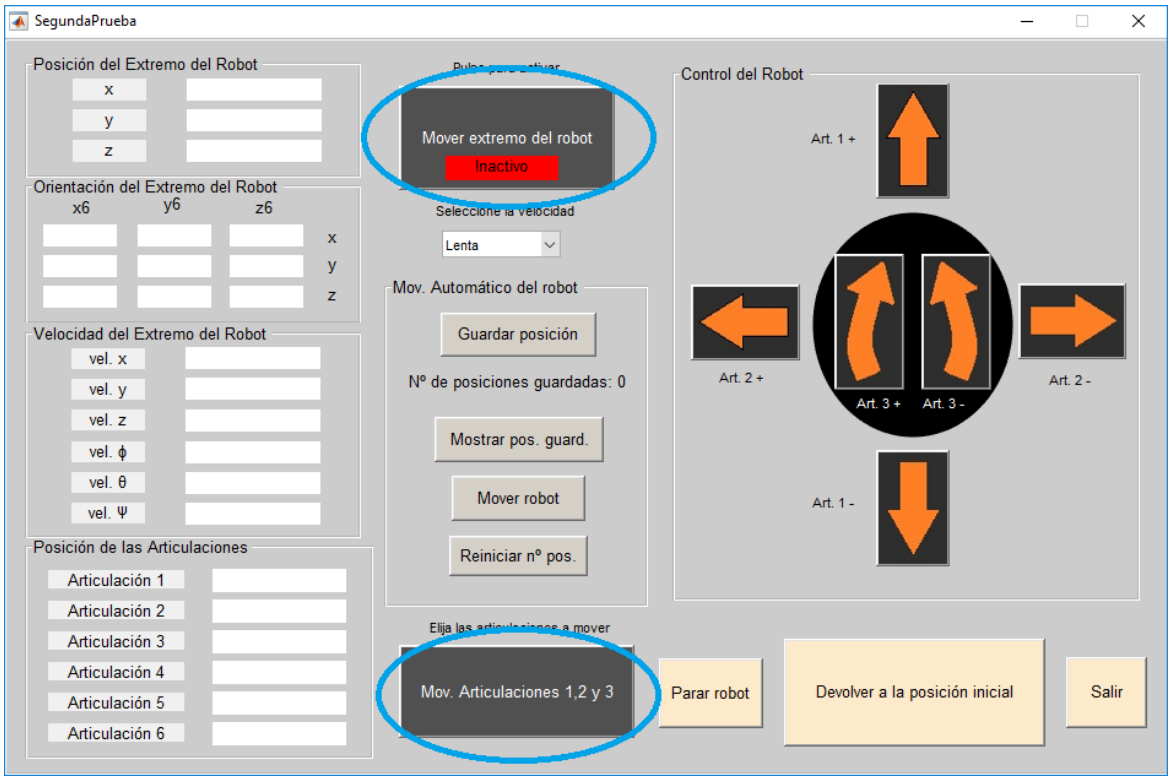

Figura 2.8. Botones para cambiar qué controlar con el joystick

Estos botones son de tipo “Toggle Button”, el cual viene explicado en la tabla 2.1 antes mencionada. El botón de abajo se encargará de que las etiquetas cambien por el nombre de las 3 últimas articulaciones, haciendo que ahora los 6 botones se encarguen de mover estas articulaciones de manera individual, y la interfaz se vería como en la figura 2.9:

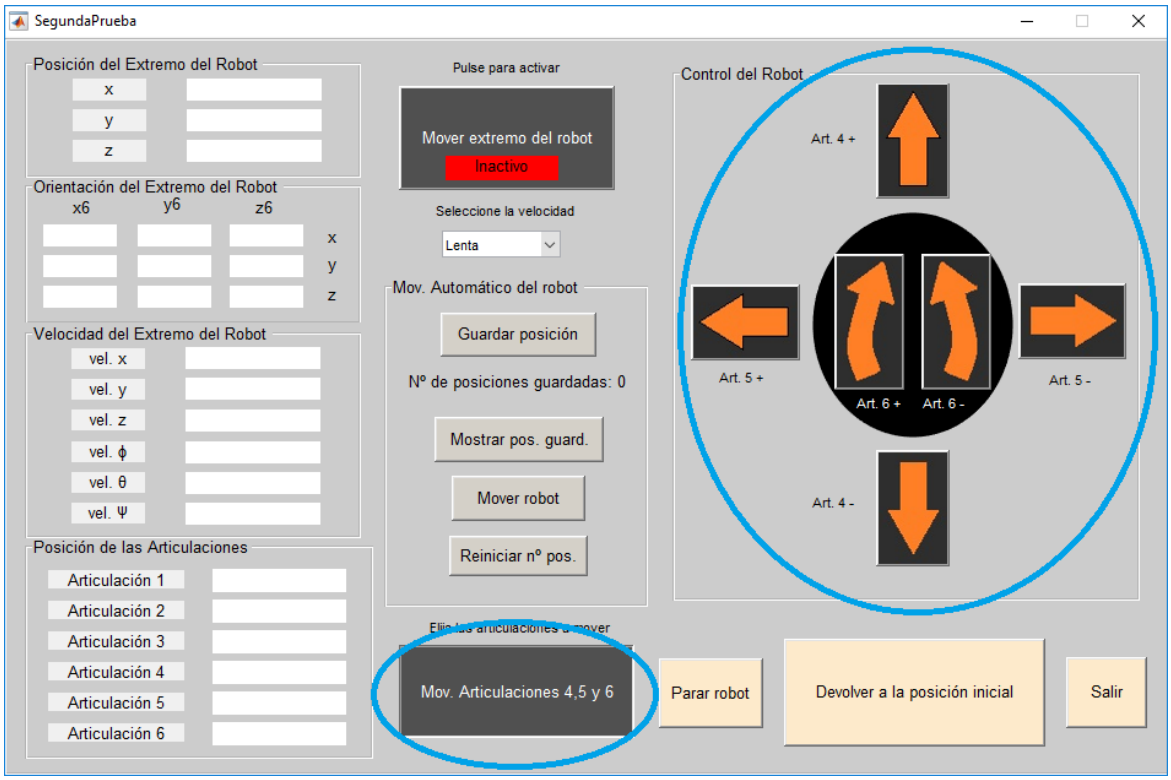

Figura 2.9. Interfaz para el control de las 3 últimas articulaciones

Por otro lado, cuando se pulse el botón de arriba, la palabra “inactivo” con fondo rojo pasará a ser “activo” y su fondo pasará a ser verde. Esto hará que las etiquetas cambien por el nombre de los 3 ejes coordenados (**x**, **y** y **z**), pasando entonces cada botón a controlar el extremo del robot, y, por tanto, moverá varias articulaciones a la vez. La muñeca del robot irá en línea recta en la dirección del eje coordenado que se quiera, y su extremo estará orientado siempre de la misma manera. Además, por más que se pulse el botón de abajo, solo se podrá mover el extremo del robot, pudiendo volver a mover solo una articulación por botón dejando en “inactivo” el botón de arriba. La interfaz se vería como en la figura 2.10:

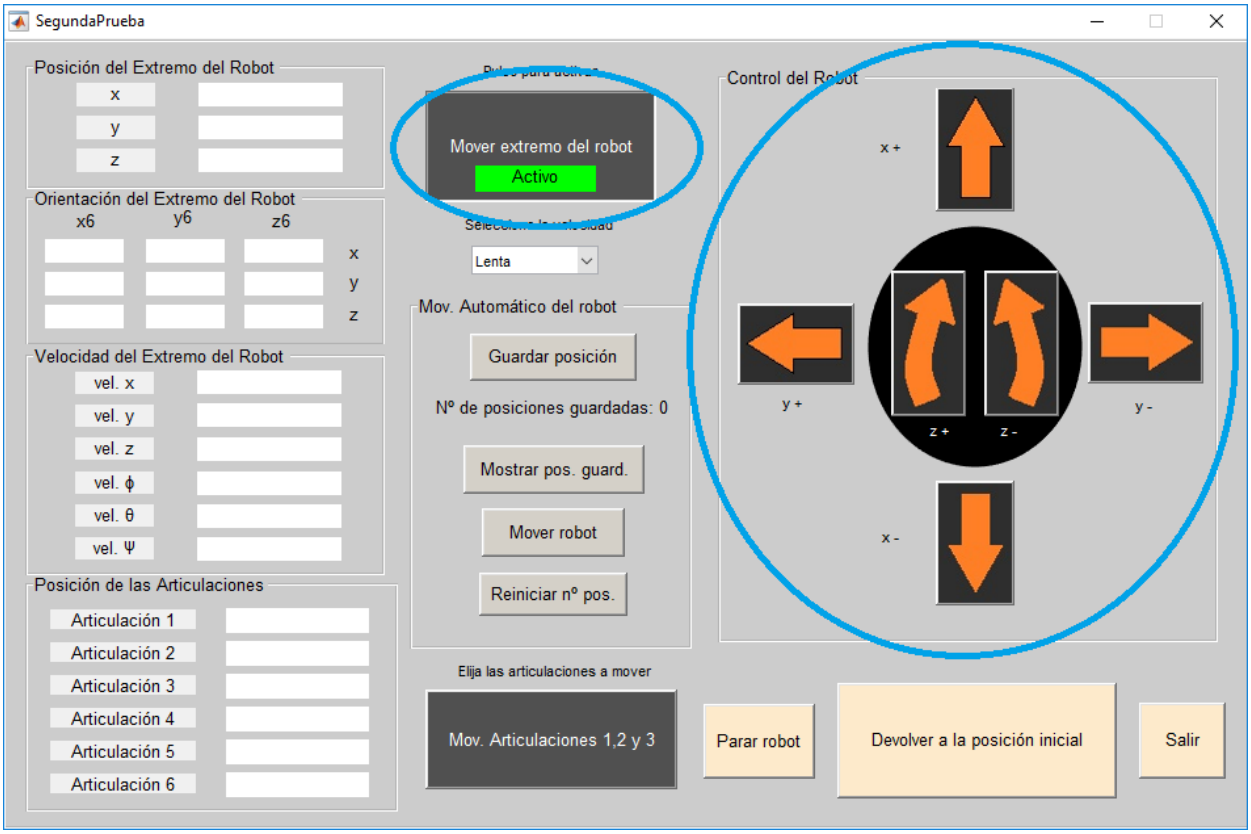

Figura 2.10. Interfaz para el control del extremo del robot

Por último, decir que cada una de las articulaciones tiene un límite de posición superior e inferior. Cuando una de ellas llegue a este límite, ya sea moviendo una articulación individualmente o moviendo el extremo del robot (varias articulaciones a la vez), aparecerá un cuadro de texto como el de la figura 2.11, donde, dependiendo de si llega al límite superior o inferior, indicará que se ha llegado a la máxima o a la mínima posición, respectivamente.

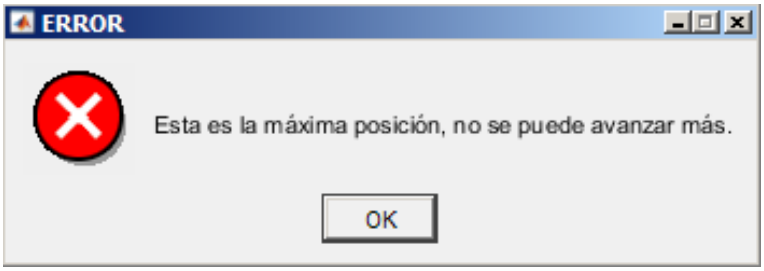

Figura 2.11. Ventana de error

El movimiento de cada una de las articulaciones se puede realizar gracias a una función de la librería m5apiw32. El movimiento de cada articulación por separado se hace con un movimiento en rampa, mientras que a la hora de hacer la trayectoria rectilínea del extremo del robot se hace un movimiento a velocidad constante. Esto se desarrollará más detenidamente en el capítulo 6. Pero sí se señalarán las funciones utilizadas de la librería: para hacer un movimiento en rampa se utiliza la siguiente función:

```
calllib('m5apiw32','PCube_moveRamp', devID, modID, pos, vel, acel)
```

Donde se especifica el ID del dispositivo, el ID del módulo, la posición en radianes a la que se debe mover, la velocidad en rad/s, y la aceleración en rad/s^2.

Por otro lado, para el movimiento a velocidad constante se utiliza la función:

```
calllib('m5apiw32','PCube_moveVel', devID, modID, vel)
```

Donde se especifica el ID del dispositivo, el ID del módulo, y la velocidad en rad/s.

Además, antes de ejecutar cualquier movimiento se deben resetear todos los módulos, ya que, si esto no se hace, puede que alguno no funcione correctamente. Esto se hace con ayuda de la siguiente función:

```
calllib('m5apiw32','PCube_resetAll', devID)
```

Donde se especifica el ID del dispositivo.

2.2.2. Posición del extremo del robot, matriz de rotación, velocidad del extremo del robot y posición de las articulaciones

A la izquierda de la interfaz se pueden encontrar 4 paneles distintos, todos ellos con etiquetas y cajas de texto.

Empezando por arriba, el panel llamado “Posición del Extremo del Robot” muestra la posición en el espacio del extremo del robot con respecto al sistema de referencia de la base del robot (en milímetros). Esto se calcula con ayuda de la cinemática directa, la cual se presenta en el capítulo 4. Por el momento se puede adelantar que estos resultados vienen dados a partir de los valores de las coordenadas articulares.

El siguiente panel, llamado “Matriz de Rotación”, muestra la orientación del extremo del robot con respecto a la base del robot. La fila de arriba (**x6**, **y6**, **z6**) se refiere a los ejes del extremo, mientras que la última columna (**x**, **y**, **z**) se refiere a los ejes del sistema de referencia. Estos valores se obtienen también a través de la cinemática directa. Es más, los resultados de estos dos primeros paneles se extraen de una misma matriz, pero eso ya se comentará más adelante.

Tras este, se puede encontrar el panel con el nombre de “Velocidad del extremo del robot”, donde se podrá ver indicada la velocidad del extremo del robot en la dirección de cada uno de los tres ejes coordenados del sistema de referencia de la base del robot, tanto la lineal en mm/s como la angular en rad/s. Estos valores se obtienen a partir de la matriz Jacobiana directa y de la velocidad de cada articulación, lo cual viene explicado en el capítulo 7.

Por último, se puede encontrar el panel llamado “Posición de las articulaciones”, donde, como su propio nombre indica, se puede encontrar el valor de cada una de las coordenadas articulares. Estos valores vienen dados en grados, y se obtienen haciendo la lectura de los encoders[1] de cada articulación con ayuda de una función de la librería m5apiw32, que es la siguiente:

```
calllib('m5apiw32','PCube_getPos', devID, modID, *pos)
```

La cual apunta con un puntero a “pos” y devuelve el valor de la posición de la articulación en radianes. En esta función se especifica tanto el ID del dispositivo, como el ID del módulo.

Además, también se puede obtener la velocidad de cada articulación en cada momento con esta función:

```
calllib('m5apiw32','PCube_getVel', devID, modID, *vel)
```

La cual apunta con un puntero a “vel” y devuelve el valor de la velocidad de la articulación en rad/s. En esta función se especifica tanto el ID del dispositivo, como el ID del módulo. Esta función se utiliza más que nada para saber la velocidad de cada articulación a la hora de realizar un movimiento en rampa, puesto que a la hora de acelerar y decelerar la velocidad no es constante.

Los valores de estos cuatro paneles se actualizan cada 50 milisegundos, que es un valor que se ha puesto al azar. Se puede hacer que se actualicen cada más o cada menos tiempo.

2.2.3. Movimiento automático del robot

El panel del medio de la interfaz tiene la función de poder mover automáticamente el robot a partir de unas posiciones guardadas previamente.

Para hacer uso de esta función, el usuario tiene que darle al botón de “Guardar posición” para guardar una posición del robot por la que se quiere que este pase en el movimiento automático. Debajo de este botón aparecerá el número de posiciones guardadas que se tenga en ese momento. Este número de posiciones guardadas puede reiniciarse pulsando sobre el botón “Reiniciar nº pos.”, lo que pone el número de posiciones guardadas a 0.

Pulsando sobre el botón “Mostrar pos. guard.” se puede ver una nueva interfaz con la apariencia de la figura 2.12:

[1] Un encoder es un dispositivo de detección que convierte el movimiento en una señal eléctrica que puede ser leída por algún tipo de dispositivo de control en un sistema de control de movimiento. Este envía una señal de respuesta que puede ser utilizado para determinar la posición, velocidad o dirección. [6]

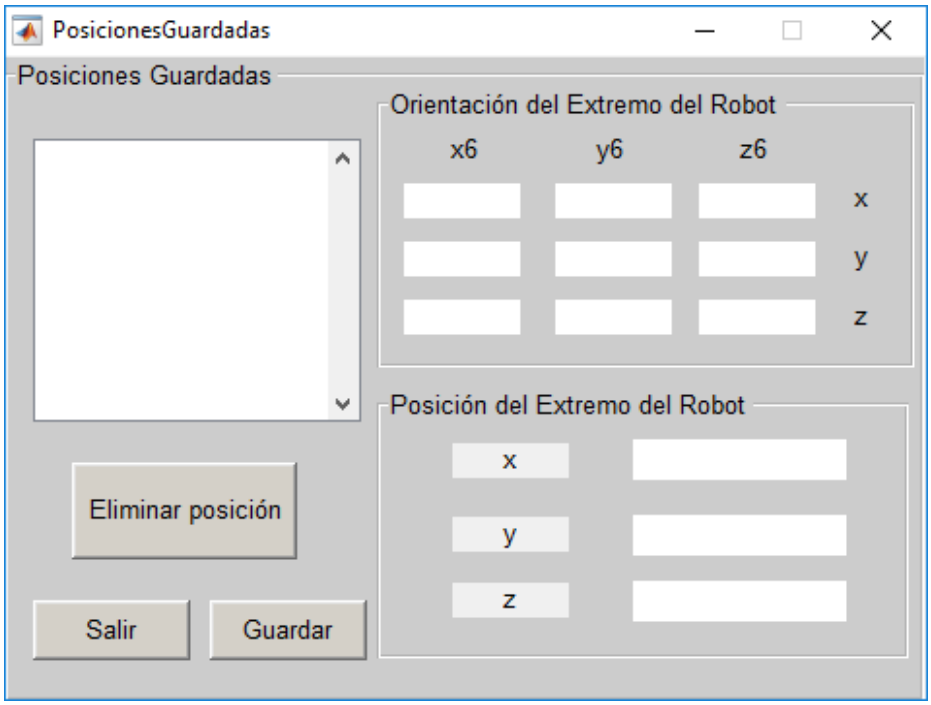

Figura 2.12. Interfaz del movimiento automático del robot

En esta interfaz, en la "Listbox" de arriba a la izquierda pueden observarse las distintas posiciones que se han ido guardando. Pulsando en cada posición, se puede observar su posición del extremo del robot y su orientación correspondientes con respecto al sistema de referencia de la base en los paneles de la derecha. Pulsando sobre el botón "Eliminar posición" se puede eliminar la posición que se desee, y pulsando el botón "Guardar" se guardarán los cambios, actualizándose también el número de posiciones guardadas de la interfaz principal. Si se pulsa el botón "Salir" y se han realizado cambios sin haberlos guardado, aparecerá un cuadro de texto donde aparecerá la pregunta de si se quieren guardar los cambios, pudiendo aceptar o no (figura 2.13).

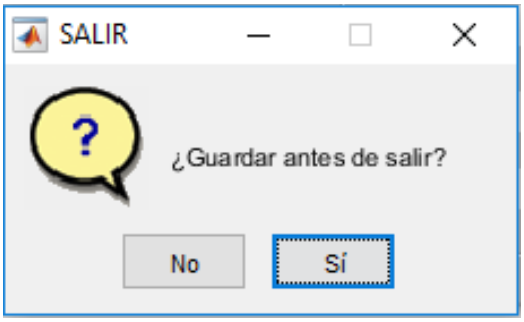

Figura 2.13. Ventana para guardar o no los cambios

Volviendo a la interfaz principal, pulsando sobre el botón "Mover robot", el robot se moverá automáticamente y pasará por todas y cada una de las posiciones guardadas, haciendo una parada de 1 segundo cada vez que llegue a una de estas posiciones, como modo de indicar que ya ha pasado por esa posición. Con esta función, todas las articulaciones que necesiten moverse lo harán al mismo tiempo, siendo unas más rápidas que otras, dependiendo de la distancia que tenga que recorrer cada una.

2.2.4. Otros controles

Aún quedan algunos controles por comentar de la interfaz. Uno de ellos es el “Pop-up menú”, donde se puede elegir la velocidad que se prefiera que tenga el robot. Existen las opciones de “Lenta”, “Normal” y “Rápida”. La lenta corresponde a 0.05 rad/s, la normal a 0.1 rad/s, y la rápida a 0.5 rad/s. En los casos en los que tengan que moverse varias articulaciones a la vez, la velocidad de la más rápida será la correspondiente a la del “Pop-up menú”.

Por otro lado, si se pulsa en el botón “Devolver a la posición inicial”, cada una de las articulaciones vuelve a su posición inicial, es decir, a la posición en el que el valor de su coordenada articular valga 0 º. Este movimiento se hace de forma individual y por orden, siendo la primera articulación en volver a su posición inicial la 1, y la última la 6.

Por último, falta por comentar los botones “Parar robot” y “Salir”. Ambos botones ejercen la acción que su propio nombre indica: el primero detiene todo movimiento del robot con ayuda de una función de la librería m5apiw32, y el segundo sale de la interfaz del robot, no sin antes preguntarle al usuario a través de un cuadro de texto si está seguro de que desea salir, pudiendo aceptar o no. La función que hace parar todo movimiento del robot es la siguiente:

```
calllib('m5apiw32','PCube_haltAll', devID)
```

En la cual se especifica el ID del dispositivo.

Además, a la hora de salir de la interfaz, se deshabilita el funcionamiento del robot con la ayuda de esta función:

```
calllib('m5apiw32','PCube_closeDevice', devID)
```

En la cual se especifica el ID del dispositivo.

Hay que comentar también que si el robot, por cualquier cosa, se volviera incontrolable y el botón de “Parar robot” no funcionase, habría que hacer uso del botón de parada de emergencia. Al ser pulsado, este deja al robot en el estado de parada de emergencia y corta el suministro de energía a todos los módulos del robot, por lo que el robot para de funcionar completamente. Este botón, que tiene la forma de la figura 2.14, siempre se debe tenerlo cerca por si ocurre alguna incidencia de este tipo.

Figura 2.14. Botón de parada de emergencia

Para hacer que se vuelva a suministrar energía a cada módulo del robot, es decir, para quitarle el estado de parada de emergencia y pueda volver a funcionar, se deben seguir los siguientes pasos [4]:

1. Comprobar que la situación peligrosa que causó que se pulsara el botón ha desaparecido.

2. Tirar del interruptor del botón haciendo un giro sobre él.

3. Presionar sobre el botón amarillo del panel de control, el cual tiene la forma de la figura 2.15:

Figura 2.15. Botón de activación del robot

Al estado de parada de emergencia del robot también se puede llegar pulsando el botón de la figura 2.15, pero es peligroso porque para pulsarlo hay que acercarse al robot, ya que este está en el panel de control, justo al lado del robot. El panel de control es como el de la figura 2.16, y su interior es como el de la figura 2.17:

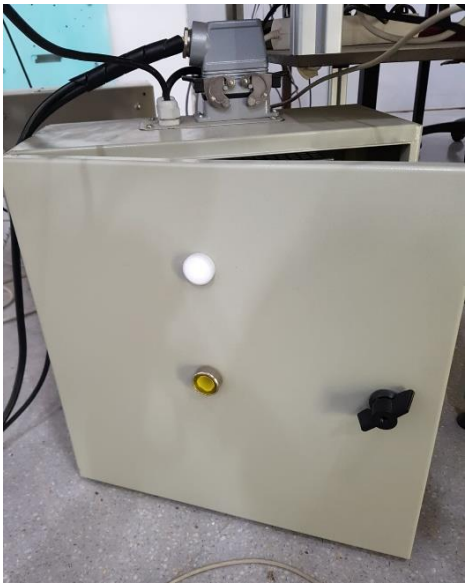

Figura 2.16. Exterior del panel de control

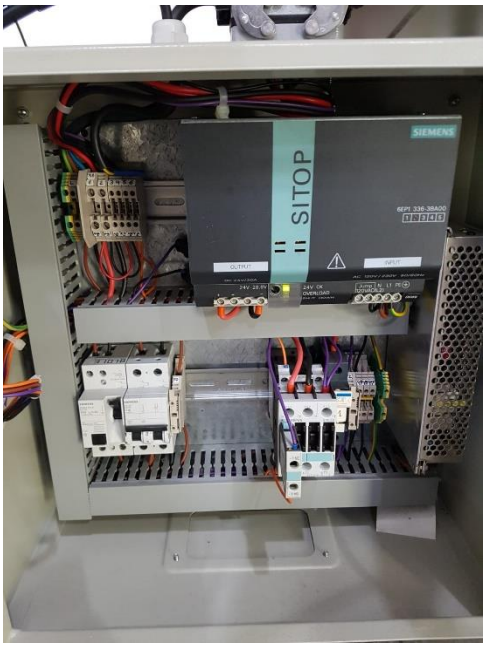

Figura 2.17. Interior del panel de control

Capítulo 3 : Descripción del robot

En este capítulo se hablará tanto del hardware del robot y sus características, como de su zona de trabajo. Además, se señalará una serie de medidas de precaución que habrá que tomar a la hora de hacer uso del robot.

3.1. Partes del robot

El robot a manejar se llama Robotnik y es un brazo robot modular. Este cuenta con 6 grados de libertad, y, por tanto, con 6 articulaciones, todas ellas rotativas[1]. La forma de este se puede ver en la figura 3.1:

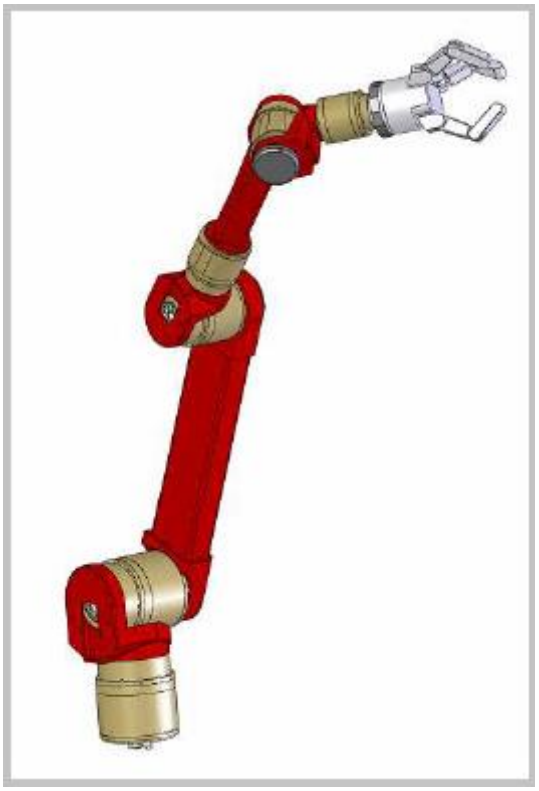

Figura 3.1. Brazo modular Robotnik [4]

Donde cada parte de color marrón claro representa una articulación. Empezando desde abajo, las dos primeras articulaciones simulan los movimientos del hombro, las dos siguientes los del codo, y las dos últimas los de la muñeca. Cada una de estas une dos eslabones, los cuales se van a representar a continuación.

La primera articulación une los eslabones 0 y 1, los cuales vienen representados por las figuras 3.2 y 3.3, respectivamente:

[1] Hay que saber distinguir entre 2 tipos de articulaciones: las rotativas y las prismáticas. Las rotativas son articulaciones giratorias, mientras que las prismáticas son articulaciones que se desplazan en línea recta.

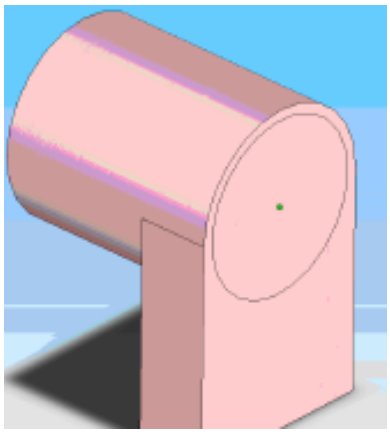

Figura 3.2. Eslabón 0 [4]

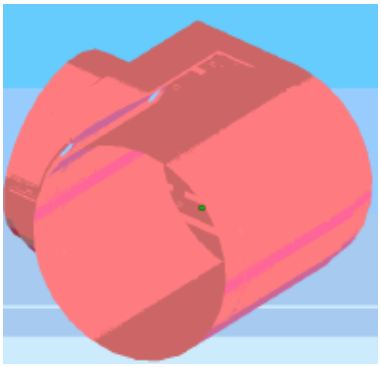

Figura 3.3. Eslabón 1 [4]

La articulación 2 une el eslabón 1 y con el eslabón 2 (figura 3.4):

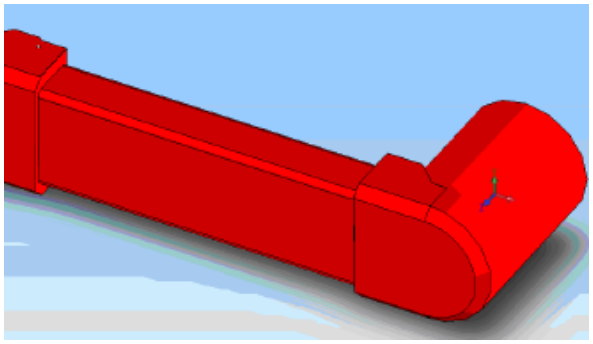

Figura 3.4. Eslabón 2 [4]

La articulación 3 une el eslabón 2 con el eslabón 3 (figura 3.5):

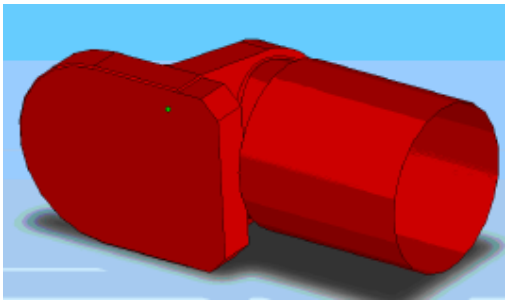

Figura 3.5. Eslabón 3 [4]

La articulación 4 el eslabón 3 con el 4 (figura 3.6):

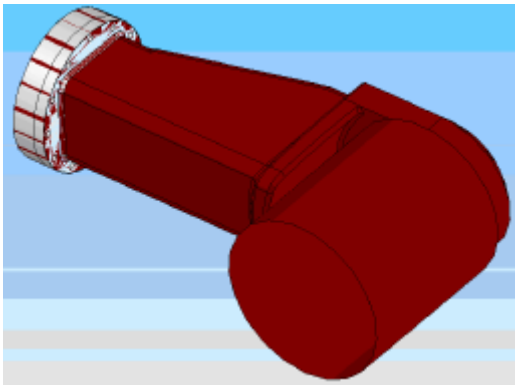

Figura 3.6. Eslabón 4 [4]

La articulación 5 el eslabón 4 con el 5 (figura 3.7):

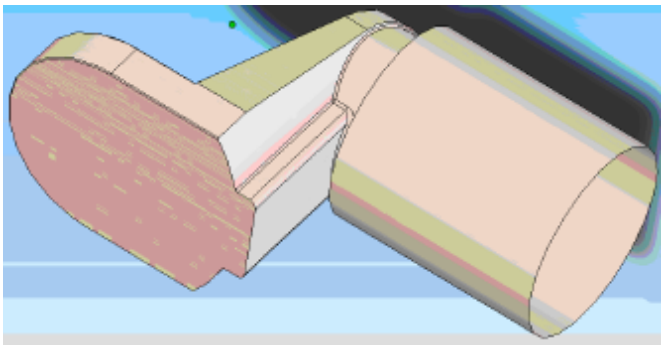

Figura 3.7. Eslabón 5 [4]

Y, por último, la articulación 6 enlaza el eslabón 5 con la herramienta del robot (figura 3.8):

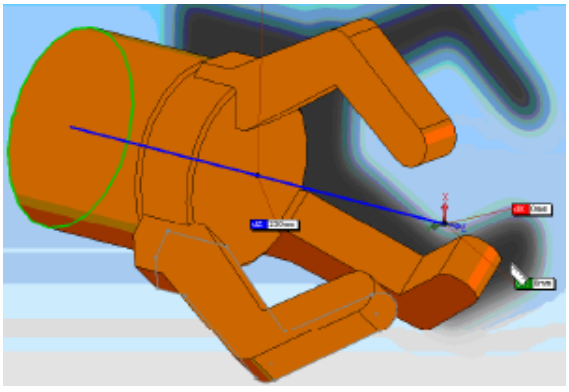

Figura 3.8. Herramienta del robot [4]

La herramienta del robot no se tendrá en cuenta para el desarrollo del proyecto, y por tanto no se tendrán en cuenta las medidas de esta para nada, excepto para determinar los límites de posición de cada una de las articulaciones, ya que esta puede chocarse contra el suelo.

Estas 6 articulaciones corresponden a distintos módulos. Estos módulos se interconectan con el ordenador con el que se controla al robot con una red CAN Bus.

3.2. Zona de trabajo del robot

Los límites superior e inferior de cada coordenada articular están fijados siguiendo tres criterios:

- Que no puedan ocurrir auto colisiones, ya que, si no se corrigiera, para unos determinados valores de cada articulación el robot podría chocarse contra sí mismo.

- Que no haya soluciones redundantes. Por ejemplo, la primera articulación podría girar sin límites; sin embargo, como solo interesa que recorra los 360º y no más para que no se repitan soluciones, los límites superior e inferior se fijan en +/-180º. Habría que tener en cuenta que los cables internos del robot impondrían también una restricción si no se pusiera ningún límite.

- Partiendo de la posición inicial del robot en la que todas las coordenadas articulares tienen el valor de 0 (que corresponde al brazo totalmente extendido en la dirección del eje **x)**, que no pueda colisionar con ningún objeto del área de trabajo en el que se encuentra si se moviese cada articulación individualmente. Por ejemplo, el límite inferior de la articulación 2 es -50º (mientras que el superior es de 126º) debido a que el brazo no está muy lejos del suelo y, si no se pusiera ningún límite, este colisionaría con él. Hay que tener en cuenta que, si se partiese de una posición no inicial, el brazo podría chocarse contra algún objeto (por ejemplo contra una mesa que hay cerca del robot o contra el mismo suelo), ya que las posiciones posibles del robot son muchísimas y para controlar esto habría que hacer un estudio profundo sobre el tema.

Cada vez que el robot llegue a uno de estos límites, ya sea moviendo el extremo del robot o moviendo cada articulación individualmente, saldrá un mensaje de error en la interfaz de la consola del robot que indique que no se puede avanzar más. Los límites fijados para cada articulación se pueden ver en la tabla 3.1.

Articulación	Posición mínima	Posición máxima
1	-180º	180º
2	-50º	126º
3	-143.24º	143.24º
4	- 180º	180º
5	-90º	90º
6	- 180º	180º

Tabla 3.1. Límites de posición de las articulaciones del robot

3.3. Precauciones que hay que tomar con el robot

El Brazo Modular Robotnik puede llegar a tomar una velocidad de 1 m/s, por lo que, a la hora de trabajar con él, se deben tomar una serie de precauciones para garantizar la seguridad [4]:

- No se debe pasar por la zona cercana al robot mientras este esté encendido.
- Mantener el botón de parada de emergencia cerca de la zona de trabajo por si el robot pierde el control y no hay otra manera de pararlo que no sea con este botón.
- Revisar regularmente que el botón de parada de emergencia funciona correctamente.
- No esparcir agua o aceite sobre el robot, caja de operación o cable de alimentación, ya que esto puede producir un mal funcionamiento del sistema o descargas eléctricas.
- Tomar medidas de precaución con los objetos que vayan a ponerse en contacto con el robot, tanto con su tamaño, peso, temperatura y composición química.
- Alimentar el sistema solo con la tensión nominal.
- Comprobar regularmente que los tornillos del robot están correctamente apretados.
- Desenchufar el cable de alimentación de la toma de corriente si el robot no va a estar en uso durante largos periodos de tiempo, ya que la acumulación de mucho polvo podría dar lugar a un incendio.
- Asegurarse de que la fuente de alimentación esté apagada a la hora de desenchufar el cable de alimentación.
- Asegurar que la fuente de alimentación está conectada a tierra de forma apropiada antes de usarse. Si no se hiciera, podría dar lugar a un mal funcionamiento, fuego, rotura del sistema o descargas eléctricas.
- No intentar modificar o desmontar el robot.
- No hacer uso del sistema mientras esté presente en el ambiente un gas inflamable o corrosivo, ya que esto podría dar lugar a un incendio o explosión.
- Instalar el robot en un lugar que pueda soportar su peso y donde haya suficiente espacio para su libre movimiento.

- Enchufar el cable de alimentación en la toma de corriente correctamente. El no hacerlo podría causar un incendio.

- Asegurarse de que el enchufe no esté cubierto de polvo.

- Comprobar que la fuente de alimentación está apagada antes de conectar el cable de alimentación.

- Apagar el sistema antes de insertar o quitar cables.

Capítulo 4 : Cinemática directa del robot

El problema de la cinemática directa consiste en utilizar el cálculo matricial para, a partir de los datos conocidos de la posición de cada una de las articulaciones del robot, hallar la posición y orientación de su extremo con respecto a un sistema de referencia fijo, el cual se encuentra en la base de dicho robot. El problema consiste, básicamente, en encontrar una matriz de transformación homogénea **T** que relacione la orientación y posición del extremo del robot con un sistema de referencia fijo. Esta matriz **T** será función de las coordenadas articulares del robot, y se obtendrá a partir del algoritmo de Denavit-Hartenberg (D-H).

4.1. Matriz de transformación homogénea T

Lo primero que hay que aclarar es qué son las coordenadas homogéneas. Pues bien, se dice que un espacio n-dimensional está representado en coordenadas homogéneas por (n+1) dimensiones, de forma que un vector **v**(x, y, z) estará representado por **v**(wx, wy, wz, w), donde w tiene un valor arbitrario y representa un factor de escala. Así, el vector 1**i**+2**j**+3**k** se puede representar en coordenadas homogéneas como [1, 2, 3, 1] ^{T}, o como [2, 4, 6, 2] ^{T}. Un vector nulo se representa como [0, 0, 0, n] ^{T}, siendo n un valor no nulo.

Es a partir del concepto de coordenadas homogéneas de donde sale el concepto de matriz de transformación homogénea. Esta es una matriz 4x4 que representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro. Está formada por las siguientes submatrices:

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{R}_{3x3} & \boldsymbol{p}_{3x1} \\ \boldsymbol{f}_{1x3} & \boldsymbol{w}_{1x1} \end{pmatrix} = \begin{bmatrix} Rotación & Traslación \\ Perspectiva & Escalado \end{bmatrix} \tag{4.1}$$

En robótica, generalmente, los valores de $\mathbf{f}_{1x3}$ se consideran nulos y el de $\mathbf{w}_{1x1}$ la unidad. Por tanto, la matriz **T** quedaría como:

$$\boldsymbol{T} = \begin{pmatrix} \boldsymbol{R}_{3x3} & \boldsymbol{p}_{3x1} \\ 0 & 1 \end{pmatrix} \tag{4.2}$$

De esta forma, si se quiere aplicar una traslación de valor el vector $\mathbf{p}=p_x\mathbf{i}+p_y\mathbf{j}+p_z\mathbf{k}$, **T** corresponderá a una matriz homogénea de traslación:

$$\boldsymbol{T}(\boldsymbol{p}) = \begin{pmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

Por otro lado, si se quiere hacer solo un movimiento de rotación (por ejemplo un ángulo θ), la matriz **T** será de una forma o de otra dependiendo de si gira en torno al eje **x**, al eje **y** o al eje **z**, y **T** será una matriz homogénea de rotación:

$$\boldsymbol{T}(\boldsymbol{x}, \theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta_i & -sen\theta & 0 \\ 0 & sen\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

$$\boldsymbol{T}(\boldsymbol{y},\theta)=\begin{pmatrix} cos\theta & 0 & sen\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sen\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

$$\boldsymbol{T}(\boldsymbol{z},\theta)=\begin{pmatrix} cos\theta & -sen\theta & 0 & 0 \\ sen\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.6}$$

Por último, si se quieren realizar varios movimientos, como uno de rotación sobre el eje **z** un ángulo θ primero y luego uno de también de rotación sobre el eje **y** un ángulo ϕ, para hallar la matriz **T** se tendrán que multiplicar estas dos matrices. El orden de multiplicación dependerá de a qué sistema vengan referidos estos movimientos, si del móvil o del fijo.

Si se consideran que los movimientos son respecto al sistema fijo, se multiplican en orden inverso, y por tanto:

$$\boldsymbol{T}=\boldsymbol{T}(\boldsymbol{y},\phi)\boldsymbol{T}(\boldsymbol{z},\theta) \tag{4.7}$$

Sin embargo, si se consideran que son respecto al sistema móvil, la multiplicación se efectúa en orden directo:

$$\boldsymbol{T}=\boldsymbol{T}(\boldsymbol{z},\theta)\boldsymbol{T}(\boldsymbol{y},\phi) \tag{4.8}$$

Con estos conocimientos, se tendrán las bases suficientes para resolver el algoritmo de Denavit-Hartenberg correspondiente al robot del trabajo.

4.2. Algoritmo de Denavit-Hartenberg

Este algoritmo es un método que acaba siendo intuitivo y simplifica el cálculo del problema cinemático directo para robots con una geometría complicada o con un número alto de articulaciones. Los pasos a seguir para llevar a cabo este algoritmo son los siguientes:

1. Numerar los eslabones, siendo el eslabón 0 la base fija del robot, y los demás eslabones (los móviles) irán numerados de 1 hasta n.

2. Numerar cada articulación, siendo 1 el primer grado de libertad y n el último.

3. Localizar el eje de cada articulación. Si esta es rotativa, este será su propio eje de giro; y si es prismática, será el eje a lo largo del cual se efectúa el desplazamiento.

4. Para i de 0 a n-1, colocar el eje $\mathbf{z}_i$ sobre el eje de la articulación i+1.

5. Situar el origen del sistema de la base $\{S_0\}$ en cualquier punto del eje $\mathbf{z}_0$. Los ejes $\mathbf{x}_0$ e $\mathbf{y}_0$ se colocarán de manera que formen un sistema dextrógiro con $\mathbf{z_0}$.

6. Para i de 1 a n-1, situar el sistema $\{S_i\}$ (solidario al eslabón i) en la intersección del eje $\mathbf{z}_i$ con la línea normal común a $\mathbf{z}_{i-1}$ y $\mathbf{z}_i$. Si ambos ejes se cortasen, se situaría $\{S_i\}$ en el punto de corte. Si fuesen paralelos, se situaría en la articulación i+1.

7. Situar $\mathbf{x}_i$ en la línea normal común a $\mathbf{z}_{i-1}$ y $\mathbf{z}_i$.

8. Situar $\mathbf{y}_i$ de manera que forme un sistema dextrógiro con $\mathbf{x}_i$ y $\mathbf{z}_i$.

9. Situar el sistema $\{S_n\}$ en el extremo del robot, de forma que $\mathbf{z}_n$ coincida con la dirección de $\mathbf{z}_{n-1}$ y $\mathbf{x}_n$ sea normal a $\mathbf{z}_{n-1}$ y $\mathbf{z}_n$.

10. Obtener θ_i como el ángulo que hay que girar en torno a $\mathbf{z}_{i-1}$ para que $\mathbf{x}_{i-1}$ y $\mathbf{x}_i$ queden paralelos.

11. Obtener d_i como la distancia, medida a lo largo del eje $\mathbf{z}_{i-1}$, que habría que desplazar $\{S_{i-1}\}$ para que $\mathbf{x}_i$ y $\mathbf{x}_{i-1}$ quedasen alineados.

12. Obtener a_i como la distancia medida a lo largo del eje $\mathbf{x}_i$ que habría que desplazar el nuevo $\{S_{i-1}\}$ para que su origen coincidiese con $\{S_i\}$.

13. Obtener α_i como el ángulo que habría que girar en torno a $\mathbf{x}_i$ para que el nuevo $\{S_{i-1}\}$ coincidiese totalmente con $\{S_i\}$.

14. Obtener las matrices de transformación ${}^{i-1}\mathbf{A}_i$ de cada articulación.

15. Obtener la matriz de transformación **T** que relaciona el sistema de la base (el sistema de referencia fijo) con el del extremo del robot.

Si se siguen los pasos del 1 hasta el 9, se obtiene lo representado en la figura 4.1:

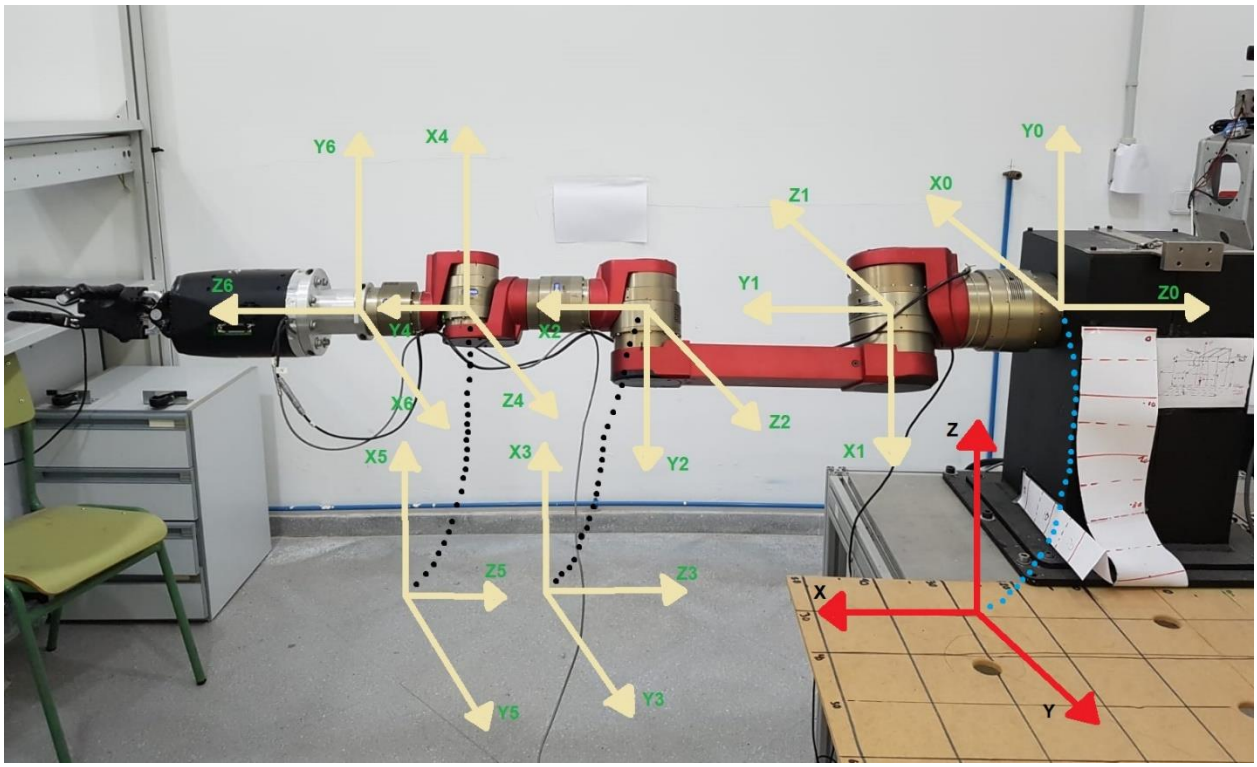

Figura 4.1. Asignación de sistemas de referencia para el brazo modular Robotnik

Donde los ejes de color rojo representan el sistema de referencia externo con los que habrá que relacionar los ejes 0, es decir, el sistema de referencia de la base del robot.

Nota: Hay que imaginar que la articulación 1 está girada unos casi 90º en sentido negativo, ya que esta no funciona y no se ha podido realizar bien la foto.

Partiendo de aquí, aplicando los pasos 10, 11, 12 y 13 se obtienen los parámetros de Denavit-Hartenberg del robot, representados en la tabla 4.1:

Articulación	θ	d	a	α
1	θ_1-90	-l_1	0	-90
2	θ_2+90	0	l_2	180
3	θ_3-90	0	0	90
4	θ_4	-l_3	0	-90
5	θ_5	0	0	90
6	θ_6+90	-l_4	0	180

Tabla 4.1. Parámetros de Denavit-Hartenberg del robot

En esta tabla, θ_i hace referencia a la posición de la articulación en grados, y el valor de cada l_i es el siguiente:

$$l_1 = 241.1\ mm \tag{4.9}$$

$$l_2 = 495\ mm \tag{4.10}$$

$$l_3 = 372\ mm \tag{4.11}$$

$$l_4 = 183.2\ mm \tag{4.12}$$

Para hallar las matrices ${}^{i-1}\mathbf{A}_i$ hay que tener en cuenta los 4 movimientos que se efectúan a la hora de relacionar el sistema de referencia $\{S_{i-1}\}$ con el $\{S_i\}$, los cuales son:

1. Un movimiento de rotación θ_i en torno al eje $\mathbf{z}_{i-1}$.
2. Un movimiento de traslación d_i a lo largo del eje $\mathbf{z}_{i-1}$.
3. Un movimiento de traslación a_i a lo largo del eje $\mathbf{x}_i$.
4. Un movimiento de rotación α_i en torno al eje $\mathbf{x}_i$.

Entonces, sabiendo que estos movimientos están referidos al sistema móvil, se hace el producto de matrices en el orden directo, con lo que se obtiene:

$$ {}^{i-1}\boldsymbol{A}_i = \boldsymbol{T}(\boldsymbol{z},\theta_i)\,\boldsymbol{T}(0,0,d_i)\boldsymbol{T}(a_i,0,0)\,\boldsymbol{T}(\boldsymbol{x},\alpha_i) \tag{4.13}$$

$$ {}^{i-1}\boldsymbol{A}_i = \begin{pmatrix} C\theta_i & S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.14}$$

$$ {}^{i-1}\boldsymbol{A}_i = \begin{pmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.15}$$

Donde

$$ C\theta_i = \cos(\theta_i) \tag{4.16}$$

$$ S\theta_i = \operatorname{sen}(\theta_i) \tag{4.17}$$

De esta matriz, se obtiene que la submatriz que contiene las tres primeras filas y las tres primeras columnas corresponde a la orientación del sistema $\{S_i\}$ con respecto a $\{S_{i-1}\}$, y la submatriz compuesta por las tres primeras filas y la cuarta columna hace referencia a la posición de $\{S_i\}$ con respecto a $\{S_{i-1}\}$. Es decir:

$$ {}^{i-1}\boldsymbol{R}_i = \begin{pmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i \\ 0 & S\alpha_i & C\alpha_i \end{pmatrix} \tag{4.18}$$

$$ {}^{i-1}\boldsymbol{P}_i = \begin{pmatrix} a_i C\theta_i \\ a_i S\theta_i \\ d_i \end{pmatrix} \tag{4.19}$$

En definitiva, la matriz ${}^{i-1}\mathbf{A}_i$ relaciona el sistema de referencia $\{S_{i-1}\}$ con el $\{S_i\}$ en su totalidad, es decir, tanto su orientación como su posición. Si se quisiera relacionar el sistema $\{S_{i-1}\}$ con el $\{S_{i+1}\}$, no habría más que realizar la siguiente multiplicación de matrices:

$$ {}^{i-1}\boldsymbol{A}_{i+1} = {}^{i-1}\boldsymbol{A}_i\,{}^{i}\boldsymbol{A}_{i+1} \tag{4.20}$$

Por tanto, si para el caso del robot que se está estudiando el sistema de referencia del extremo del robot viene dado por $\{S_6\}$, para relacionarlo con el sistema de referencia de la base ($\{S_0\}$) habrá que hacer la siguiente operación:

$$ {}^{0}\boldsymbol{A}_{6} = {}^{0}\boldsymbol{A}_{1}\,{}^{1}\boldsymbol{A}_{2}\,{}^{2}\boldsymbol{A}_{3}\,{}^{3}\boldsymbol{A}_{4}\,{}^{4}\boldsymbol{A}_{5}\,{}^{5}\boldsymbol{A}_{6} \tag{4.21} $$

Sin embargo, hay un problema, y es que el sistema de referencia que se ha puesto en la base no coincide exactamente con el sistema de referencia al que se quiere asociar el extremo del robot, ya que los ejes coordenados de la base están situados de una manera distinta a la que se quiere, como se puede comprobar en la figura 4.1. Esto se soluciona relacionando el sistema de referencia de la base con el sistema de referencia externo, lo que da lugar a la siguiente matriz:

$$ \boldsymbol{A}_{0} = \begin{matrix} \\ \cos(90^0) \\ \cos(180^0) \\ \cos(270^0) \\ 0 \end{matrix} \begin{matrix} \boldsymbol{x_0} \\ \cos(90^0) \\ \cos(270^0) \\ \cos(0^0) \\ 0 \end{matrix} \begin{matrix} \boldsymbol{y_0} \\ \cos(180^0) \\ \cos(270^0) \\ \cos(90^0) \\ 0 \end{matrix} \begin{matrix} \boldsymbol{z_0} \\ 0 \\ 0 \\ 0 \\ 1 \end{matrix} \begin{matrix} \\ \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \\ \end{matrix} = \begin{pmatrix} 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.22} $$

Y las demás matrices, sustituyendo los valores de la tabla 4.1 en (4.15), quedan de la siguiente manera (paso 14 del algoritmo de D-H):

$$ {}^{0}\boldsymbol{A}_{1} = \begin{pmatrix} S1 & 0 & C1 & 0 \\ -C1 & 0 & S1 & 0 \\ 0 & -1 & 0 & -l_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.23} $$

$$ {}^{1}\boldsymbol{A}_{2} = \begin{pmatrix} -S2 & C2 & 0 & -l_2 S2 \\ C2 & S2 & 0 & l_2 C2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.24} $$

$$ {}^{2}\boldsymbol{A}_{3} = \begin{pmatrix} S3 & 0 & -C3 & 0 \\ -C3 & 0 & -S3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.25} $$

$$ {}^{3}\boldsymbol{A}_{4} = \begin{pmatrix} C4 & 0 & -S4 & 0 \\ S4 & 0 & C4 & 0 \\ 0 & -1 & 0 & -l_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.26} $$

$$ {}^{4}\boldsymbol{A}_{5} = \begin{pmatrix} C5 & 0 & S5 & 0 \\ S5 & 0 & -C5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.27} $$

$$ {}^{5}\boldsymbol{A}_{6} = \begin{pmatrix} -S6 & C6 & 0 & 0 \\ C6 & S6 & 0 & 0 \\ 0 & 0 & -1 & -l_4 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.28} $$

Donde

$$ Ci = \cos(\theta_i) \tag{4.29} $$

$$ Si = sen(\theta_i) \tag{4.30} $$

Y, por último, se aplica el paso 15 del algoritmo de D-H teniendo en cuenta la matriz $\mathbf{A}_0$:

$$\boldsymbol{T} = \boldsymbol{A}_0{}^0\boldsymbol{A}_1{}^1\boldsymbol{A}_2{}^2\boldsymbol{A}_3{}^3\boldsymbol{A}_4{}^4\boldsymbol{A}_5{}^5\boldsymbol{A}_6 \tag{4.31}$$

Y el resultado es la matriz que relaciona el extremo del robot con la base del robot. Esta matriz, cuyo resultado viene dado en función de las coordenadas articulares, queda finalmente de la siguiente manera:

Primera fila:

[C6*S4*S(2-3) - S6*(S5*C(2-3) - C4*C5*S(2-3)),

C6*(S5*C(2-3) - C4*C5*S(2-3)) + S4*S6*S(2-3),

C5*C(2-3) + C4*S5*S(2-3), (4.32)

495*C2 + 372*C(2-3) + (916*C5*C(2-3))/5 - (458*S(2-3)*S(4-5))/5 + (458*S(4+5)*S(2-3))/5 + 2411/10]

Segunda fila:

[S6*(S5*S1*S(3-2) - C5*(C4*S1*C(3-2) + C1*S4)) - C6*(S4*S1*C(3-2) - C1*C4),

- C6*(S5*S1*S(3-2) - C5*(C4*S1*C(3-2) + C1*S4)) - S6*(S4*S1*C(3-2) - C1*C4),

- C5*S1*S(3-2) - S5*(C4*S1*C(3-2) + C1*S4), (4.33)

495*S1*S2 - (916*C5*S1*S(3-2))/5 - (916*S5*(C4*S1*C(3-2) + C1*S4))/5 - 372*S1*S(3-2)]

Tercera fila:

[S6*(S5*C1*S(3-2) + C5*(S1*S4 - C4*C1*C(3-2))) - C6*(S4*C1*C(3-2) + C4*S1),

- C6*(S5*C1*S(3-2) + C5*(S1*S4 - C4*C1*C(3-2))) - S6*(S4*C1*C(3-2) + C4*S1),

S5*(S1*S4 - C4*C1*C(3-2)) - C5*C1*S(3-2), (4.34)

(916*S5*(S1*S4 - C4*C1*C(3-2)))/5 + 495*C1*S2 - (916*C5*C1*S(3-2))/5 - 372*C1*S(3-2)]

Cuarta fila

[0, 0, 0, 1] (4.35)

La cual, como se puede comprobar, es una matriz con componentes muy grandes y resultaría muy exhaustivo hallar su valor a mano, ya que, como se ha dicho, depende del valor de todas las coordenadas articulares y se tiene que trabajar en función de ellas. Esta se ha obtenido con ayuda de la herramienta de cálculo simbólico de Matlab.

Conociendo el valor de cada coordenada articular, se podrá obtener el valor de la matriz **T** en su totalidad y, por tanto, la posición y orientación del extremo del robot con respecto a su base.

Capítulo 5 : Cinemática inversa del robot

El problema de la cinemática inversa consiste en lo contrario que el problema de la cinemática directa (Figura 5.1): conocido el valor de la posición y orientación del extremo del robot con respecto a su base, se debe hallar el valor de la posición de cada una de sus articulaciones. Se utiliza para saber qué valores deben adoptar las coordenadas articulares para que el extremo del robot se encuentre en una determinada posición y orientación.

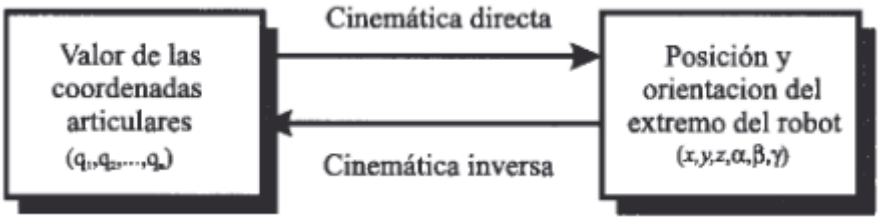

Figura 5.1. Cinemática directa y cinemática inversa [3]

Las ecuaciones del problema cinemático inverso dependen fuertemente de la configuración que tenga el robot. Cada valor de las coordenadas articulares será función de la posición y de la orientación del extremo del robot.

Este problema tiene la peculiaridad de que la solución no es única, ya que distintos valores de la posición de cada articulación permiten colocar el extremo del robot de la misma manera. Para solucionar esto, se suelen poner una serie de límites o restricciones para que la solución obtenida sea la más adecuada.

En la resolución de este problema, se parte de la ventaja de que, normalmente, los robots están hecho de manera que los 3 primeros grados de libertad son los de posicionamiento y tienen una estructura planar, lo que quiere decir que los tres primeros elementos quedan contenidos en un plano. Por otro lado, los 3 últimos grados de libertad suelen estar dispuestos de manera que se dedican solo a orientar el robot y corresponden a giros sobre ejes que se cortan en un punto, que suele ser lo que informalmente se llama “la muñeca del robot”. Todo esto hace que el problema cinemático inverso no sea tan complejo de resolver como podría llegar a serlo y, por tanto, se pueda resolver de manera sistemática.

Para hallar las ecuaciones del problema cinemático inverso, se pueden recurrir a varias opciones:

1. **Métodos geométricos.** Se suelen utilizar para hallar el valor de la posición de las primeras articulaciones, las que posicionan al robot. Para resolverlo se hace uso de relaciones trigonométricas y geométricas sobre los elementos del robot.

2. **Uso de la matriz homogénea**. Se puede hacer uso de la matriz a la que se llega a partir del problema cinemático directo, ya que esta está constituida por 12 componentes que dependen de las coordenadas articulares (9 componentes de orientación y 3 de posición). Es decir, se tendrían 12 ecuaciones y el mismo número de incógnitas que grados de libertad tenga el robot. Sin embargo, a veces

el problema de combinar estas ecuaciones llega a ser muy complejo, y llega a ser muy poco conveniente utilizar este método.

3. **Desacoplo cinemático.** Este método se puede utilizar solo para robots con 6 grados de libertad. Consiste en separar el problema de posicionamiento del problema de orientación, lo que simplifica el problema de manera considerable.

El método que se usará en este trabajo será el desacoplo cinemático, puesto que el robot tiene 6 grados de libertad y, con este método, el problema es simplificado de manera notable.

5.1. Desacoplo cinemático

Como se ha dicho, en el desacoplo cinemático se separan el problema de posicionamiento del extremo del robot y el problema de su orientación: primero se hallarán los valores de la posición de las articulaciones cuyo objetivo sea posicionar al extremo robot (la de las 3 primeras) de manera que no dependan de las articulaciones que se dediquen a orientarlo (las 3 últimas); y, tras hallar estos resultados, se podrán hallar los restantes sin alta complejidad.

Todas estas incógnitas, los valores de las coordenadas articulares, se podrán hallar teniendo en cuenta que los datos de los que se parten son los de la matriz de transformación homogénea **T** (hallada en el capítulo 4 en función de las coordenadas articulares), la cual da lugar a la posición y orientación final del extremo del robot.

5.1.1. Cálculo de las tres primeras coordenadas articulares

Para hallar el valor de las tres primeras coordenadas articulares, se partirá de la matriz de transformación homogénea que posiciona a la muñeca del robot, es decir, de la matriz ${}^0\mathbf{A}_4$, la cual viene dada por:

$$ {}^0\boldsymbol{A}_4 = {}^0\boldsymbol{A}_1\,{}^1\boldsymbol{A}_2\,{}^2\boldsymbol{A}_3\,{}^3\boldsymbol{A}_4 \tag{5.1} $$

La cual, si se calcula a partir de los valores de ${}^{i-1}A_i$ dados en el capítulo 4, teniendo en cuenta que la posición de la muñeca (la cuarta columna de la matriz) solo será función de las tres primeras incógnitas puesto que son las que la posicionan, da lugar al siguiente resultado:

$$ {}^0\boldsymbol{A}_4 = \begin{pmatrix} & {}^0\boldsymbol{R}_4 & & \begin{matrix} l_2C2 + l_2C(2-3) + l_1 \\ S1 \cdot (l_2S2 + l_3S(2-3)) \\ C1 \cdot (l_2S2 + l_3S(2-3)) \end{matrix} & \begin{matrix} Pm_x \\ Pm_y \\ Pm_z \end{matrix} \\ 0 & 0 & 0 & 1 & \end{pmatrix} \tag{5.2} $$

Siendo Pm_x, Pm_y y Pm_z la posición de la muñeca del robot con respecto al eje **x**, al eje **y** y al eje **z,** respectivamente, y

$$ C(i-j) = cos(\theta_i - \theta_j) \tag{5.3} $$

$$ S(i-j) = sen(\theta_i - \theta_j) \tag{5.4} $$

Además, se tiene que el valor de su posición es dato, ya que, sabiendo que la muñeca del robot y su extremo están separados una distancia l_4 = 183.2 mm y conociendo el valor de la matriz homogénea **T**, se obtiene el valor de **Pm** con la siguiente relación:

$$\boldsymbol{Pm} = \boldsymbol{T}\begin{pmatrix} 0 \\ 0 \\ -l_4 \\ 1 \end{pmatrix} \tag{5.5}$$

A partir de esto, se puede hallar el valor de las tres primeras coordenadas articulares. Usando (5.2), el valor de θ_1 se obtiene directamente:

$$\theta_1 = atg\left(\frac{Pm_y}{Pm_z}\right) \tag{5.6}$$

Cuando Pm_y y Pm_z tengan el valor de 0, habrá infinitas soluciones posibles, es decir, habrá una singularidad. Por tanto, habrá que evitar esta posición a la hora de manejar al robot con ayuda de la cinemática inversa (lo cual se explicará en el capítulo 6), y mandar un mensaje de error cuando esa posición esté cerca.

A partir de esto, hay que diferenciar 2 posibles casos: cuando θ_1 tenga el valor de 0 o π, y cuando no lo tenga, ya que se tendrá que desarrollar una ecuación en la que aparece como divisor el seno o el coseno de θ_1.

1º. Si $\theta_1 \neq 0$ o $\pm\pi$

También de (5.2) se obtienen las siguientes dos ecuaciones:

$$\begin{cases} Pm_y = S1 \cdot (l_2 S2 + l_3 S(2-3)) \\ Pm_x = l_2 C2 + l_3 C(2-3) + l_1 \end{cases} \tag{5.7}$$

Ordenándolas un poco:

$$\begin{cases} \dfrac{Pm_y}{S1} = l_2 S2 + l_3 S(2-3) \\ Pm_x - l_1 = l_2 C2 + l_3 C(2-3) \end{cases} \tag{5.8}$$

Si se elevan ambas al cuadrado:

$$\begin{cases} \left(\dfrac{Pm_y}{S1}\right)^2 = (l_2 S2 + l_3 S(2-3))^2 \\ (Pm_x - l_1)^2 = (l_2 C2 + l_3 C(2-3))^2 \end{cases} \tag{5.9}$$

$$\begin{cases} \left(\dfrac{Pm_y}{S1}\right)^2 = (l_2 S2)^2 + (l_3 S(2-3))^2 + 2 l_2 l_3 S2 S(2-3) \\ (Pm_x - l_1)^2 = (l_2 C2)^2 + (l_3 C(2-3))^2 + 2 l_2 l_3 C2 C(2-3) \end{cases} \tag{5.10}$$

Y si se suman ambas ecuaciones y se aplican algunas propiedades trigonométricas, se tiene:

$$\left(\frac{Pm_y}{S1}\right)^2 + (Pm_x - l_1)^2 = {l_2}^2 + {l_3}^2 + 2 l_2 l_3 \cdot (S2S(2-3) + C2C(2-3)) \tag{5.11}$$

$$\left(\frac{Pm_y}{S1}\right)^2 + (Pm_x - l_1)^2 = {l_2}^2 + {l_3}^2 + 2l_2l_3\mathrm{C}[2-(2-3)] \tag{5.12}$$

A partir de la cual se puede hallar el valor de la tercera coordenada articular:

$$\theta_3 = \mathrm{acos}\left(\frac{\left(\frac{Pm_y}{S1}\right)^2 + (Pm_x - l_1)^2 - {l_2}^2 - {l_3}^2}{2l_2l_3}\right) \tag{5.13}$$

Por último, faltaría hallar el valor de la segunda coordenada articular: Para ello, primero se desarrolla un poco la primera ecuación de (5.8):

$$\frac{Pm_y}{S1} = l_2S2 + l_3[S2C3 - C2S3] \tag{5.14}$$

$$S2 = \frac{\frac{Pm_y}{S1} + l_3C2S3}{l_2 + l_3C3} \tag{5.15}$$

Si además se desarrolla la segunda ecuación de (5.8), se tiene:

$$Pm_x - l_1 = l_2C2 + l_3[C2C3 - S2S3] \tag{5.16}$$

Y sustituyendo (5.15) en (5.16):

$$Pm_x - l_1 = l_2C2 + l_3\left[C2C3 - \left(\frac{\frac{Pm_y}{S1} + l_3C2S3}{l_2 + l_3C3}\right)S3\right] \tag{5.17}$$

Despejando de esta ecuación, se puede hallar que:

$$C2 = \frac{(l_2 + l_3C3)(Pm_x - l_1) - \frac{Pm_y}{S1}S3l_3}{l_2(l_2 + l_3C3) + l_3C3(l_2 + l_3C3) + {l_3}^2(S3)^2} \tag{5.18}$$

Que, si se simplifica un poco:

$$C2 = \frac{(l_2 + l_3C3)(Pm_x - l_1) - \frac{Pm_y}{S1}S3l_3}{l_2^2 + 2l_2l_3C3 + l_3^2} \tag{5.19}$$

Y, a partir de aquí, se tiene que θ_2 puede hallarse a partir de (5.15) y (5.19):

$$\theta_2 = atg\left(\frac{S2}{C2}\right) \tag{5.20}$$

Por tanto, la solución de las tres primeras coordenadas articulares estaría compuesta por (5.6), (5.13) y (5.20).

Hay que tener en cuenta que, en (5.13), cuando el numerador sea mayor que el denominador no podrá existir solución, y el problema trataría de obtener una posición físicamente imposible de alcanzar para el robot. Por tanto, cuando el cálculo de la

posición de la articulación θ_3 de lugar a un número no real (hecho que sucede cuando se calcula el arco coseno de un número mayor que 1), aparecerá un mensaje de error diciendo que no se puede avanzar más.

2º. Si $\theta_1 = 0$ o $\pm\pi$

En este caso, se parte de las siguientes ecuaciones de (5.2):

$$\begin{cases} Pm_z = C1 \cdot (l_2 S2 + l_3 S(2-3)) \\ Pm_x = l_2 C2 + l_3 C(2-3) + l_1 \end{cases} \tag{5.21}$$

Que, haciendo un desarrollo semejante al del primer caso, dan lugar al valor de las coordenadas articulares θ_2 y θ_3:

$$\theta_3 = \text{acos}\left(\frac{\left(\frac{Pm_z}{C1}\right)^2 + (Pm_x - l_1)^2 - {l_2}^2 - {l_3}^2}{2 l_2 l_3} \right) \tag{5.22}$$

$$S2 = \frac{\frac{Pm_z}{C1} + l_3 C2 S3}{l_2 + l_3 C3} \tag{5.23}$$

$$C2 = \frac{(l_2 + l_3 C3)(Pm_x - l_1) - \frac{Pm_z}{C1} S3 l_3}{l_2^2 + 2 l_2 l_3 C3 + l_3^2} \tag{5.24}$$

$$\theta_2 = atg\left(\frac{S2}{C2}\right) \tag{5.25}$$

Siendo entonces la solución en este caso la compuesta por (5.6), (5.22) y (5.25).

5.1.2. Cálculo de las tres últimas coordenadas articulares

Para hallar las coordenadas articulares restantes, se parte de la siguiente relación:

$$^0\boldsymbol{R}_6 = {}^0\boldsymbol{R}_3 \, {}^3\boldsymbol{R}_6 \tag{5.26}$$

De la cual, se tienen como dato 2 de las 3 matrices de rotación: $^0\mathbf{R}_6$, la cual se obtiene directamente de la matriz de transformación homogénea **T** (estando esta submatriz compuesta por sus tres primeras filas y tres primeras columnas); y $^0\mathbf{R}_3$, que se obtiene a partir de las tres primeras coordenadas articulares, halladas con las ecuaciones obtenidas en el apartado 5.1.1. La matriz $^0\mathbf{R}_3$ se obtiene más concretamente haciendo la operación

$$^0\boldsymbol{A}_3 = {}^0\boldsymbol{A}_1 \, {}^1\boldsymbol{A}_2 \, {}^2\boldsymbol{A}_3 \tag{5.27}$$

Y, posteriormente, se obtiene $^0\mathbf{R}_3$ de la misma forma que $^0\mathbf{R}_6$.

Por tanto, la única matriz incógnita es $^3\mathbf{R}_6$, por lo que es la que se tiene que quedar despejada:

$$({}^0\boldsymbol{R}_3)^{-1}\,{}^0\boldsymbol{R}_6 = {}^3\boldsymbol{R}_6 \tag{5.28}$$

Además, la matriz de rotación es una matriz ortonormal, lo que quiere decir que su inversa es igual a su traspuesta:

$$({}^0\boldsymbol{R}_3)^{-1} = ({}^0\boldsymbol{R}_3)^T \tag{5.29}$$

Por tanto, la ecuación final quedaría:

$${}^3\boldsymbol{R}_6 = ({}^0\boldsymbol{R}_3)^T\,{}^0\boldsymbol{R}_6 \tag{5.30}$$

Entonces, si se hallan las componentes de ${}^3\mathbf{R}_6$ de la misma manera que en las otras dos matrices y en función de las 3 últimas coordenadas articulares, se tiene que:

$${}^3\boldsymbol{R}_6 = \begin{pmatrix} -C6S4 - C4C5S6 & C4C5C6 - S4S6 & -C4S5 \\ C4C6 - C5S4S6 & C4S6 + C5C6S4 & -S4S5 \\ S5S6 & -C6S5 & -C5 \end{pmatrix} \tag{5.31}$$

Si se llaman r_{ij} a las componentes conocidas de $({}^0\mathbf{R}_3)^T\,{}^0\mathbf{R}_6$, se puede deducir que:

$$r_{13} = -C4S5 \tag{5.32}$$

$$r_{23} = -S4S5 \tag{5.33}$$

$$r_{31} = S5S6 \tag{5.34}$$

$$r_{32} = -C6S5 \tag{5.35}$$

$$r_{33} = -C5 \tag{5.36}$$

Y a partir de esto se puede hallar el valor de cada una de las posiciones de las articulaciones restantes:

$$\theta_4 = \operatorname{atan}\left(\frac{r_{23}}{r_{13}}\right) \tag{5.37}$$

$$\theta_5 = \operatorname{acos}(-r_{33}) \tag{5.38}$$

$$\theta_6 = \operatorname{atan}\left(-\frac{r_{31}}{r_{32}}\right) \tag{5.39}$$

Estos resultados, junto con los obtenidos en el apartado 5.1.1, constituyen la solución completa del problema cinemático inverso del robot.

Hay que aclarar que sí que es verdad que la variación de las tres últimas coordenadas articulares varían la posición del extremo del robot con respecto a cómo lo posicionan las tres primeras articulaciones, pero el verdadero objetivo de estas es poder orientar la herramienta del robot libremente en el espacio.

5.2. Diferentes configuraciones del robot

Como se ha visto en el apartado 5.1, el comienzo de la resolución de la cinemática inversa parte de la obtención del valor de la coordenada articular θ_1, la cual podría dar lugar a un resultado positivo o negativo, dependiendo de las coordenadas de la muñeca del robot. Esto da lugar a dos posibles configuraciones: hombro izquierdo y hombro derecho, las cuales se pueden ver en la figura 5.2:

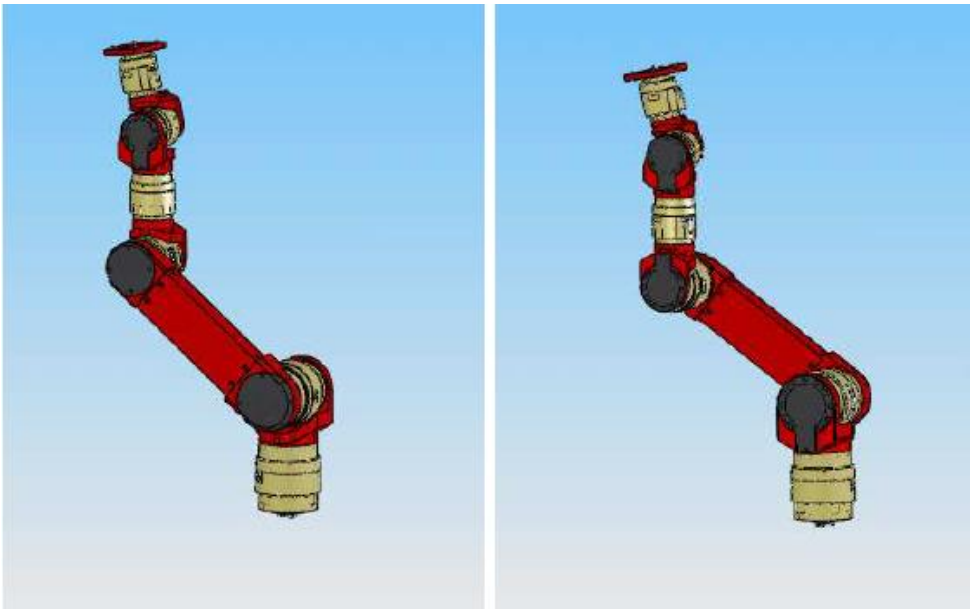

Figura 5.2. Configuraciones de hombro izquierdo y hombro derecho [4]

Tras θ_1, la siguiente coordenada articular a hallar es θ_3. Esta depende de θ_1 y, además, puede ser interpretada de manera que su resultado sea positivo o negativo, lo que da lugar a dos nuevas configuraciones: codo arriba y codo abajo, que se muestran en la figura 5.3:

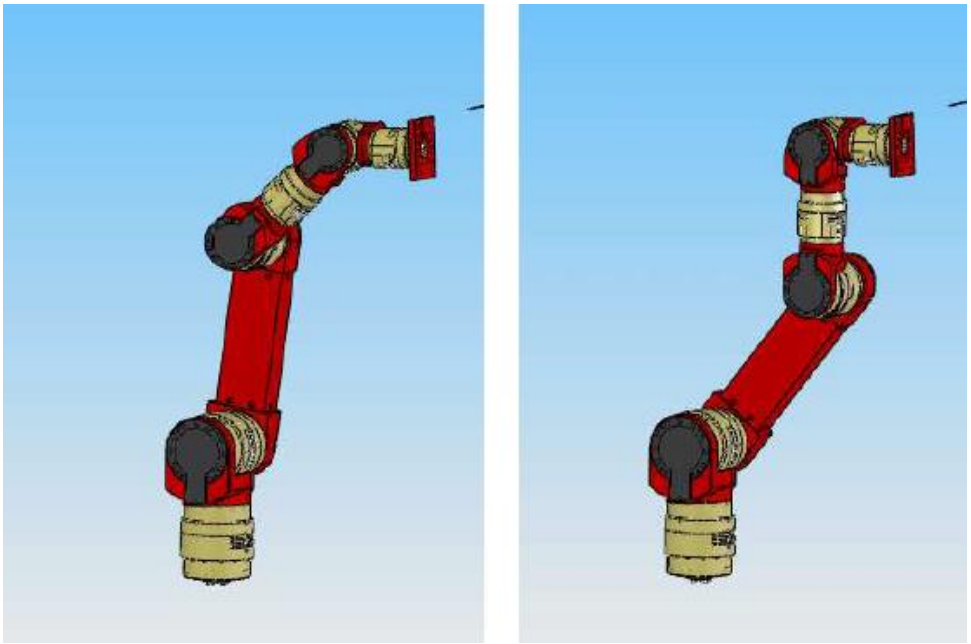

Figura 5.3. Configuraciones de codo arriba y codo abajo [4]

El signo de θ_3 influye directamente en el numerador de θ_2, ya que depende del seno de este (también depende de su coseno, pero esto no influye, ya que el valor va a ser el mismo, independientemente del signo). Por tanto, la posición de θ_3 afecta directamente a la posición de θ_2, como se puede ver en la figura anterior.

Trabajar con el brazo totalmente extendido no es recomendable, ya que esto puede dar lugar a un comportamiento inestable, por lo que hay que evitar llegar a esta posición mostrando un mensaje de error cuando esta se vaya a alcanzar (esta posición corresponde a $\theta_3 = 0º$).

Tras esto, se hallan las tres últimas coordenadas articulares. De θ_5 se podría coger la solución negativa o la solución positiva, de lo cual nacen 2 nuevas configuraciones: muñeca girada y muñeca no girada, que se pueden observar en la figura 5.4:

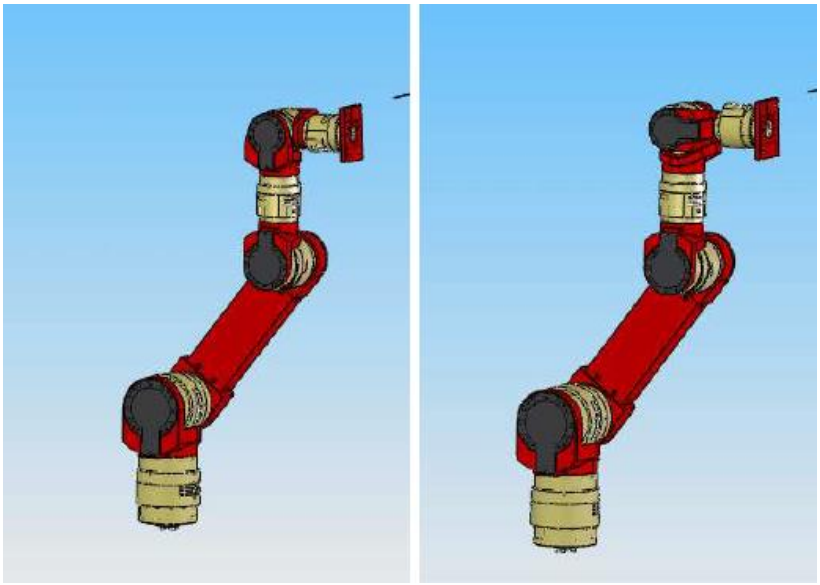

Figura 5.4. Configuraciones de muñeca girada y muñeca no girada [4]

Cuando $\theta_5 = 0º$ las soluciones posibles son infinitas, ya que las articulaciones 4 y 6 podrían combinarse de muchas maneras distintas para poder llegar a esa posición, por tanto, hay una singularidad. Por tanto, se evitará llegar a esta posición a la hora de realizar trayectorias con el robot, ya que si no se produciría un comportamiento inestable del robot. Lo que se suele hacer en estos casos es dejar la "muñeca rota", es decir, dejar la articulación 5 movida al menos unos 5º, lo que da una mayor seguridad al correcto funcionamiento del robot.

Resumiendo, existen 3 tipos de configuraciones diferentes: hombro derecho/izquierdo, codo arroba/abajo, y muñeca girada/no girada. Esto quiere decir que existen $2^3 = 8$ combinaciones distintas para llegar a una misma posición y orientación del extremo del robot. Por tanto, se deberá elegir qué configuración se debe seguir en cada momento, lo cual será sencillo: cuando la coordenada articular en el momento de moverla tenga signo positivo, esta mantendrá su signo positivo; y cuando tenga signo negativo mantendrá el signo negativo, cambiando de signo cuando pase por 0º (si no está restringido). Cuando una de las coordenadas articulares se acerque a una posición a la que haya que evitar llegar (las cuales están especificadas anteriormente) aparecerá un mensaje de error diciendo que no se puede continuar con el movimiento y el robot parará de moverse.

Capítulo 6 : Generación de trayectorias rectilíneas

En este capítulo, se hablará de los movimientos utilizados a la hora de manejar del robot y de cómo se ha conseguido que este haga trayectorias rectilíneas con su muñeca.

6.1. Tipos de movimientos utilizados

Como se adelantó en el capítulo 2, el movimiento de cada una de las articulaciones se ha realizado a partir de 2 funciones distintas de la librería m5apiw32, las cuales se recuerda que son:

```
calllib('m5apiw32','PCube_moveRamp', devID, modID, pos, vel, acel)
```

```
calllib('m5apiw32','PCube_moveVel', devID, modID, vel)
```

La primera de ellas produce un movimiento en rampa, es decir, la articulación acelera con la aceleración especificada en rad/s^2 hasta llegar a su velocidad objetivo (dada en rad/s), que permanece constante hasta que la articulación esté cerca de la posición objetivo (dada en radianes), que es cuando comienza a decelerar con la misma aceleración especificada, pero esta vez con signo contrario, hasta llegar a esa posición. Esta función se ha utilizado para el movimiento individual de cada articulación. Si se analiza esta función en una de sus articulaciones, se obtiene el perfil de velocidad de la figura 6.1, que corresponde a una interpolación a tramos:

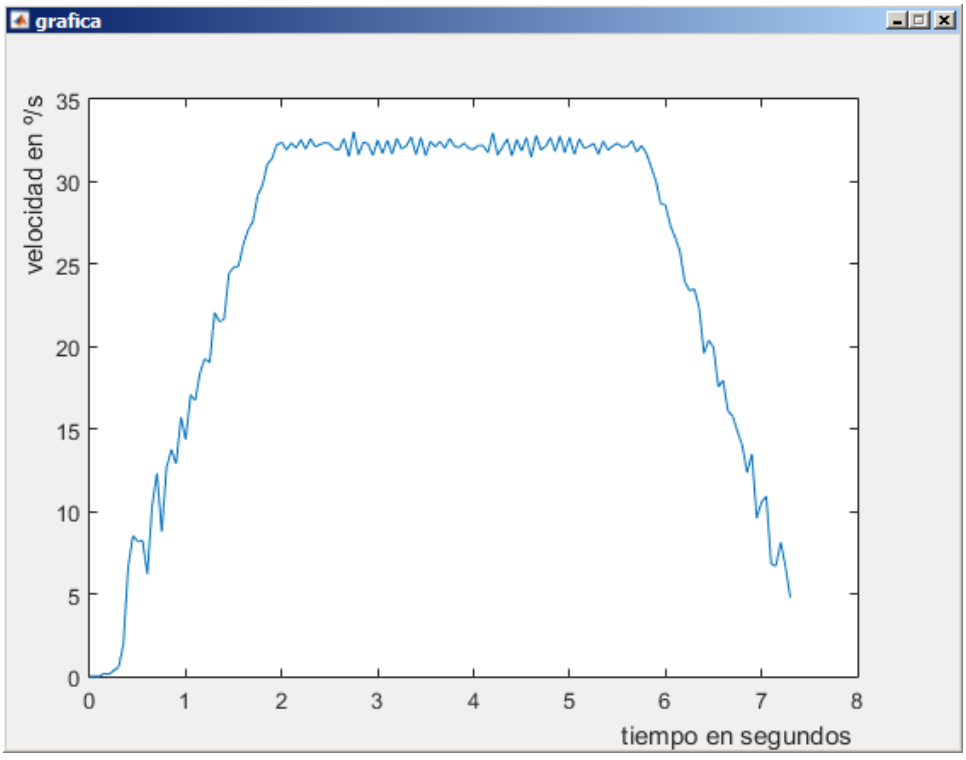

Figura 6.1. Movimiento en rampa de una articulación

Por otro lado, la segunda de las anteriores dos funciones produce que la articulación se mueva a una velocidad constante desde el principio hasta el final. Esta función se utiliza a la hora de manejar el extremo del robot, es decir, cuando se quieran manejar varias articulaciones a la vez con el objetivo de desarrollar trayectorias rectilíneas en la muñeca

del robot y dejar su extremo orientado de la misma manera. Si se analiza esta función en una de sus articulaciones, se obtiene la gráfica de la figura 6.2, que corresponde a una interpolación lineal:

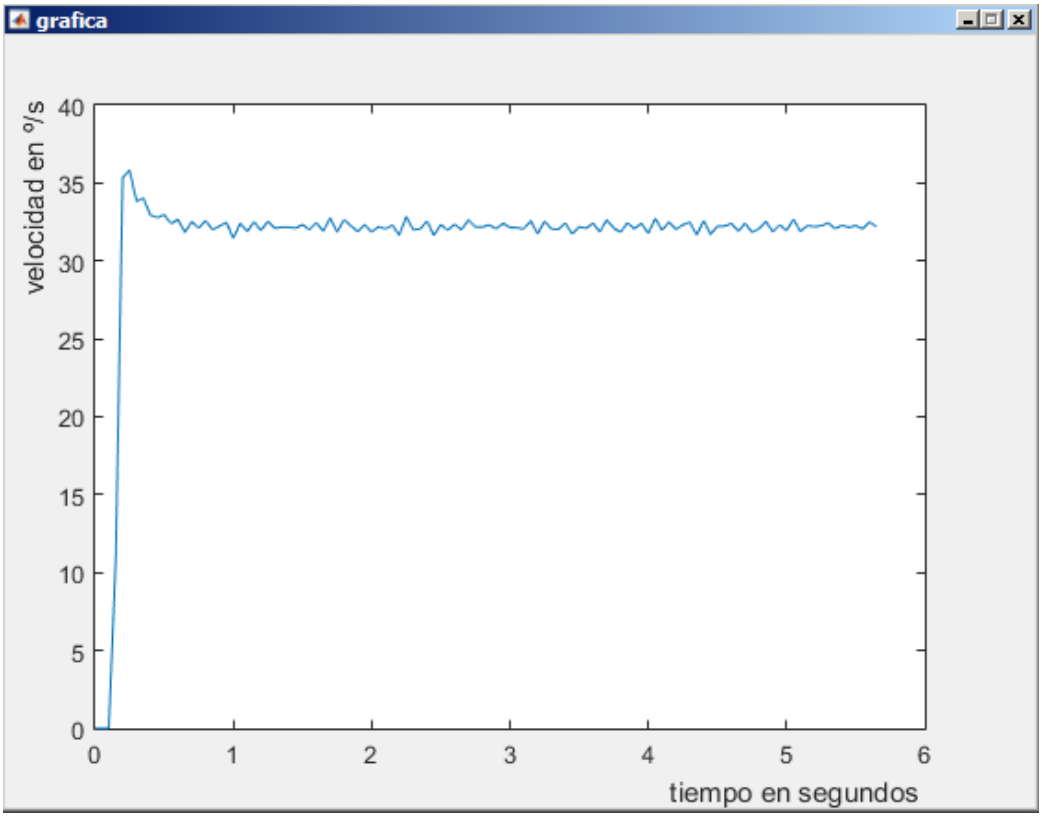

Figura 6.2. Velocidad constante de una articulación

6.2. Obtención de trayectorias rectilíneas

Para la obtención de una trayectoria rectilínea de la muñeca del robot, se hace uso de la solución al problema de la cinemática inversa del robot, obtenida en el capítulo 5. Lo que se hace es, por ejemplo, si se quiere que la muñeca recorra en línea recta el eje coordenado **y**, es decir, manteniendo constante la posición del eje **x** y **z** en la que se encuentra, sumar en bucle 1 mm a la posición Pm_y, sin variar Pm_x ni Pm_z. De esta manera, se van actualizando las nuevas posiciones a las que cada una de las tres primeras coordenadas articulares tiene que llegar para poder mantener esa trayectoria rectilínea, puesto que el valor de estas tres depende de la posición de la muñeca del robot, ya que son las que posicionan a esta, como se puede comprobar en las ecuaciones halladas en el subapartado 5.1.1.

Además, también se actualizarán las 3 últimas coordenadas articulares para que el extremo del robot quede orientado siempre de la misma manera, ya que, de la ecuación utilizada para hallar el valor de estas tres últimas coordenadas articulares

$$^{3}\boldsymbol{R}_{6} = (^{0}\boldsymbol{R}_{3})^{T}\,^{0}\boldsymbol{R}_{6} \tag{6.1}$$

Se tiene que la matriz $^{0}\mathbf{R}_{6}$ siempre será la misma desde que se pulse el botón del joystick hasta que se despulse (ya que se quiere que la orientación permanezca fija desde el

principio hasta el final), y $({}^0\mathbf{R}_3)^T$ será una matriz que dependerá siempre de las tres primeras coordenadas articulares, por lo que esta irá variando constantemente, lo que hace que la matriz ${}^3\mathbf{R}_6$ tenga constantemente nuevos valores en sus componentes y, por tanto, también los tengan las tres últimas coordenadas articulares.

Para mantener la trayectoria rectilínea en la muñeca y una orientación fija en el extremo del robot, las articulaciones deberán cambiar de posición de forma coordinada. Para ello se le aplicará a cada una de ellas una determinada velocidad, la cual dependerá de una fácil ecuación de la física:

$$\theta_i = \theta_{i0} + \frac{1}{2}(\dot{\theta}_i + \dot{\theta}_{i0})t_i \tag{6.2}$$

Donde $\dot{\theta}_i$ es la velocidad angular de la articulación i en rad/s, $\dot{\theta}_{i0}$ es la velocidad angular inicial, t_i es el tiempo en segundos que tarda en alcanzar la posición, θ_i es la posición a alcanzar en radianes y θ_{i0} la posición inicial.

Despejando el tiempo, esta ecuación queda de la forma:

$$t_i = \frac{2(\theta_i - \theta_{i0})}{(\dot{\theta}_i + \dot{\theta}_{i0})} \tag{6.3}$$

Si se quiere que todas las articulaciones tengan que llegar a la posición especificada en el mismo tiempo, entonces:

$$t_1 = t_2 = t_3 = t_4 = t_5 = t_6 \tag{6.4}$$

Si esta igualación se hace con solo dos articulaciones y se desarrolla la ecuación:

$$\frac{2(\theta_1 - \theta_{10})}{(\dot{\theta}_1 + \dot{\theta}_{10})} = \frac{2(\theta_2 - \theta_{20})}{(\dot{\theta}_2 + \dot{\theta}_{20})} \tag{6.5}$$

$$\dot{\theta}_2 = \frac{(\theta_2 - \theta_{20})}{(\theta_1 - \theta_{10})}(\dot{\theta}_1 + \dot{\theta}_{10}) - \dot{\theta}_{20} \tag{6.6}$$

A partir de esta ecuación, si se quiere que la velocidad máxima sea la de la articulación que más camino tenga que recorrer, y la cual corresponde a la velocidad escogida en el pop-menú, la ecuación final será:

$$\dot{\theta}_i = \frac{(\theta_i - \theta_{i0})}{\Delta\theta_{máx}}(\dot{\theta}_{máx} + \dot{\theta}_{máx0}) - \dot{\theta}_{i0} \tag{6.7}$$

Donde $\dot{\theta}_i$ es la velocidad de la articulación i, θ_i la posición a la que tiene que llegar esta, θ_{i0} su posición inicial, $\dot{\theta}_{i0}$ su velocidad inicial, $\Delta\theta_{máx}$ la máxima diferencia de posición entre la objetivo y la inicial de entre las 6 articulaciones, $\dot{\theta}_{máx}$ la velocidad escogida en el pop-up menú, y $\dot{\theta}_{máx0}$ la velocidad inicial de la articulación que tenga que ir más rápido.

Es entonces la ecuación (6.6) la que se utilizará para especificar la velocidad de cada articulación a la hora de manejar el extremo del robot.

Para el movimiento automático del robot (en el cual se recuerda que también se manejan varias articulaciones a la vez), la función que se utiliza para mover a cada articulación es la que produce un movimiento en rampa, ya que es una función más segura para el robot puesto que la velocidad objetivo no la toma de golpe, sino poco a poco, es decir, no necesita un arranque brusco. Sin embargo, aunque no sea del todo exacto, la velocidad que se le da a cada articulación también es la dada por (6.6), ya que no se necesita que todas lleguen al mismo tiempo, sino, simplemente, que pasen por la posición especificada. No obstante, estas sí que aparentan llegar a la vez debido a la poca descoordinación que hay a la hora de llegar cada articulación a la posición objetivo, ya que en el único intervalo de tiempo en el que estas no van del todo coordinadas es en la aceleración y deceleración de la articulación, las cuales no duran mucho. Hay que tener en cuenta que la velocidad inicial de cada articulación en este caso es nula.

Capítulo 7 : Matriz Jacobiana del robot

En ocasiones es interesante saber qué velocidad se necesita que tenga cada articulación del robot para que el extremo del robot tenga una velocidad específica, o saber qué velocidad tiene el extremo del robot a partir de la velocidad de cada coordenada articular, siendo estas obviamente conocidas. Esto se consigue con ayuda de la matriz Jacobiana inversa y con la matriz Jacobiana directa, respectivamente, y se puede ver resumido en la figura 7.1 para un robot de n grados de libertad:

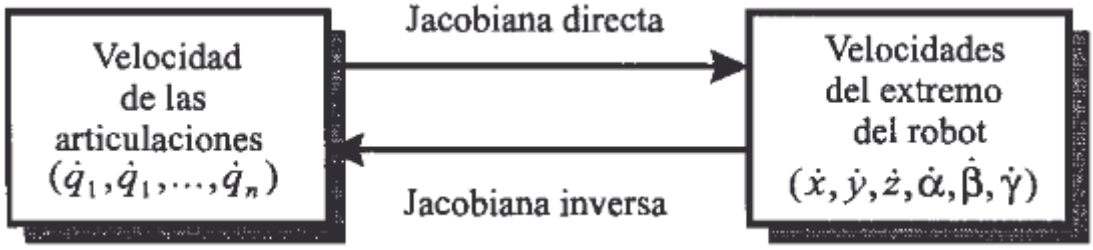

Figura 7.1. Jacobiana directa y Jacobiana inversa [3]

Para el caso de la interfaz del robot solo se implantará la Jacobiana directa, ya que interesa saber cuál es la velocidad del extremo del robot dependiendo de la velocidad de sus articulaciones.

7.1. Matriz Jacobiana directa

La matriz Jacobiana directa (**J**) sirve para hallar la velocidad del extremo del robot siendo conocida la velocidad de cada una de sus articulaciones.

Esta matriz, a su vez, se divide en 2 submatrices: la Jacobiana de velocidad lineal ($\mathbf{J_v}$), y la de velocidad angular ($\mathbf{J_w}$). Todo esto se representa de la siguiente manera para un robot de n grados de libertad:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \boldsymbol{J} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \vdots \\ \vdots \\ \dot{q}_n \end{pmatrix} \tag{7.1}$$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} \boldsymbol{J_v} \\ \boldsymbol{J_w} \end{pmatrix} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \vdots \\ \vdots \\ \dot{q}_n \end{pmatrix} \tag{7.2}$$

El procedimiento de obtención de la matriz Jacobiana directa se divide entonces en la obtención de estas dos submatrices por separado, las cuales se obtendrán haciendo uso de la solución del problema cinemático directo, desarrollado en el capítulo 4.

Empezando por la Jacobiana de velocidad lineal, si se consideran las componentes de la última fila de la matriz de transformación homogénea **T**, las cuales representan las componentes de la posición del extremo del robot con respecto a su base, se tiene que:

$$x = f_x(q_1, q_2, \dots, q_n) \tag{7.3}$$

$$y = f_y(q_1, q_2, \dots, q_n) \tag{7.4}$$

$$z = f_z(q_1, q_2, \dots, q_n) \tag{7.5}$$

Y a partir de esto, derivando con respecto al tiempo los dos lados de cada ecuación, se tiene que:

$$\dot{x} = \sum_{1}^{n} \frac{\partial f_x}{\partial q_i} \dot{q}_i \tag{7.6}$$

$$\dot{y} = \sum_{1}^{n} \frac{\partial f_y}{\partial q_i} \dot{q}_i \tag{7.7}$$

$$\dot{z} = \sum_{1}^{n} \frac{\partial f_z}{\partial q_i} \dot{q}_i \tag{7.8}$$

Lo cual se puede expresar de forma matricial, de manera que:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \boldsymbol{J_v} \begin{pmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \vdots \\ \vdots \\ \dot{q}_n \end{pmatrix} \tag{7.9}$$

Siendo $\mathbf{J_v}$ la submatriz Jacobiana directa de velocidad lineal:

$$\boldsymbol{J_v} = \begin{pmatrix} \frac{\partial f_x}{\partial q_1} & \dots & \frac{\partial f_x}{\partial q_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_z}{\partial q_1} & \dots & \frac{\partial f_z}{\partial q_n} \end{pmatrix} \tag{7.10}$$

Aplicando esto al robot del trabajo, el cual se recuerda que tiene 6 articulaciones rotativas (6 grados de libertad), la submatriz sería de la siguiente forma:

$$\boldsymbol{J_v} = \begin{pmatrix} \frac{\partial f_x}{\partial \theta_1} & \frac{\partial f_x}{\partial \theta_2} & \frac{\partial f_x}{\partial \theta_3} & \frac{\partial f_x}{\partial \theta_4} & \frac{\partial f_x}{\partial \theta_5} & \frac{\partial f_x}{\partial \theta_6} \\ \frac{\partial f_y}{\partial \theta_1} & \frac{\partial f_y}{\partial \theta_2} & \frac{\partial f_y}{\partial \theta_3} & \frac{\partial f_y}{\partial \theta_4} & \frac{\partial f_y}{\partial \theta_5} & \frac{\partial f_y}{\partial \theta_6} \\ \frac{\partial f_z}{\partial \theta_1} & \frac{\partial f_z}{\partial \theta} & \frac{\partial f_z}{\partial \theta_3} & \frac{\partial f_z}{\partial \theta_4} & \frac{\partial f_z}{\partial \theta_5} & \frac{\partial f_z}{\partial \theta} \end{pmatrix} \tag{7.11}$$

Se puede hallar cada componente de la submatriz Jacobiana directa de velocidad lineal con ayuda del cálculo simbólico de Matlab. El resultado de esta queda en función de las coordenadas articulares y es de la siguiente manera:

Primera fila:

[0,

(458*S(4+5)*C(2-3))/5 - 372*S(2-3) - (916*C5*S(2-3))/5 - (458*C(2-3)*S(4-5))/5 - 495*S2,

372*S(2-3) + (916*C5*S(2-3))/5 - (916*C4*S5*C(2-3))/5, (7.12)

-(916*S4*S5*S(2-3))/5,

(916*C4*C5*S(2-3))/5 - (916*S5*C(2-3))/5,

0]

Segunda fila:

[495*C1*S2 - (916*S5*(C4*C1*C(2-3) - S1*S4))/5 - (916*C5*C1*S(3-2))/5 - 372*C1*S(3-2),

(S1*(2475*C2 + 1860*C(2-3) + 916*C5*C(2-3) - 916*C4*S5*S(3-2)))/5,

(916*C4*S5*S1*S(3-2))/5 - 372*S1*C(3-2) - (916*C5*S1*C(3-2))/5, (7.13)

(916*S5*(S4*S1*C(3-2) - C1*C4))/5,

(916*S5*S1*S(3-2))/5 - (916*C5*(C4*S1*C(3-2) + C1*S4))/5,

0]

Tercera fila:

[(916*C5*S1*S(3-2))/5 - 495*S1*S2 + (916*S5*(C4*S1*C(3-2) + C1*S4))/5 + 372*S1*S(3-2),

(C1*(2475*C2 + 1860*C(3-2) + 916*C5*C(3-2) - 916*C4*S5*S(3-2)))/5,

(916*C4*S5*C1*S(3-2))/5 - 372*C1*C(3-2) - (916*C5*C1*C(3-2))/5, (7.14)

(916*S5*(C4*S1+S4*C1*C(3-2)))/5,

(916*S5*C1*S(3-2))/5 - (916*C5*(C4*C1*C(3-2) - S1*S4))/5,

0]

Y, multiplicando los componentes de esta matriz por los valores de velocidad de cada una de las articulaciones como se hace en (7.9), se podrá obtener el valor de las componentes de velocidad lineal del extremo del robot en mm/s con respecto al sistema de referencia establecido en la base del robot.

Lo siguiente es hallar la submatriz Jacobiana de velocidad angular. Para ello se hace uso de las matrices de rotación de cada articulación con respecto al sistema de referencia de

la base del robot. La velocidad angular del extremo del robot y la velocidad de cada articulación se relacionan con la submatriz Jacobiana de velocidad angular de la siguiente manera:

$$\begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\Psi}\end{pmatrix} = {}^{0}\boldsymbol{R}_1\vec{\dot{\boldsymbol{q}}}_1 + {}^{0}\boldsymbol{R}_2\vec{\dot{\boldsymbol{q}}}_2 + \cdots + {}^{0}\boldsymbol{R}_n\vec{\dot{\boldsymbol{q}}}_n = \boldsymbol{J_w}\begin{pmatrix}\dot{q}_1\\ \dot{q}_2\\ \vdots\\ \vdots\\ \vdots\\ \dot{q}_n\end{pmatrix} \tag{7.15}$$

Siendo

$$\vec{\dot{\boldsymbol{q}}}_i = \begin{pmatrix}0\\ 0\\ 1\end{pmatrix}\dot{q}_i \tag{7.16}$$

Ya que el giro se produce en torno al eje **z** de cada articulación.

Además, se recuerda que:

$${}^{0}\boldsymbol{R}_n = \boldsymbol{R}_0\,{}^{0}\boldsymbol{R}_1\,{}^{1}\boldsymbol{R}_2 \ldots {}^{n-1}\boldsymbol{R}_n \tag{7.17}$$

Donde cada $^{i-1}\mathbf{R}_i$ es la submatriz de rotación extraída de su matriz de transformación homogénea $^{i-1}\mathbf{A}_i$ correspondiente (está compuesta por sus tres primeras filas y sus tres primeras columnas). Estas matrices $^{i-1}\mathbf{A}_i$ correspondientes al robot del trabajo están halladas en el capítulo 4.

Si esto se aplica al robot en cuestión, se tiene que 7.15 toma la siguiente forma:

$$\begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\Psi}\end{pmatrix} = {}^{0}\boldsymbol{R}_1\vec{\dot{\boldsymbol{\theta}}}_1 + {}^{0}\boldsymbol{R}_2\vec{\dot{\boldsymbol{\theta}}}_2 + {}^{0}\boldsymbol{R}_3\vec{\dot{\boldsymbol{\theta}}}_3 + {}^{0}\boldsymbol{R}_4\vec{\dot{\boldsymbol{\theta}}}_4 + {}^{0}\boldsymbol{R}_5\vec{\dot{\boldsymbol{\theta}}}_5 + {}^{0}\boldsymbol{R}_6\vec{\dot{\boldsymbol{\theta}}}_6 = \boldsymbol{J_w}\begin{pmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \dot{\theta}_3\\ \dot{\theta}_4\\ \dot{\theta}_5\\ \dot{\theta}_6\end{pmatrix} \tag{7.18}$$

De esta forma, la tercera columna de cada $^{0}\mathbf{R}_i$ representará la columna i de la matriz Jacobiana directa de velocidad angular ($\mathbf{J_w}$). Por tanto, hallar $\mathbf{J_w}$ es fácil si se hace uso de la herramienta de cálculo simbólico de Matlab, y da como resultado lo siguiente:

Primera fila:

[0,

0,

-C(2-3),

S4*S(2-3),

- C5*C(2-3) - C4*S5*S(2-3),

C5*C(2-3) + C4*S5*S(2-3)] (7.19)

Segunda fila:

[-C1,

C1,

-S1*S(2-3),

C1*C4 - S4*S1*C(3-2),

C5*S1*S(3-2) + S5*(C4*S1*C(3-2) + C1*S4),

- C5*S1*S(3-2) - S5*(C4*S1*C(3-2) + C1*S4)] (7.20)

Tercera fila:

[S1,

-S1,

-C1*S(2-3),

- S4*C1*C(3-2) - C4*S1,

C5*C1*S(3-2) + S5*(C4*C1*C(3-2) - S1*S4),

- C5*C1*S(3-2) - S5*(C4*C1*C(3-2) - S1*S4)] (7.21)

Si se aplica entonces (7.18) multiplicando esta matriz obtenida por la velocidad de cada una de las articulaciones del robot, se obtendrán las componentes de la velocidad angular del extremo del robot en rad/s con respecto al sistema de referencia establecido en la base del robot.

Estas tres últimas filas halladas representan las 3 filas de la submatriz Jacobiana directa de velocidad angular y las tres últimas de la matriz Jacobiana directa, siendo sus 3 primeras filas las correspondientes a la matriz Jacobiana directa de velocidad lineal. Es decir, las 6 filas de la matriz Jacobiana directa del robot son, respectivamente (7.12), (7.13), (7.14), (7.19), (7.20) y (7.21).

Tras hallar esta, solo falta conocer la velocidad de cada articulación, que se recuerda que dependerá de la velocidad que se seleccione en el Pop-up menú de la interfaz del robot: cuando se mueva solo una articulación, esta será su velocidad en el momento que se mantenga la velocidad constante (siendo obviamente menor en los tramos de aceleración y deceleración); mientras que si se mueven varias al mismo tiempo, esta será la velocidad que tome la articulación más rápida. La velocidad que tomen las demás articulaciones vendrá dada por la ecuación desarrollara en el capítulo 6, la cual se recuerda que quedaba de la siguiente manera:

$$\dot{\theta}_i = \frac{(\theta_i - \theta_{i0})}{\Delta\theta_{máx}}\left(\dot{\theta}_{máx} + \dot{\theta}_{máx0}\right) - \dot{\theta}_{i0} \quad (7.22)$$

7.2. Matriz Jacobiana inversa

La matriz Jacobiana inversa (**J**$^{-1}$), la cual da como resultado la velocidad que necesita cada articulación para que el extremo del robot tenga una velocidad determinada, se representa de la siguiente manera para el robot del trabajo:

$$\begin{pmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \end{pmatrix} = \boldsymbol{J^{-1}} \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \tag{7.23}$$

Y es, simplemente, la inversa de la Jacobiana directa, hallada en el apartado 7.1. Esta no puede ser hallada con ayuda del cálculo simbólico de Matlab, puesto que la matriz es excesivamente grande y sobrepasa el límite de caracteres que puede escribir Matlab en una línea de resultado (25,000). Por tanto, para hallar esta habría que evaluar numéricamente la matriz Jacobiana directa en una configuración determinada e invertirla, puesto que eso sí que se puede resolver sin problemas.

Capítulo 8 : Conclusiones y futuros trabajos

En este capítulo se hablará de las conclusiones obtenidas al terminar el trabajo, tanto de lo que se ha aprendido como de lo que se ha desarrollado, además de las incidencias que han ocurrido durante el transcurso del proyecto. También se comentarán algunos de los posibles desarrollos futuros que pueden efectuarse a partir de este.

8.1. Conclusiones

Resumiendo, en este trabajo se ha desarrollado una interfaz capaz de permitir a una persona manejar un robot, tanto cualquiera de sus articulaciones por individual, como el extremo de este, es decir, manejando varias articulaciones a la vez. Además, en ella, se puede controlar tanto la posición de estas articulaciones, como la orientación y la posición de su extremo, además de poder controlar la velocidad de este. Por otra parte, también se pueden guardar varias posiciones de las articulaciones para que el robot pase por ellas, en orden, con tan solo pulsar un botón, moviendo también varias articulaciones al mismo tiempo.

Para el desarrollo de esta, se han aplicado conceptos básicos de robótica y se ha trabajado con la función GUI de Matlab, de los cuales no se habla nada en el grado de GITI en la UPCT. Gracias a que se ha trabajado con un robot real, los conceptos de robótica han sido más fáciles de entender y, por tanto, de aprender e implementar. Estos conceptos que se han estudiado a lo largo del proyecto han ayudado a desarrollar los siguientes apartados referentes al ámbito de la robótica: la resolución del problema de la cinemática directa para obtener la posición y orientación del extremo del robot con respecto al sistema de referencia establecido en su base, la resolución del problema de la cinemática inversa para desarrollar trayectorias rectilíneas en la muñeca del robot, y el cálculo de la matriz Jacobiana directa, la cual se utiliza para el cálculo de los componentes de la velocidad, tanto lineal como angular, del extremo del robot en cada uno de sus tres ejes coordenados. También hay que decir que, al tener que buscar información sobre el funcionamiento de Matlab y su función GUI para poder crear la interfaz y hacer que funcione bien, uno aprende a ser más autodidacta a la hora de buscar información. Para el manejo del robot se ha hecho uso de la librería m5apiw32, la cual también hubo que investigar un poco.

Como incidencias, decir que, cuando se estaba intentando implantar la generación de trayectorias rectilíneas, una de las articulaciones paró de moverse (más concretamente la primera), y no ha vuelto a funcionar. Sin embargo, sí que se puede obtener la posición de esta, aunque tanto esta como la segunda articulación tienen una nueva posición inicial (correspondiente a la posición de 0º) que no se corresponde con la que tenían al principio, es decir, se han desconfigurado. Se escribió a la empresa suministradora del robot, pero no ha respondido, por lo que no se ha podido arreglar. Por tanto, no se ha podido hacer el vídeo de demostración del funcionamiento del manejo del extremo del robot como se quería inicialmente.

8.2. Futuros trabajos

Como un buen trabajo futuro, se puede hacer que el robot, aparte de poder moverse, pueda coger objetos con su herramienta, la cual tiene la apariencia de una mano y cuenta con 3 dedos, que son los mínimos necesarios para poder ejercer la función de esta. El robot, de esta manera, podría trasladar objetos de un lado a otro, lo cual puede llegar a resultar interesante y útil. Esto se haría manejando estos 3 dedos, los cuales tendrían que controlarse de forma conjunta. También se podría manejar cada uno de los dedos por separado, todo esto con ayuda de la interfaz gráfica, de la cual se debería hacer una nueva versión.

Capítulo 9 : Anexo: Código del programa

Función para colocar en los botones del joystick los dibujos de las flechas:

```
function []=dibujo(imagen,destino)
%Se guarda la imagen que se quiere colocar en el botón
a=imread(imagen);
%Se establecen en el botón las unidades de píxeles para poder luego
%redimensionar coorectamente la imagen
set(destino,'Units','pixels');
%Se obtiene el vector position del botón para obtener su ancho y su alto
b=get(destino,'Position');
%Se redimensiona la imagen
c=imresize(a,[b(4) b(3)]);
%Se vuelven a establecer en el botón las unidades que tenía especificadas
%previamente
set(destino,'Units','characters');
%Se coloca la imagen en el botón
set(destino,'CData',c);
```

Código para hallar la matriz Jacobiana directa:

```
%Se halla la matriz de transformación homogénea del robot
syms a b c d e f;
A0=[0 0 -1 0; -1 0 0 0; 0 1 0 0; 0 0 0 1];
A1=[sin(a) 0 cos(a) 0; -cos(a) 0 sin(a) 0; 0 -1 0 -241.1; 0 0 0 1];
A2=[-sin(b) cos(b) 0 -495*sin(b); cos(b) sin(b) 0 ...
    495*cos(b); 0 0 -1 0; 0 0 0 1];
A3=[sin(c) 0 -cos(c) 0; -cos(c) 0 -sin(c) 0; 0 1 0 0; 0 0 0 1];
A4=[cos(d) 0 -sin(d) 0; sin(d) 0 cos(d) 0; 0 -1 0 -372; 0 0 0 1];
A5=[cos(e) 0 sin(e) 0; sin(e) 0 -cos(e) 0; 0 1 0 0; 0 0 0 1];
A6=[-sin(f) cos(f) 0 0;cos(f)  sin(f) 0 0; 0 0 -1 -183.2; 0 0 0 1];
T=A0*A1*A2*A3*A4*A5*A6;

%Se halla la matriz Jacobiana directa de velocidad lineal
Jv=[diff(T(1,4),a) diff(T(1,4),b) diff(T(1,4),c) diff(T(1,4),d) ...
    diff(T(1,4),e) diff(T(1,4),f); diff(T(2,4),a) diff(T(2,4),b) ...
    diff(T(2,4),c) diff(T(2,4),d) diff(T(2,4),e) diff(T(2,4),f); ...
    diff(T(3,4),a) diff(T(3,4),b) diff(T(3,4),c) diff(T(3,4),d) ...
    diff(T(3,4),e) diff(T(3,4),f)];

%Se inicializan las matrices de rotación de cada articulación
syms a b c d e f;
R0=A0(1:3,1:3);
R1=A1(1:3,1:3);
R2=A2(1:3,1:3);
R3=A3(1:3,1:3);
R4=A4(1:3,1:3);
R5=A5(1:3,1:3);
R6=A6(1:3,1:3);

%Se halla cada matriz de rotación referida al sistema de referencia de la
%base del robot
R01=R0*R1;
R02=R01*R2;
R03=R02*R3;
R04=R03*R4;
R05=R04*R5;
R06=R05*R6;
```

```
%Se hallan cada una de las columnas de la matriz Jacobiana direta de
%velocidad angular
Jwc1=R01*[0 0 1]';
Jwc2=R02*[0 0 1]';
Jwc3=R03*[0 0 1]';
Jwc4=R04*[0 0 1]';
Jwc5=R05*[0 0 1]';
Jwc6=R06*[0 0 1]';

%Se concatenan todas las columnas para hallar Jw
Jw=[Jwc1 Jwc2 Jwc3 Jwc4 Jwc5 Jwc6];

%Y, por último, se concatenan Jv y Jw para hallar la matriz Jacobiana
%directa
J=[Jv; Jw];
```

Código de la interfaz de “Mostrar posiciones guardadas”:

```
function varargout = PosicionesGuardadas(varargin)
% POSICIONESGUARDADAS MATLAB code for PosicionesGuardadas.fig
%      POSICIONESGUARDADAS, by itself, creates a new POSICIONESGUARDADAS or
raises the existing
%      singleton*.
%
%      H = POSICIONESGUARDADAS returns the handle to a new
POSICIONESGUARDADAS or the handle to
%      the existing singleton*.
%
%      POSICIONESGUARDADAS('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in POSICIONESGUARDADAS.M with the given input
arguments.
%
%      POSICIONESGUARDADAS('Property','Value',...) creates a new
POSICIONESGUARDADAS or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before PosicionesGuardadas_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to PosicionesGuardadas_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 PosicionesGuardadas

% Last Modified by GUIDE v2.5 08-Jul-2018 12:05:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PosicionesGuardadas_OpeningFcn, ...
                   'gui_OutputFcn',  @PosicionesGuardadas_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 PosicionesGuardadas is made visible.
function PosicionesGuardadas_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 PosicionesGuardadas (see VARARGIN)

%Se declaran variables auxiliares para que no se guarde nada en el otro GUI
%si no se le da a guardar, ya que todos los cambios se hacen en las
%variables auxiliares y, cuando se le de a guardar, todo lo de las variables
%auxiliares pasará a las variables buenas
global c d e f g h cont c2 d2 e2 f2 g2 h2 cont2;
cont2=cont;
for i=1:cont
    c2(i)=c(i);
    d2(i)=d(i);
    e2(i)=e(i);
    f2(i)=f(i);
    g2(i)=g(i);
    h2(i)=h(i);
end
%Si se ha guardado al menos una posición, en la listbox se colocará el
%nombre Posición 1 principalmente
if cont>=1
    set(handles.lbxPosGuard,'String', 'Posición 1');
    %Con en este bucle se van contatenando todos las posiciones guardadas
    %hasta llegar a la última
    for i=2:cont
        lista = get(handles.lbxPosGuard,'String');
        %Para que la función char(,) tenga efecto, hay que convertir en
        %tipo char lo que hay dentro de la listbox
        listaenchar=char(lista);
        %Esta función char concatena en vertical varias variables char
        listamasuno = char(listaenchar,char(['Posición ', num2str(i)]));
        %Y, por último, se actualiza lo que aparecerá en la listbox
        set(handles.lbxPosGuard,'String',cellstr(listamasuno));
    end
    %Después, como por defecto se marca el primer elemento de la lista,
    %aparecerán en los textos todos los valores de la matriz homogénea de la
    %primera posición guardada
    %Matriz de transformación homogénea del robot
    A0=[0 0 -1 0; -1 0 0 0; 0 1 0 0; 0 0 0 1];
    A1=[sin(c(1)) 0 cos(c(1)) 0; -cos(c(1)) 0 sin(c(1)) 0; 0 -1 0 -241.1; 0 0
0 1];
    A2=[-sin(d(1)) cos(d(1)) 0 -495*sin(d(1)); cos(d(1)) sin(d(1)) 0 ...
      495*cos(d(1)); 0 0 -1 0; 0 0 0 1];
```

```
    A3=[sin(e(1)) 0 -cos(e(1)) 0; -cos(e(1)) 0 -sin(e(1)) 0; 0 1 0 0; 0 0 0
1];
    A4=[cos(f(1)) 0 -sin(f(1)) 0; sin(f(1)) 0 cos(f(1)) 0; 0 -1 0 -372; 0 0 0
1];
    A5=[cos(g(1)) 0 sin(g(1)) 0; sin(g(1)) 0 -cos(g(1)) 0; 0 1 0 0; 0 0 0 1];
    A6=[-sin(h(1)) cos(h(1)) 0 0;cos(h(1))  sin(h(1)) 0 0; 0 0 -1 -183.2; 0 0
0 1];
    T=A0*A1*A2*A3*A4*A5*A6;

    %Posiciones x y z
    set(handles.txtPosExtRobotXPosGuard,'String',[num2str(round(T(1,4),2)),
'mm']);
    set(handles.txtPosExtRobotYPosGuard,'String',[num2str(round(T(2,4),2)),
'mm']);
    set(handles.txtPosExtRobotZPosGuard,'String',[num2str(round(T(3,4),2)),
'mm']);

    %Matriz de rotación
    set(handles.txtMatRotX6XPosGuard,'String',num2str(round(T(1,1),2)));
    set(handles.txtMatRotX6YPosGuard,'String',num2str(round(T(2,1),2)));
    set(handles.txtMatRotX6ZPosGuard,'String',num2str(round(T(3,1),2)));
    set(handles.txtMatRotY6XPosGuard,'String',num2str(round(T(1,2),2)));
    set(handles.txtMatRotY6YPosGuard,'String',num2str(round(T(2,2),2)));
    set(handles.txtMatRotY6ZPosGuard,'String',num2str(round(T(3,2),2)));
    set(handles.txtMatRotZ6XPosGuard,'String',num2str(round(T(1,3),2)));
    set(handles.txtMatRotZ6YPosGuard,'String',num2str(round(T(2,3),2)));
    set(handles.txtMatRotZ6ZPosGuard,'String',num2str(round(T(3,3),2)));
end

% Choose default command line output for PosicionesGuardadas
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PosicionesGuardadas wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = PosicionesGuardadas_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;


% --- Executes on selection change in lbxPosGuard.
function lbxPosGuard_Callback(hObject, eventdata, handles)
% hObject    handle to lbxPosGuard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns lbxPosGuard
contents as cell array
```

```
%        contents{get(hObject,'Value')} returns selected item from
lbxPosGuard
global c2 d2 e2 f2 g2 h2;
%Se obtiene qué posición guardada se está marcando y se actualizan los
%text box de los paneles para referirse a los valores de la nueva posición
%marcada
i=get(handles.lbxPosGuard,'Value');

%Matriz de transformación homogénea del robot
A0=[0 0 -1 0; -1 0 0 0; 0 1 0 0; 0 0 0 1];
A1=[sin(c2(i)) 0 cos(c2(i)) 0; -cos(c2(i)) 0 sin(c2(i)) 0; 0 -1 0 -241.1; 0 0
0 1];
A2=[-sin(d2(i)) cos(d2(i)) 0 -495*sin(d2(i)); cos(d2(i)) sin(d2(i)) 0 ...
    495*cos(d2(i)); 0 0 -1 0; 0 0 0 1];
A3=[sin(e2(i)) 0 -cos(e2(i)) 0; -cos(e2(i)) 0 -sin(e2(i)) 0; 0 1 0 0; 0 0 0
1];
A4=[cos(f2(i)) 0 -sin(f2(i)) 0; sin(f2(i)) 0 cos(f2(i)) 0; 0 -1 0 -372; 0 0 0
1];
A5=[cos(g2(i)) 0 sin(g2(i)) 0; sin(g2(i)) 0 -cos(g2(i)) 0; 0 1 0 0; 0 0 0 1];
A6=[-sin(h2(i)) cos(h2(i)) 0 0;cos(h2(i))  sin(h2(i)) 0 0; 0 0 -1 -183.2; 0 0
0 1];
T=A0*A1*A2*A3*A4*A5*A6;

%Posiciones x y z
set(handles.txtPosExtRobotXPosGuard,'String',[num2str(round(T(1,4),2)),
'mm']);
set(handles.txtPosExtRobotYPosGuard,'String',[num2str(round(T(2,4),2)),
'mm']);
set(handles.txtPosExtRobotZPosGuard,'String',[num2str(round(T(3,4),2)),
'mm']);

%Matriz de rotación
set(handles.txtMatRotX6XPosGuard,'String',num2str(round(T(1,1),2)));
set(handles.txtMatRotX6YPosGuard,'String',num2str(round(T(2,1),2)));
set(handles.txtMatRotX6ZPosGuard,'String',num2str(round(T(3,1),2)));
set(handles.txtMatRotY6XPosGuard,'String',num2str(round(T(1,2),2)));
set(handles.txtMatRotY6YPosGuard,'String',num2str(round(T(2,2),2)));
set(handles.txtMatRotY6ZPosGuard,'String',num2str(round(T(3,2),2)));
set(handles.txtMatRotZ6XPosGuard,'String',num2str(round(T(1,3),2)));
set(handles.txtMatRotZ6YPosGuard,'String',num2str(round(T(2,3),2)));
set(handles.txtMatRotZ6ZPosGuard,'String',num2str(round(T(3,3),2)));



% --- Executes during object creation, after setting all properties.
function lbxPosGuard_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lbxPosGuard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



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

```
% hObject    handle to btnGuardarPosGuard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global c d e f g h cont c2 d2 e2 f2 g2 h2 cont2;
%Si se guarda, las variables buenas tomarán los valores de las variables
%auxiliares y no se podrá retroceder
for i=1:cont2
    c(i)=c2(i);
    d(i)=d2(i);
    e(i)=e2(i);
    f(i)=f2(i);
    g(i)=g2(i);
    h(i)=h2(i);
end
cont=cont2;


% --- Executes on button press in btnEliminarPosisionPosGuard.
function btnEliminarPosisionPosGuard_Callback(hObject, eventdata, handles)
% hObject    handle to btnEliminarPosisionPosGuard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global c2 d2 e2 f2 g2 h2 cont2;
%Primero se recoge el elemento que se está marcando.
elemento=get(handles.lbxPosGuard,'Value');

%Si se elimina una posición guardada, todas las posiciones a partir de ese
%valor tomarán el valor de la posición de después suya
for i=elemento:cont2-1
    c2(i)=c2(i+1);
    d2(i)=d2(i+1);
    e2(i)=e2(i+1);
    f2(i)=f2(i+1);
    g2(i)=g2(i+1);
    h2(i)=h2(i+1);
end

%Si se elimina el último elemento, el valor que marcará la listbox será el
%anterior y no el siguiente, puesto que no lo hay. También aparecerán en
%los textos de la matriz homogénea los valores de ese valor anterior. Solo
%podrá ocurrir cuando haya más de 1 posición guardada
if elemento==cont2 && cont2>1
    set(handles.lbxPosGuard, 'Value', elemento-1)

    A0=[0 0 -1 0; -1 0 0 0; 0 1 0 0; 0 0 0 1];
    A1=[sin(c2(elemento-1)) 0 cos(c2(elemento-1)) 0; -cos(c2(elemento-1)) 0
sin(c2(elemento-1)) 0; 0 -1 0 -241.1; 0 0 0 1];
    A2=[-sin(d2(elemento-1)) cos(d2(elemento-1)) 0 -495*sin(d2(elemento-1));
cos(d2(elemento-1)) sin(d2(elemento-1)) 0 ...
      495*cos(d2(elemento-1)); 0 0 -1 0; 0 0 0 1];
    A3=[sin(e2(elemento-1)) 0 -cos(e2(elemento-1)) 0; -cos(e2(elemento-1)) 0
-sin(e2(elemento-1)) 0; 0 1 0 0; 0 0 0 1];
    A4=[cos(f2(elemento-1)) 0 -sin(f2(elemento-1)) 0; sin(f2(elemento-1)) 0
cos(f2(elemento-1)) 0; 0 -1 0 -372; 0 0 0 1];
    A5=[cos(g2(elemento-1)) 0 sin(g2(elemento-1)) 0; sin(g2(elemento-1)) 0 -
cos(g2(elemento-1)) 0; 0 1 0 0; 0 0 0 1];
    A6=[-sin(h2(elemento-1)) cos(h2(elemento-1)) 0 0;cos(h2(elemento-1))
sin(h2(elemento-1)) 0 0; 0 0 -1 -183.2; 0 0 0 1];
    T=A0*A1*A2*A3*A4*A5*A6;
```

```
    %Posiciones x y z
    set(handles.txtPosExtRobotXPosGuard,'String',[num2str(round(T(1,4),2)),
'mm']);
    set(handles.txtPosExtRobotYPosGuard,'String',[num2str(round(T(2,4),2)),
'mm']);
    set(handles.txtPosExtRobotZPosGuard,'String',[num2str(round(T(3,4),2)),
'mm']);

    %Matriz de rotación
    set(handles.txtMatRotX6XPosGuard,'String',num2str(round(T(1,1),2)));
    set(handles.txtMatRotX6YPosGuard,'String',num2str(round(T(2,1),2)));
    set(handles.txtMatRotX6ZPosGuard,'String',num2str(round(T(3,1),2)));
    set(handles.txtMatRotY6XPosGuard,'String',num2str(round(T(1,2),2)));
    set(handles.txtMatRotY6YPosGuard,'String',num2str(round(T(2,2),2)));
    set(handles.txtMatRotY6ZPosGuard,'String',num2str(round(T(3,2),2)));
    set(handles.txtMatRotZ6XPosGuard,'String',num2str(round(T(1,3),2)));
    set(handles.txtMatRotZ6YPosGuard,'String',num2str(round(T(2,3),2)));
    set(handles.txtMatRotZ6ZPosGuard,'String',num2str(round(T(3,3),2)));
%Si no se elimina el último elemento, se colocarán en los textos de la
%matriz homogénea los correspondientes a la posición del elemento de después,
%que ahora está guardada en la posición que estaba marcada previamente
else
    A0=[0 0 -1 0; -1 0 0 0; 0 1 0 0; 0 0 0 1];
    A1=[sin(c2(elemento)) 0 cos(c2(elemento)) 0; -cos(c2(elemento)) 0
sin(c2(elemento)) 0; 0 -1 0 -241.1; 0 0 0 1];
    A2=[-sin(d2(elemento)) cos(d2(elemento)) 0 -495*sin(d2(elemento));
cos(d2(elemento)) sin(d2(elemento)) 0 ...
      495*cos(d2(elemento)); 0 0 -1 0; 0 0 0 1];
    A3=[sin(e2(elemento)) 0 -cos(e2(elemento)) 0; -cos(e2(elemento)) 0 -
sin(e2(elemento)) 0; 0 1 0 0; 0 0 0 1];
    A4=[cos(f2(elemento)) 0 -sin(f2(elemento)) 0; sin(f2(elemento)) 0
cos(f2(elemento)) 0; 0 -1 0 -372; 0 0 0 1];
    A5=[cos(g2(elemento)) 0 sin(g2(elemento)) 0; sin(g2(elemento)) 0 -
cos(g2(elemento)) 0; 0 1 0 0; 0 0 0 1];
    A6=[-sin(h2(elemento)) cos(h2(elemento)) 0 0;cos(h2(elemento))
sin(h2(elemento)) 0 0; 0 0 -1 -183.2; 0 0 0 1];
    T=A0*A1*A2*A3*A4*A5*A6;

    %Posiciones x y z
    set(handles.txtPosExtRobotXPosGuard,'String',[num2str(round(T(1,4),2)),
'mm']);
    set(handles.txtPosExtRobotYPosGuard,'String',[num2str(round(T(2,4),2)),
'mm']);
    set(handles.txtPosExtRobotZPosGuard,'String',[num2str(round(T(3,4),2)),
'mm']);

    %Matriz de rotación
    set(handles.txtMatRotX6XPosGuard,'String',num2str(round(T(1,1),2)));
    set(handles.txtMatRotX6YPosGuard,'String',num2str(round(T(2,1),2)));
    set(handles.txtMatRotX6ZPosGuard,'String',num2str(round(T(3,1),2)));
    set(handles.txtMatRotY6XPosGuard,'String',num2str(round(T(1,2),2)));
    set(handles.txtMatRotY6YPosGuard,'String',num2str(round(T(2,2),2)));
    set(handles.txtMatRotY6ZPosGuard,'String',num2str(round(T(3,2),2)));
    set(handles.txtMatRotZ6XPosGuard,'String',num2str(round(T(1,3),2)));
    set(handles.txtMatRotZ6YPosGuard,'String',num2str(round(T(2,3),2)));
    set(handles.txtMatRotZ6ZPosGuard,'String',num2str(round(T(3,3),2)));
end
%Si se elimina el último elemento que queda, se pone un espacio en el
%primer y único elemento de la lista para que en esta se pueda marcar algún
%elemento, puesto que si no marca ninguno la lista desaparece. También se
%dejan en blanco todos los textos de la matriz homogénea. Aclarar que la
```

```
%longitud del vector lista vale 1 porque aún no se ha actualizado la lista
%y, por tanto, aún no tiene el elemento borrado
if cont2<=1
    set(handles.lbxPosGuard, 'String',' ');
    set(handles.lbxPosGuard, 'Value', 1);

    %Posiciones x y z
    set(handles.txtPosExtRobotXPosGuard,'String',' ');
    set(handles.txtPosExtRobotYPosGuard,'String',' ');
    set(handles.txtPosExtRobotZPosGuard,'String',' ');

    %Matriz de rotación
    set(handles.txtMatRotX6XPosGuard,'String',' ');
    set(handles.txtMatRotX6YPosGuard,'String',' ');
    set(handles.txtMatRotX6ZPosGuard,'String',' ');
    set(handles.txtMatRotY6XPosGuard,'String',' ');
    set(handles.txtMatRotY6YPosGuard,'String',' ');
    set(handles.txtMatRotY6ZPosGuard,'String',' ');
    set(handles.txtMatRotZ6XPosGuard,'String',' ');
    set(handles.txtMatRotZ6YPosGuard,'String',' ');
    set(handles.txtMatRotZ6ZPosGuard,'String',' ');
%Si quedan más elementos, simplemente se actualiza la lista haciendo los
%mismos pasos que en el openingfcn, ya que hay que actualizar el número de
%cada posición
else
    set(handles.lbxPosGuard,'String', 'Posición 1');
    for i=2:cont2-1
        lista = get(handles.lbxPosGuard,'string');
        listaenchar=char(lista);
        listamasuno = char(listaenchar,char(['Posición ', num2str(i)]));
        set(handles.lbxPosGuard,'String',cellstr(listamasuno));
    end
end
%Si se da a guardar posición, cuando no haya ninguna posición disponible en
%la lista el contador no debe reducirse a un número negativo, por lo que
%solo se reducirá cuando haya más de una posición guardada sin guardar
if cont2>=1
cont2=cont2-1;
end


% --- Executes on button press in btnSalirPosGuard.
function btnSalirPosGuard_Callback(hObject, eventdata, handles)
% hObject    handle to btnSalirPosGuard (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global c d e f g h cont c2 d2 e2 f2 g2 h2 cont2;
%Si no se han guardado los cambios, se dará la opción de guardarlos o no
%una vez se intente cerrar la ventana
if cont~=cont2
    opc=questdlg('¿Guardar antes de salir?', 'SALIR', 'No', 'Sí', 'Sí');
    if strcmp(opc,'No')
        close (handles.figure1);
    elseif strcmp(opc,'Sí')
        for i=1:cont2
        c(i)=c2(i);
        d(i)=d2(i);
        e(i)=e2(i);
        f(i)=f2(i);
        g(i)=g2(i);
        h(i)=h2(i);
```

```
        end
    cont=cont2;
        close (handles.figure1);
    end
else
    close (handles.figure1);
end
```

Código de la interfaz del manejo del robot:

```
function varargout = SegundaPrueba(varargin)
% SEGUNDAPRUEBA MATLAB code for SegundaPrueba.fig
%      SEGUNDAPRUEBA, by itself, creates a new SEGUNDAPRUEBA or raises the
existing
%      singleton*.
%
%      H = SEGUNDAPRUEBA returns the handle to a new SEGUNDAPRUEBA or the
handle to
%      the existing singleton*.
%
%      SEGUNDAPRUEBA('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SEGUNDAPRUEBA.M with the given input
arguments.
%
%      SEGUNDAPRUEBA('Property','Value',...) creates a new SEGUNDAPRUEBA or
raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before SegundaPrueba_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to SegundaPrueba_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 SegundaPrueba

% Last Modified by GUIDE v2.5 08-Jul-2018 12:26:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SegundaPrueba_OpeningFcn, ...
                   'gui_OutputFcn',  @SegundaPrueba_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 SegundaPrueba is made visible.
function SegundaPrueba_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 SegundaPrueba (see VARARGIN)

%Se inicializa la variable cont, que contará el número de posiciones
%guardadas para el posterior movimiento automático del robot
global cont;
cont=0;
set(handles.lblNumPosGuardadas,'String',['N° de posiciones guardadas:
',num2str(cont)]);

%Se declara la variable movAutomatico, la cual sirve para que la función
%WindowsButtonUpcFcn solo funcione para los botones del joystick, puesto
%que cuando el robot esté en movimiento (cuando esté yendo a la posición
%inicial o cuando se esté moviendo automáticamente), movAutomatico tomará el
%valor de 1 y no se podrá salir de la interfaz. También permite que los
%botones del joystick no puedan funcionar cuando el robot esté en movimiento
global movAutomatico;
movAutomatico=0;

%Se dibuja el círculo negro central del joystick
t=0:pi/100:pi*2;
x=sin(t);
y=cos(t);
fill(x,y,'k');
axis off;

%Se pone el dibujo de despulsado a los botones
dibujo('derecha1.jpg',handles.btnDerecha);
dibujo('izquierda1.jpg',handles.btnIzquierda);
dibujo('arriba1.jpg',handles.btnArriba);
dibujo('abajo1.jpg',handles.btnAbajo);
dibujo('antihorario1.jpg',handles.btnAntiHorario);
dibujo('horario1.jpg',handles.btnHorario);

%Se inicializa la biblioteca para el manejo del robot
addpath('./dll');
if (libisloaded('m5apiw32')==0),
    loadlibrary('m5apiw32', 'm5apiw32.h');
end
%Se carga el robot
global dev;
dev=0;
calllib('m5apiw32','PCube_openDevice',dev,'ESD:0,1000');
calllib('m5apiw32','PCube_resetAll',dev);

%Se muestra la posición inicial de cada articulación
global q;
pos=0;
b=[handles.txtPosArt1 handles.txtPosArt2 handles.txtPosArt3...
    handles.txtPosArt4 handles.txtPosArt5 handles.txtPosArt6];
[~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
set(b(1),'String',[num2str(round(q(1)*180/pi,2)), '°']);
[~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
```

```
set(b(2),'String',[num2str(round(q(2)*180/pi,2)), '°']);
[~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
set(b(3),'String',[num2str(round(q(3)*180/pi,2)), '°']);
[~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
set(b(4),'String',[num2str(round(q(4)*180/pi,2)), '°']);
[~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
set(b(5),'String',[num2str(round(q(5)*180/pi,2)), '°']);
[~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
set(b(6),'String',[num2str(round(q(6)*180/pi,2)), '°']);

%Se actualizan los text box de los paneles restantes
posyrot(handles);

% Choose default command line output for SegundaPrueba
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes SegundaPrueba wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = SegundaPrueba_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;


% --- Executes on button press in tglMovArticulaciones.
function tglMovArticulaciones_Callback(hObject, eventdata, handles)
% hObject    handle to tglMovArticulaciones (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 tglMovArticulaciones state of
tglMovArticulaciones
%Cuando se pulse este botón, cambiará tanto el texto del propio botón como
%el texto del joystick, ya que se indicará qué articulaciones se pueden
%mover en ese momento
if get(hObject,'Value')==1 && get(handles.tglMovExtremoRobot,'Value')==0
    set(hObject,'String','Mov. Articulaciones 4,5 y 6');
    set(handles.lblXYArt1Y4Pos,'String', 'Art. 4 +');
    set(handles.lblXYArt1Y4Neg,'String', 'Art. 4 -');
    set(handles.lblYYArt2Y5Pos,'String', 'Art. 5 +');
    set(handles.lblYYArt2Y5Neg,'String', 'Art. 5 -');
    set(handles.lblZYArt3Y6Pos,'String', 'Art. 6 +');
    set(handles.lblZYArt3Y6Neg,'String', 'Art. 6 -');
elseif get(hObject,'Value')==0 && get(handles.tglMovExtremoRobot,'Value')==0
    set(hObject,'String','Mov. Articulaciones 1,2 y 3');
    set(handles.lblXYArt1Y4Pos,'String', 'Art. 1 +');
    set(handles.lblXYArt1Y4Neg,'String', 'Art. 1 -');
    set(handles.lblYYArt2Y5Pos,'String', 'Art. 2 +');
    set(handles.lblYYArt2Y5Neg,'String', 'Art. 2 -');
```

```
    set(handles.lblZYArt3Y6Pos,'String', 'Art. 3 +');
    set(handles.lblZYArt3Y6Neg,'String', 'Art. 3 -');
elseif get(hObject,'Value')==0 %Para que solo cambie el texto del botón
    set(hObject,'String','Mov. Articulaciones 1,2 y 3');
elseif get(hObject,'Value')==1 %Para que solo cambie el texto del botón
    set(hObject,'String','Mov. Articulaciones 4,5 y 6');
end

guidata(hObject, handles);


% --- Executes on button press in tglMovExtremoRobot.
function tglMovExtremoRobot_Callback(hObject, eventdata, handles)
% hObject    handle to tglMovExtremoRobot (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 tglMovArticulaciones state of
tglMovExtremoRobot
%Cuando se pulse este botón, cambiará tanto el texto del propio botón como
%el texto del joystick, ya que se indicará qué articulaciones se pueden
%mover en ese momento o si lo que se manejará será el extremo del robot
if get(hObject,'Value')==1

set(handles.lblActivacionMovExtremoRobot,'String','Activo','BackgroundColor',
'g');
    set(handles.lblXYArt1Y4Pos,'String', 'x +');
    set(handles.lblXYArt1Y4Neg,'String', 'x -');
    set(handles.lblYYArt2Y5Pos,'String', 'y +');
    set(handles.lblYYArt2Y5Neg,'String', 'y -');
    set(handles.lblZYArt3Y6Pos,'String', 'z +');
    set(handles.lblZYArt3Y6Neg,'String', 'z -');
elseif get(hObject,'Value')==0 &&
get(handles.tglMovArticulaciones,'Value')==0

set(handles.lblActivacionMovExtremoRobot,'String','Inactivo','BackgroundColor
','r');
    set(handles.lblXYArt1Y4Pos,'String', 'Art. 1 +');
    set(handles.lblXYArt1Y4Neg,'String', 'Art. 1 -');
    set(handles.lblYYArt2Y5Pos,'String', 'Art. 2 +');
    set(handles.lblYYArt2Y5Neg,'String', 'Art. 2 -');
    set(handles.lblZYArt3Y6Pos,'String', 'Art. 3 +');
    set(handles.lblZYArt3Y6Neg,'String', 'Art. 3 -');
elseif get(hObject,'Value')==0 &&
get(handles.tglMovArticulaciones,'Value')==1

set(handles.lblActivacionMovExtremoRobot,'String','Inactivo','BackgroundColor
','r');
    set(handles.lblXYArt1Y4Pos,'String', 'Art. 4 +');
    set(handles.lblXYArt1Y4Neg,'String', 'Art. 4 -');
    set(handles.lblYYArt2Y5Pos,'String', 'Art. 5 +');
    set(handles.lblYYArt2Y5Neg,'String', 'Art. 5 -');
    set(handles.lblZYArt3Y6Pos,'String', 'Art. 6 +');
    set(handles.lblZYArt3Y6Neg,'String', 'Art. 6 -');
end

guidata(hObject, handles);


% --- Executes on button press in btnPararRobot.
```

```
function btnPararRobot_Callback(hObject, eventdata, handles)
% hObject    handle to btnPararRobot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global dev movAutomatico;
movAutomatico=0;
calllib('m5apiw32','PCube_resetAll',dev);
calllib('m5apiw32','PCube_haltAll',dev);


% --- Executes on button press in btnSalir.
function btnSalir_Callback(hObject, eventdata, handles)
% hObject    handle to btnSalir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global dev movAutomatico;
%Si el robot se está moviendo sin ayuda del joystick, no se podrá salir
if movAutomatico==0
    opc=questdlg('¿Está seguro/a de que quiere salir?', 'SALIR',...
    'Sí', 'No', 'No');
if strcmp(opc,'No')
    return
elseif strcmp(opc,'Sí')
    calllib('m5apiw32','PCube_closeDevice',dev);
    close (handles.figure1);
end
else
    errordlg('Espere a que el robot termine de moverse o párelo.',...
        'ERROR');
    return
end


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
global dev movAutomatico;
%Si el robot se está moviendo sin ayuda del joystick, no se podrá salir
if movAutomatico==0
    calllib('m5apiw32','PCube_closeDevice',dev);
else
    errordlg('Espere a que el robot termine de moverse o párelo.',...
        'ERROR');
    return
end
delete(hObject);


% --- Executes during object creation, after setting all properties.
function pmnSeleccionVelocidad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to pmnSeleccionVelocidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown;
%Mientras se pulse el ratón, la variable buttondown valdrá 1 y servirá para
%la condición del while de todos las funciones Buttondown
buttondown=1;

guidata(hObject, handles);


% --- Executes on mouse press over figure background, over a disabled or
% --- inactive control, or over an axes background.
function figure1_WindowButtonUpFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev movAutomatico q;
buttondown=0;
%Cuando se despulsa el ratón, se les pone a las imágenes el dibujo de
%despulsado
if movAutomatico==0
    dibujo('derecha1.jpg',handles.btnDerecha);
    dibujo('izquierda1.jpg',handles.btnIzquierda);
    dibujo('arriba1.jpg',handles.btnArriba);
    dibujo('abajo1.jpg',handles.btnAbajo);
    dibujo('antihorario1.jpg',handles.btnAntiHorario);
    dibujo('horario1.jpg',handles.btnHorario);

    %También se para todo movimiento del robot cuando se despulsa el ratón
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_haltAll',dev);
end

%Además, se actualizará completamente el valor de los text box de los
%paneles, ya que si no se hace aparecerán los valores de justo el momento
%antes de parar de pulsar el joystick
b=[handles.txtPosArt1 handles.txtPosArt2 handles.txtPosArt3...
    handles.txtPosArt4 handles.txtPosArt5 handles.txtPosArt6];
pos=0;
[~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
[~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
[~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
[~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
[~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
[~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
set(b(1),'String',[num2str(round(q(1)*180/pi,2)), '°']);
set(b(2),'String',[num2str(round(q(2)*180/pi,2)), '°']);
set(b(3),'String',[num2str(round(q(3)*180/pi,2)), '°']);
set(b(4),'String',[num2str(round(q(4)*180/pi,2)), '°']);
set(b(5),'String',[num2str(round(q(5)*180/pi,2)), '°']);
```

```
set(b(6),'String',[num2str(round(q(6)*180/pi,2)), '°']);
posyrot(handles);

guidata(hObject, handles);


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over btnArriba.
function btnArriba_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to btnArriba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev q movAutomatico Pm T;
pos=0;
%Dependiendo del valor escogido en el Pop-up menu, la velocidad será una u
%otra
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
        vel=0.1;
    case 3
        vel=0.5;
end
%Cuando esté activo el "mover articulaciones 1, 2 y 3" e inactivo "mover el
%extremo del robot" se moverá la articulación 1
if get(handles.tglMovArticulaciones,'Value')==0 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    %Se establece un límite de posición de la articulación del brazo
    maxpos=pi;
    %Se pone el dibujo de pulsado en el botón que se pulsa, en este caso el
    %de arriba
    dibujo('arriba2.jpg',handles.btnArriba);
    %Se mueve la articulación, en este caso la 1
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 1, maxpos, vel, 0.2);
    while buttondown==1
        %Se pide la posición de la articulación y se establece en el texto
        [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
        set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
        %Se llama a la función posyrot para que se actualicen las posiciónes
        %x y z, la matriz de rotación y la velocidad del extremo del robot
        posyrot(handles);
        %Cuando se llega al límite aparece un mensaje de error
        if round(q(1)*180/pi)==round(maxpos*180/pi)
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        %Se establece cada cuánto tiempo se quiere que se actualice el valor
        %del texto para que no cambie tan rápido y se pueda ver el cambio
        %con claridad
        pause(0.05);
    end
    posyrot(handles);
%Cuando esté activo el "mover articulaciones 4, 5 y 6" e inactivo "mover el
%extremo del robot" se moverá la articulación 4, y los pasos serán los
%mismos que para la articulación 1
```

```
elseif get(handles.tglMovArticulaciones,'Value')==1 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    maxpos=pi;
    dibujo('arriba2.jpg',handles.btnArriba);
    %Mover articulación 4
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 4, maxpos, vel, 0.2);
    while buttondown==1
        [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
        set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(4)*180/pi)==round(maxpos*180/pi)
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovExtremoRobot,'Value')==1 && movAutomatico==0
    dibujo('arriba2.jpg',handles.btnArriba);
    maxposq2=2.2;
    minposq2=-0.873;
    maxposq3=2.5;
    maxposq4=pi;
    maxposq5=pi/2;
    maxposq6=pi;
    %Se halla la matriz de rotación R06 para conocer cuál es la orientación
    %que se debe mantener del extremo del robot
    R06=T(1:3,1:3);
    %Mover extremo x+
    calllib('m5apiw32','PCube_resetAll',dev);
    %Con la cinemática inversa se haya la posición a la que tiene que ir
    %cada una de las articulaciones para conseguir una trayectoria en línea
    %recta en el eje x y para mantener orientado de la misma manera el
    %extremo del robot. Para ello, en este caso, el valor de Pmx va
    %aumentando de 1 en 1 para que la muñeca vaya de punto a punto para
    %conseguir esta trayectoria en línea recta. Si q(1) vale 0 o pi, las
    %ecuaciones de la cinemática inversa son diferentes. A la hora de hacer
    %la trayectoria en el eje x la articulación 1 no se moverá, mientras
    %que cuando se haga en eje y o z sí que lo hará

    %Lo que dará fin a la operación será un if, por lo que aquí solo se pone
    %de condición el buttondown. A la hora de ponerle el signo a posq2,
    %como el acos() solo halla valores del primer y segundo cuadrante (de 0
    %a pi), cuando el ángulo actual de la articulación sea negativo, le
    %corresponderá un valor negativo de 0 a -pi, es decir, un valor del
    %tercer o cuarto cuadrante, por lo que se debe cambiar el signo a mano
    while buttondown==1
        %Si al hallar posq1 se intenta dividir 0/0, quiere decir que hay
        %una signularidad y que hay infinitas soluciones, lo cual hay que
        %evitar
        if Pm(2)==0 && Pm(3)==0
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
```

```
        end
        posq1=atan2(Pm(2),Pm(3));
        %Dependiendo del valor de posq1 se trabajará con cos(posq1) o con
        %sin(posq1), ya que para hallar posq2 y posq3 hay que dividir en su
        %numerador entre uno de los dos
        if abs(round(posq1*180/pi))<=1 || abs(round(posq1*180/pi))>=179
            if q(3)>=0
                posq3=acos(((Pm(3)/cos(posq1))^2+(Pm(1)+1-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(3)/cos(posq1))^2+(Pm(1)+1-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            %Cuando la solución de posq3 sea no real (compleja), el brazo no
            %podrá avanzar más, puesto que no habrá solución real
            if isreal(posq3)==0
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)+1-241.1)-
Pm(3)/cos(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(3)/cos(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        else
            if q(3)>=0
                posq3=acos(((Pm(2)/sin(posq1))^2+(Pm(1)+1-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(2)/sin(posq1))^2+(Pm(1)+1-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)+1-241.1)-
Pm(2)/sin(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(2)/sin(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        end
        %Se halla la matriz de rotación R03 para poder hallar R36 y con
        %ella obtener los valores de las coordenadas articulares restantes
        R0=[0 0 -1; -1 0 0; 0 1 0];
        R1=[sin(posq1) 0 cos(posq1); -cos(posq1) 0 sin(posq1); 0 -1 0];
        R2=[-sin(posq2) cos(posq2) 0; cos(posq2) sin(posq2) 0; 0 0 -1];
        R3=[sin(posq3) 0 -cos(posq3); -cos(posq3) 0 -sin(posq3); 0 1 0];
        R03=R0*R1*R2*R3;
        R36=R03'*R06;
        posq4=atan2(-R36(2,3),-R36(1,3));
```

```
        if q(5)>=0
            posq5=acos(-R36(3,3));
        else
            posq5=-acos(-R36(3,3));
        end
        posq6=atan2(R36(3,1),-R36(3,2));
        %Se halla la velocidad inicial de cada articulación para aplicarla
        %en la velocidad que deba tener cada una de ellas
        vel0=0;
        [~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel0);
        [~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel0);
        [~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel0);
        [~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel0);
        [~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel0);
        %Se halla la máxima diferencia de posiciones para saber cuál es la
        %articulación que tiene que moverse más rápido
        maxDifAbs=abs(posq2-q(2));
        maxDif=posq2-q(2);
        velInicArticMasRapida=qd(2);
        if abs(posq3-q(3))>maxDifAbs
            maxDifAbs=abs(posq3-q(3));
            maxDif=posq3-q(3);
            velInicArticMasRapida=qd(3);
        end
        if abs(posq4-q(4))>maxDifAbs
            maxDifAbs=abs(posq4-q(4));
            maxDif=posq4-q(4);
            velInicArticMasRapida=qd(4);
        end
        if abs(posq5-q(5))>maxDifAbs
            maxDifAbs=abs(posq5-q(5));
            maxDif=posq5-q(5);
            velInicArticMasRapida=qd(5);
        end
        if abs(posq6-q(6))>maxDifAbs
            maxDifAbs=abs(posq6-q(6));
            maxDif=posq6-q(6);
            velInicArticMasRapida=qd(6);
        end
        %Si la maxima diferencia es mayor que 180, querrá decir que la
        %función atan2 ha pasado de dar valores positivos a valores
        %negativos (o viceversa) por pasar del límite de 180 al de -180 (da
        %la vuelta)
        if maxDifAbs*180/pi>180
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        %A la velocidad le pondrá el signo la diferencia de posiciones,
        %así, cuando sea negativo la articulación se dirigirá a la
        %mínima posición, y cuando sea positivo hacia la máxima
        calllib ('m5apiw32', 'PCube_moveVel', dev, 2,...
            (posq2-q(2))/maxDifAbs*vel+(posq2-
q(2))/maxDif*velInicArticMasRapida-qd(2));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 3,...
            (posq3-q(3))/maxDifAbs*vel+(posq3-
q(3))/maxDif*velInicArticMasRapida-qd(3));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 4,...
```

```
            (posq4-q(4))/maxDifAbs*vel+(posq4-
q(4))/maxDif*velInicArticMasRapida-qd(4));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 5,...
            (posq5-q(5))/maxDifAbs*vel+(posq5-
q(5))/maxDif*velInicArticMasRapida-qd(5));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 6,...
            (posq6-q(6))/maxDifAbs*vel+(posq6-
q(6))/maxDif*velInicArticMasRapida-qd(6));
        %Cuando la articulación que más camino tenga que recorrer llegue a
        %la posición especificada (en principio todas las articulaciones
        %llegarán a la vez), se saldrá del bucle while y se hallarán nuevas
        %posiciones objetivo
        while round(maxDif*180/pi)~=0 && buttondown==1
            [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
            set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
            [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
            set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
            [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
            set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
            [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
            set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
            [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
            set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
            posyrot(handles);
            %Se ponen valores absolutos porque los límites superior e
inferior
            %son el mismo pero con distinto signo
            if round(q(2)*180/pi)>=round(maxposq2*180/pi) || ...
                    round(q(2)*180/pi)<=round(minposq2*180/pi) || ...
                    abs(round(q(3)*180/pi))>=round(maxposq3*180/pi) || ...
                    abs(round(q(3)*180/pi))<=1 || ...
                    abs(round(q(4)*180/pi))>=round(maxposq4*180/pi) || ...
                    abs(round(q(5)*180/pi))>=round(maxposq5*180/pi) || ...
                    abs(round(q(5)*180/pi))<=5 || ...
                    abs(round(q(6)*180/pi))>=round(maxposq6*180/pi)
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            pause(0.05);
        end
    end
end

%Todos los demás botones del joystick funcionan igual que este
guidata(hObject, handles);


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over btnAbajo.
function btnAbajo_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to btnAbajo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev q movAutomatico Pm T;
pos=0;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
        vel=0.1;
    case 3
        vel=0.5;
end
if get(handles.tglMovArticulaciones,'Value')==0 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    minpos=-pi;
    dibujo('abajo2.jpg',handles.btnAbajo);
    %Mover articulación 1
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 1, minpos, vel, 0.2);
    while buttondown==1
        [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
        set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(1)*180/pi)==round(minpos*180/pi)
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovArticulaciones,'Value')==1 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    minpos=-pi;
    dibujo('abajo2.jpg',handles.btnAbajo);
    %Mover articulación 4
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 4, minpos, vel, 0.2);
    while buttondown==1
        [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
        set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(4)*180/pi)==round(minpos*180/pi)
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovExtremoRobot,'Value')==1 && movAutomatico==0
    dibujo('abajo2.jpg',handles.btnAbajo);
    maxposq2=2.2;
    minposq2=-0.873;
    maxposq3=2.5;
    maxposq4=pi;
    maxposq5=pi/2;
```

```
    maxposq6=pi;
    R06=T(1:3,1:3);
    %Mover extremo x-
    calllib('m5apiw32','PCube_resetAll',dev);
    while buttondown==1
        if Pm(2)==0 && Pm(3)==0
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        posq1=atan2(Pm(2),Pm(3));
        if abs(round(posq1*180/pi))<=1 || abs(round(posq1*180/pi))>=179
            if q(3)>=0
                posq3=acos(((Pm(3)/cos(posq1))^2+(Pm(1)-1-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(3)/cos(posq1))^2+(Pm(1)-1-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-1-241.1)-
Pm(3)/cos(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(3)/cos(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        else
            if q(3)>=0
                posq3=acos(((Pm(2)/sin(posq1))^2+(Pm(1)-1-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(2)/sin(posq1))^2+(Pm(1)-1-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-1-241.1)-
Pm(2)/sin(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(2)/sin(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        end
        R0=[0 0 -1; -1 0 0; 0 1 0];
        R1=[sin(posq1) 0 cos(posq1); -cos(posq1) 0 sin(posq1); 0 -1 0];
```

```
        R2=[-sin(posq2) cos(posq2) 0; cos(posq2) sin(posq2) 0; 0 0 -1];
        R3=[sin(posq3) 0 -cos(posq3); -cos(posq3) 0 -sin(posq3); 0 1 0];
        R03=R0*R1*R2*R3;
        R36=R03'*R06;
        posq4=atan2(-R36(2,3),-R36(1,3));
        if q(5)>=0
            posq5=acos(-R36(3,3));
        else
            posq5=-acos(-R36(3,3));
        end
        posq6=atan2(R36(3,1),-R36(3,2));
        vel0=0;
        [~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel0);
        [~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel0);
        [~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel0);
        [~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel0);
        [~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel0);
        maxDifAbs=abs(posq2-q(2));
        maxDif=posq2-q(2);
        velInicArticMasRapida=qd(2);
        if abs(posq3-q(3))>maxDifAbs
            maxDifAbs=abs(posq3-q(3));
            maxDif=posq3-q(3);
            velInicArticMasRapida=qd(3);
        end
        if abs(posq4-q(4))>maxDifAbs
            maxDifAbs=abs(posq4-q(4));
            maxDif=posq4-q(4);
            velInicArticMasRapida=qd(4);
        end
        if abs(posq5-q(5))>maxDifAbs
            maxDifAbs=abs(posq5-q(5));
            maxDif=posq5-q(5);
            velInicArticMasRapida=qd(5);
        end
        if abs(posq6-q(6))>maxDifAbs
            maxDifAbs=abs(posq6-q(6));
            maxDif=posq6-q(6);
            velInicArticMasRapida=qd(6);
        end
        if maxDifAbs*180/pi>180
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        calllib ('m5apiw32', 'PCube_moveVel', dev, 2,...
            (posq2-q(2))/maxDifAbs*vel+(posq2-
q(2))/maxDif*velInicArticMasRapida-qd(2));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 3,...
            (posq3-q(3))/maxDifAbs*vel+(posq3-
q(3))/maxDif*velInicArticMasRapida-qd(3));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 4,...
            (posq4-q(4))/maxDifAbs*vel+(posq4-
q(4))/maxDif*velInicArticMasRapida-qd(4));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 5,...
            (posq5-q(5))/maxDifAbs*vel+(posq5-
q(5))/maxDif*velInicArticMasRapida-qd(5));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 6,...
```

```
            (posq6-q(6))/maxDifAbs*vel+(posq6-
q(6))/maxDif*velInicArticMasRapida-qd(6));
        while round(maxDif*180/pi)~=0 && buttondown==1
            [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
            set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
            [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
            set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
            [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
            set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
            [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
            set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
            [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
            set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
            posyrot(handles);
            if round(q(2)*180/pi)>=round(maxposq2*180/pi) || ...
                    round(q(2)*180/pi)<=round(minposq2*180/pi) || ...
                    abs(round(q(3)*180/pi))>=round(maxposq3*180/pi) || ...
                    abs(round(q(3)*180/pi))<=1 || ...
                    abs(round(q(4)*180/pi))>=round(maxposq4*180/pi) || ...
                    abs(round(q(5)*180/pi))>=round(maxposq5*180/pi) || ...
                    abs(round(q(5)*180/pi))<=5 || ...
                    abs(round(q(6)*180/pi))>=round(maxposq6*180/pi)
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            pause(0.05);
        end
    end
end

guidata(hObject, handles);


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
btnIzquierda.
function btnIzquierda_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to btnIzquierda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev q movAutomatico Pm T;
pos=0;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
        vel=0.1;
    case 3
        vel=0.5;
end
```

```
if get(handles.tglMovArticulaciones,'Value')==0 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    maxpos=2.2;
    dibujo('izquierda2.jpg',handles.btnIzquierda);
    %Mover articulación 2
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 2, maxpos, vel, 0.2);
    while buttondown==1
        [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
        set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(2)*180/pi)==round(maxpos*180/pi)
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovArticulaciones,'Value')==1 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    maxpos=pi/2;
    dibujo('izquierda2.jpg',handles.btnIzquierda);
    %Mover articulación 5
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 5, maxpos, vel, 0.2);
    while buttondown==1
        [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
        set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(5)*180/pi)==round(maxpos*180/pi)
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovExtremoRobot,'Value')==1 && movAutomatico==0
    dibujo('izquierda2.jpg',handles.btnIzquierda);
    maxposq1=pi;
    maxposq2=2.2;
    minposq2=-0.873;
    maxposq3=2.5;
    maxposq4=pi;
    maxposq5=pi/2;
    maxposq6=pi;
    R06=T(1:3,1:3);
    %Mover extremo y+
    calllib('m5apiw32','PCube_resetAll',dev);
    while buttondown==1
        if Pm(2)+1==0 && Pm(3)==0
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
```

```
            return
        end
        posq1=atan2((Pm(2)+1),Pm(3));
        if abs(round(posq1*180/pi))<=1 || abs(round(posq1*180/pi))>=179
            if q(3)>=0
                posq3=acos(((Pm(3)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(3)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-
Pm(3)/cos(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(3)/cos(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        else
            if q(3)>=0
                posq3=acos((((Pm(2)+1)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos((((Pm(2)+1)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-
(Pm(2)+1)/sin(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=((Pm(2)+1)/sin(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        end
        R0=[0 0 -1; -1 0 0; 0 1 0];
        R1=[sin(posq1) 0 cos(posq1); -cos(posq1) 0 sin(posq1); 0 -1 0];
        R2=[-sin(posq2) cos(posq2) 0; cos(posq2) sin(posq2) 0; 0 0 -1];
        R3=[sin(posq3) 0 -cos(posq3); -cos(posq3) 0 -sin(posq3); 0 1 0];
        R03=R0*R1*R2*R3;
        R36=R03'*R06;
        posq4=atan2(-R36(2,3),-R36(1,3));
        if q(5)>=0
            posq5=acos(-R36(3,3));
        else
            posq5=-acos(-R36(3,3));
        end
        posq6=atan2(R36(3,1),-R36(3,2));
```

```
        vel0=0;
        [~, qd(1)] = calllib('m5apiw32','PCube_getVel', dev, 1, vel0);
        [~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel0);
        [~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel0);
        [~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel0);
        [~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel0);
        [~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel0);
        maxDifAbs=abs(posq1-q(1));
        maxDif=posq1-q(1);
        velInicArticMasRapida=qd(1);
        if abs(posq2-q(2))>maxDifAbs
            maxDifAbs=abs(posq2-q(2));
            maxDif=posq2-q(2);
            velInicArticMasRapida=qd(2);
        end
        if abs(posq3-q(3))>maxDifAbs
            maxDifAbs=abs(posq3-q(3));
            maxDif=posq3-q(3);
            velInicArticMasRapida=qd(3);
        end
        if abs(posq4-q(4))>maxDifAbs
            maxDifAbs=abs(posq4-q(4));
            maxDif=posq4-q(4);
            velInicArticMasRapida=qd(4);
        end
        if abs(posq5-q(5))>maxDifAbs
            maxDifAbs=abs(posq5-q(5));
            maxDif=posq5-q(5);
            velInicArticMasRapida=qd(5);
        end
        if abs(posq6-q(6))>maxDifAbs
            maxDifAbs=abs(posq6-q(6));
            maxDif=posq6-q(6);
            velInicArticMasRapida=qd(6);
        end
        if maxDifAbs*180/pi>180
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        calllib ('m5apiw32', 'PCube_moveVel', dev, 1,...
            (posq1-q(1))/maxDifAbs*vel+(posq1-
q(1))/maxDif*velInicArticMasRapida-qd(1));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 2,...
            (posq2-q(2))/maxDifAbs*vel+(posq2-
q(2))/maxDif*velInicArticMasRapida-qd(2));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 3,...
            (posq3-q(3))/maxDifAbs*vel+(posq3-
q(3))/maxDif*velInicArticMasRapida-qd(3));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 4,...
            (posq4-q(4))/maxDifAbs*vel+(posq4-
q(4))/maxDif*velInicArticMasRapida-qd(4));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 5,...
            (posq5-q(5))/maxDifAbs*vel+(posq5-
q(5))/maxDif*velInicArticMasRapida-qd(5));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 6,...
            (posq6-q(6))/maxDifAbs*vel+(posq6-
q(6))/maxDif*velInicArticMasRapida-qd(6));
```

```
        while round(maxDif*180/pi)~=0 && buttondown==1
            [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
            set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
            [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
            set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
            [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
            set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
            [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
            set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
            [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
            set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
            [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
            set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
            posyrot(handles);
            if abs(round(q(1)*180/pi))>=round(maxposq1*180/pi) || ...
                    round(q(2)*180/pi)>=round(maxposq2*180/pi) || ...
                    round(q(2)*180/pi)<=round(minposq2*180/pi) || ...
                    abs(round(q(3)*180/pi))>=round(maxposq3*180/pi) || ...
                    abs(round(q(3)*180/pi))<=1 || ...
                    abs(round(q(4)*180/pi))>=round(maxposq4*180/pi) || ...
                    abs(round(q(5)*180/pi))>=round(maxposq5*180/pi) || ...
                    abs(round(q(5)*180/pi))<=5 || ...
                    abs(round(q(6)*180/pi))>=round(maxposq6*180/pi)
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            pause(0.05);
        end
    end
end

guidata(hObject, handles);


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
btnDerecha.
function btnDerecha_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to btnDerecha (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev q movAutomatico Pm T;
pos=0;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
        vel=0.1;
    case 3
        vel=0.5;
```

```
end
if get(handles.tglMovArticulaciones,'Value')==0 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    minpos=-0.873;
    dibujo('derecha2.jpg',handles.btnDerecha);
    %Mover articulación 2
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 2, minpos, vel, 0.2);
    while buttondown==1
        [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
        set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(2)*180/pi)==round(minpos*180/pi)
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovArticulaciones,'Value')==1 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    minpos=-pi/2;
    dibujo('derecha2.jpg',handles.btnDerecha);
    %Mover articulación 5
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 5, minpos, vel, 0.2);
    while buttondown==1
        [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
        set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(5)*180/pi)==round(minpos*180/pi)
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovExtremoRobot,'Value')==1 && movAutomatico==0
    dibujo('derecha2.jpg',handles.btnDerecha);
    maxposq1=pi;
    maxposq2=2.2;
    minposq2=-0.873;
    maxposq3=2.5;
    maxposq4=pi;
    maxposq5=pi/2;
    maxposq6=pi;
    R06=T(1:3,1:3);
    %Mover extremo y-
    calllib('m5apiw32','PCube_resetAll',dev);
    while buttondown==1
        if Pm(2)-1==0 && Pm(3)==0
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
```

```
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        posq1=atan2((Pm(2)-1),Pm(3));
        if abs(round(posq1*180/pi))<=1 || abs(round(posq1*180/pi))>=179
            if q(3)>=0
                posq3=acos(((Pm(3)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(3)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-
Pm(3)/cos(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(3)/cos(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        else
            if q(3)>=0
                posq3=acos((((Pm(2)-1)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos((((Pm(2)-1)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-(Pm(2)-
1)/sin(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);
            senq2=((Pm(2)-
1)/sin(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        end
        R0=[0 0 -1; -1 0 0; 0 1 0];
        R1=[sin(posq1) 0 cos(posq1); -cos(posq1) 0 sin(posq1); 0 -1 0];
        R2=[-sin(posq2) cos(posq2) 0; cos(posq2) sin(posq2) 0; 0 0 -1];
        R3=[sin(posq3) 0 -cos(posq3); -cos(posq3) 0 -sin(posq3); 0 1 0];
        R03=R0*R1*R2*R3;
        R36=R03'*R06;
        posq4=atan2(-R36(2,3),-R36(1,3));
        if q(5)>=0
            posq5=acos(-R36(3,3));
        else
            posq5=-acos(-R36(3,3));
        end
```

```
        posq6=atan2(R36(3,1),-R36(3,2));
        vel0=0;
        [~, qd(1)] = calllib('m5apiw32','PCube_getVel', dev, 1, vel0);
        [~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel0);
        [~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel0);
        [~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel0);
        [~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel0);
        [~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel0);
        maxDifAbs=abs(posq1-q(1));
        maxDif=posq1-q(1);
        velInicArticMasRapida=qd(1);
        if abs(posq2-q(2))>maxDifAbs
            maxDifAbs=abs(posq2-q(2));
            maxDif=posq2-q(2);
            velInicArticMasRapida=qd(2);
        end
        if abs(posq3-q(3))>maxDifAbs
            maxDifAbs=abs(posq3-q(3));
            maxDif=posq3-q(3);
            velInicArticMasRapida=qd(3);
        end
        if abs(posq4-q(4))>maxDifAbs
            maxDifAbs=abs(posq4-q(4));
            maxDif=posq4-q(4);
            velInicArticMasRapida=qd(4);
        end
        if abs(posq5-q(5))>maxDifAbs
            maxDifAbs=abs(posq5-q(5));
            maxDif=posq5-q(5);
            velInicArticMasRapida=qd(5);
        end
        if abs(posq6-q(6))>maxDifAbs
            maxDifAbs=abs(posq6-q(6));
            maxDif=posq6-q(6);
            velInicArticMasRapida=qd(6);
        end
        if maxDifAbs*180/pi>180
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        calllib ('m5apiw32', 'PCube_moveVel', dev, 1,...
            (posq1-q(1))/maxDifAbs*vel+(posq1-
q(1))/maxDif*velInicArticMasRapida-qd(1));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 2,...
            (posq2-q(2))/maxDifAbs*vel+(posq2-
q(2))/maxDif*velInicArticMasRapida-qd(2));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 3,...
            (posq3-q(3))/maxDifAbs*vel+(posq3-
q(3))/maxDif*velInicArticMasRapida-qd(3));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 4,...
            (posq4-q(4))/maxDifAbs*vel+(posq4-
q(4))/maxDif*velInicArticMasRapida-qd(4));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 5,...
            (posq5-q(5))/maxDifAbs*vel+(posq5-
q(5))/maxDif*velInicArticMasRapida-qd(5));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 6,...
```

```
            (posq6-q(6))/maxDifAbs*vel+(posq6-
q(6))/maxDif*velInicArticMasRapida-qd(6));
        while round(maxDif*180/pi)~=0 && buttondown==1
            [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
            set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
            [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
            set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
            [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
            set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
            [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
            set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
            [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
            set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
            [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
            set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
            posyrot(handles);
            if abs(round(q(1)*180/pi))>=round(maxposq1*180/pi) || ...
                    round(q(2)*180/pi)>=round(maxposq2*180/pi) || ...
                    round(q(2)*180/pi)<=round(minposq2*180/pi) || ...
                    abs(round(q(3)*180/pi))>=round(maxposq3*180/pi) || ...
                    abs(round(q(3)*180/pi))<=1 || ...
                    abs(round(q(4)*180/pi))>=round(maxposq4*180/pi) || ...
                    abs(round(q(5)*180/pi))>=round(maxposq5*180/pi) || ...
                    abs(round(q(5)*180/pi))<=5 || ...
                    abs(round(q(6)*180/pi))>=round(maxposq6*180/pi)
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            pause(0.05);
        end
    end
end

guidata(hObject, handles);


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
btnHorario.
function btnHorario_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to btnHorario (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev q movAutomatico Pm T;
pos=0;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
        vel=0.1;
```

```
    case 3
        vel=0.5;
end
if get(handles.tglMovArticulaciones,'Value')==0 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    maxpos=2.5;
    dibujo('horario2.jpg',handles.btnHorario);
    %Mover articulación 3
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 3, maxpos, vel, 0.2);
    while buttondown==1
        [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
        set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(3)*180/pi)==round(maxpos*180/pi)
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovArticulaciones,'Value')==1 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    maxpos=pi;
    dibujo('horario2.jpg',handles.btnHorario);
    %Mover articulación 6
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 6, maxpos, vel, 0.2);
    while buttondown==1
        [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
        set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(6)*180/pi)==round(maxpos*180/pi)
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovExtremoRobot,'Value')==1 && movAutomatico==0
    dibujo('horario2.jpg',handles.btnHorario);
    maxposq1=pi;
    maxposq2=2.2;
    minposq2=-0.873;
    maxposq3=2.5;
    maxposq4=pi;
    maxposq5=pi/2;
    maxposq6=pi;
    R06=T(1:3,1:3);
    %Mover extremo z+
    calllib('m5apiw32','PCube_resetAll',dev);
    while buttondown==1
        if Pm(2)==0 && Pm(3)+1==0
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
```

```
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        posq1=atan2((Pm(2)),(Pm(3)+1));
        if abs(round(posq1*180/pi))<=1 || abs(round(posq1*180/pi))>=179
            if q(3)>=0
                posq3=acos((((Pm(3)+1)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos((((Pm(3)+1)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-
(Pm(3)+1)/cos(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=((Pm(3)+1)/cos(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        else
            if q(3)>=0
                posq3=acos(((Pm(2)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(2)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-
Pm(2)/sin(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(2)/sin(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        end
        R0=[0 0 -1; -1 0 0; 0 1 0];
        R1=[sin(posq1) 0 cos(posq1); -cos(posq1) 0 sin(posq1); 0 -1 0];
        R2=[-sin(posq2) cos(posq2) 0; cos(posq2) sin(posq2) 0; 0 0 -1];
        R3=[sin(posq3) 0 -cos(posq3); -cos(posq3) 0 -sin(posq3); 0 1 0];
        R03=R0*R1*R2*R3;
        R36=R03'*R06;
        posq4=atan2(-R36(2,3),-R36(1,3));
        if q(5)>=0
            posq5=acos(-R36(3,3));
        else
```

```
            posq5=-acos(-R36(3,3));
        end
        posq6=atan2(R36(3,1),-R36(3,2));
        vel0=0;
        [~, qd(1)] = calllib('m5apiw32','PCube_getVel', dev, 1, vel0);
        [~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel0);
        [~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel0);
        [~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel0);
        [~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel0);
        [~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel0);
        maxDifAbs=abs(posq1-q(1));
        maxDif=posq1-q(1);
        velInicArticMasRapida=qd(1);
        if abs(posq2-q(2))>maxDifAbs
            maxDifAbs=abs(posq2-q(2));
            maxDif=posq2-q(2);
            velInicArticMasRapida=qd(2);
        end
        if abs(posq3-q(3))>maxDifAbs
            maxDifAbs=abs(posq3-q(3));
            maxDif=posq3-q(3);
            velInicArticMasRapida=qd(3);
        end
        if abs(posq4-q(4))>maxDifAbs
            maxDifAbs=abs(posq4-q(4));
            maxDif=posq4-q(4);
            velInicArticMasRapida=qd(4);
        end
        if abs(posq5-q(5))>maxDifAbs
            maxDifAbs=abs(posq5-q(5));
            maxDif=posq5-q(5);
            velInicArticMasRapida=qd(5);
        end
        if abs(posq6-q(6))>maxDifAbs
            maxDifAbs=abs(posq6-q(6));
            maxDif=posq6-q(6);
            velInicArticMasRapida=qd(6);
        end
        if maxDifAbs*180/pi>180
            errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        calllib ('m5apiw32', 'PCube_moveVel', dev, 1,...
            (posq1-q(1))/maxDifAbs*vel+(posq1-
q(1))/maxDif*velInicArticMasRapida-qd(1));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 2,...
            (posq2-q(2))/maxDifAbs*vel+(posq2-
q(2))/maxDif*velInicArticMasRapida-qd(2));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 3,...
            (posq3-q(3))/maxDifAbs*vel+(posq3-
q(3))/maxDif*velInicArticMasRapida-qd(3));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 4,...
            (posq4-q(4))/maxDifAbs*vel+(posq4-
q(4))/maxDif*velInicArticMasRapida-qd(4));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 5,...
            (posq5-q(5))/maxDifAbs*vel+(posq5-
q(5))/maxDif*velInicArticMasRapida-qd(5));
```

```
        calllib ('m5apiw32', 'PCube_moveVel', dev, 6,...
            (posq6-q(6))/maxDifAbs*vel+(posq6-
q(6))/maxDif*velInicArticMasRapida-qd(6));
        while round(maxDif*180/pi)~=0 && buttondown==1
            [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
            set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
            [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
            set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
            [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
            set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
            [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
            set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
            [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
            set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
            [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
            set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
            posyrot(handles);
            if abs(round(q(1)*180/pi))>=round(maxposq1*180/pi) || ...
                    round(q(2)*180/pi)>=round(maxposq2*180/pi) || ...
                    round(q(2)*180/pi)<=round(minposq2*180/pi) || ...
                    abs(round(q(3)*180/pi))>=round(maxposq3*180/pi) || ...
                    abs(round(q(3)*180/pi))<=1 || ...
                    abs(round(q(4)*180/pi))>=round(maxposq4*180/pi) || ...
                    abs(round(q(5)*180/pi))>=round(maxposq5*180/pi) || ...
                    abs(round(q(5)*180/pi))<=5 || ...
                    abs(round(q(6)*180/pi))>=round(maxposq6*180/pi)
                errordlg('Esta es la máxima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            pause(0.05);
        end
    end
end

guidata(hObject, handles);


% --- If Enable == 'on', executes on mouse press in 5 pixel border.
% --- Otherwise, executes on mouse press in 5 pixel border or over
btnAntiHorario.
function btnAntiHorario_ButtonDownFcn(hObject, eventdata, handles)
% hObject    handle to btnAntiHorario (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global buttondown dev q movAutomatico Pm T;
pos=0;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
```

```
        vel=0.1;
    case 3
        vel=0.5;
end
if get(handles.tglMovArticulaciones,'Value')==0 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    minpos=-2.5;
    dibujo('antihorario2.jpg',handles.btnAntiHorario);
    %Mover articulación 3
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 3, minpos, vel, 0.2);
    while buttondown==1
        [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
        set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(3)*180/pi)==round(minpos*180/pi)
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovArticulaciones,'Value')==1 &&
get(handles.tglMovExtremoRobot,'Value')==0 && movAutomatico==0
    minpos=-pi;
    dibujo('antihorario2.jpg',handles.btnAntiHorario);
    %Mover articulación 6
    calllib('m5apiw32','PCube_resetAll',dev);
    calllib('m5apiw32','PCube_moveRamp', dev, 6, minpos, vel, 0.2);
    while buttondown==1
        [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
        set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
        posyrot(handles);
        if round(q(6)*180/pi)==round(minpos*180/pi)
          errordlg('Esta es la mínima posición, no se puede avanzar más.',...
                'ERROR');
            return
        end
        pause(0.05);
    end
    posyrot(handles);
elseif get(handles.tglMovExtremoRobot,'Value')==1 && movAutomatico==0
    dibujo('antihorario2.jpg',handles.btnAntiHorario);
    maxposq1=pi;
    maxposq2=2.2;
    minposq2=-0.873;
    maxposq3=2.5;
    maxposq4=pi;
    maxposq5=pi/2;
    maxposq6=pi;
    R06=T(1:3,1:3);
    %Mover extremo z-
    calllib('m5apiw32','PCube_resetAll',dev);
    while buttondown==1
        if Pm(2)==0 && Pm(3)-1==0
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
```

```
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        posq1=atan2((Pm(2)),(Pm(3)-1));
        if abs(round(posq1*180/pi))<=1 || abs(round(posq1*180/pi))>=179
            if q(3)>=0
                posq3=acos((((Pm(3)-1)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos((((Pm(3)-1)/cos(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-(Pm(3)-
1)/cos(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);
            senq2=((Pm(3)-
1)/cos(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        else
            if q(3)>=0
                posq3=acos(((Pm(2)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            else
                posq3=-acos(((Pm(2)/sin(posq1))^2+(Pm(1)-241.1)^2-495^2-
372^2)/(2*495*372));
            end
            if isreal(posq3)==0
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            cosq2=((495+372*cos(posq3))*(Pm(1)-241.1)-
Pm(2)/sin(posq1)*sin(posq3)...
                    *372)/(495^2+2*495*372*cos(posq3)+372^2);

senq2=(Pm(2)/sin(posq1)+372*cosq2*sin(posq3))/(495+372*cos(posq3));
            posq2=atan2(senq2,cosq2);
        end
        R0=[0 0 -1; -1 0 0; 0 1 0];
        R1=[sin(posq1) 0 cos(posq1); -cos(posq1) 0 sin(posq1); 0 -1 0];
        R2=[-sin(posq2) cos(posq2) 0; cos(posq2) sin(posq2) 0; 0 0 -1];
        R3=[sin(posq3) 0 -cos(posq3); -cos(posq3) 0 -sin(posq3); 0 1 0];
        R03=R0*R1*R2*R3;
        R36=R03'*R06;
        posq4=atan2(-R36(2,3),-R36(1,3));
        if q(5)>=0
            posq5=acos(-R36(3,3));
        else
```

```
            posq5=-acos(-R36(3,3));
        end
        posq6=atan2(R36(3,1),-R36(3,2));
        vel0=0;
        [~, qd(1)] = calllib('m5apiw32','PCube_getVel', dev, 1, vel0);
        [~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel0);
        [~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel0);
        [~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel0);
        [~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel0);
        [~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel0);
        maxDifAbs=abs(posq1-q(1));
        maxDif=posq1-q(1);
        velInicArticMasRapida=qd(1);
        if abs(posq2-q(2))>maxDifAbs
            maxDifAbs=abs(posq2-q(2));
            maxDif=posq2-q(2);
            velInicArticMasRapida=qd(2);
        end
        if abs(posq3-q(3))>maxDifAbs
            maxDifAbs=abs(posq3-q(3));
            maxDif=posq3-q(3);
            velInicArticMasRapida=qd(3);
        end
        if abs(posq4-q(4))>maxDifAbs
            maxDifAbs=abs(posq4-q(4));
            maxDif=posq4-q(4);
            velInicArticMasRapida=qd(4);
        end
        if abs(posq5-q(5))>maxDifAbs
            maxDifAbs=abs(posq5-q(5));
            maxDif=posq5-q(5);
            velInicArticMasRapida=qd(5);
        end
        if abs(posq6-q(6))>maxDifAbs
            maxDifAbs=abs(posq6-q(6));
            maxDif=posq6-q(6);
            velInicArticMasRapida=qd(6);
        end
        if maxDifAbs*180/pi>180
            errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                'ERROR');
            calllib('m5apiw32','PCube_haltAll',dev);
            calllib('m5apiw32','PCube_resetAll',dev);
            return
        end
        calllib ('m5apiw32', 'PCube_moveVel', dev, 1,...
            (posq1-q(1))/maxDifAbs*vel+(posq1-
q(1))/maxDif*velInicArticMasRapida-qd(1));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 2,...
            (posq2-q(2))/maxDifAbs*vel+(posq2-
q(2))/maxDif*velInicArticMasRapida-qd(2));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 3,...
            (posq3-q(3))/maxDifAbs*vel+(posq3-
q(3))/maxDif*velInicArticMasRapida-qd(3));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 4,...
            (posq4-q(4))/maxDifAbs*vel+(posq4-
q(4))/maxDif*velInicArticMasRapida-qd(4));
        calllib ('m5apiw32', 'PCube_moveVel', dev, 5,...
            (posq5-q(5))/maxDifAbs*vel+(posq5-
q(5))/maxDif*velInicArticMasRapida-qd(5));
```

```
        calllib ('m5apiw32', 'PCube_moveVel', dev, 6,...
            (posq6-q(6))/maxDifAbs*vel+(posq6-
q(6))/maxDif*velInicArticMasRapida-qd(6));
        while round(maxDif*180/pi)~=0 && buttondown==1
            [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
            set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
            [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
            set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
            [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
            set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
            [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
            set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
            [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
            set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
            [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
            set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
            posyrot(handles);
            if abs(round(q(1)*180/pi))>=round(maxposq1*180/pi) || ...
                    round(q(2)*180/pi)>=round(maxposq2*180/pi) || ...
                    round(q(2)*180/pi)<=round(minposq2*180/pi) || ...
                    abs(round(q(3)*180/pi))>=round(maxposq3*180/pi) || ...
                    abs(round(q(3)*180/pi))<=1 || ...
                    abs(round(q(4)*180/pi))>=round(maxposq4*180/pi) || ...
                    abs(round(q(5)*180/pi))>=round(maxposq5*180/pi) || ...
                    abs(round(q(5)*180/pi))<=5 || ...
                    abs(round(q(6)*180/pi))>=round(maxposq6*180/pi)
                errordlg('Esta es la mínima posición, no se puede avanzar
más.',...
                    'ERROR');
                calllib('m5apiw32','PCube_haltAll',dev);
                calllib('m5apiw32','PCube_resetAll',dev);
                return
            end
            pause(0.05);
        end
    end
end

guidata(hObject, handles);


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

%Con este botón se devolverán todas las articulaciones a su posición
%inicial (0°) una por una, empezando por la 1 y acabando con la 6
global movAutomatico;
movAutomatico=1;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
```

```
    case 2
        vel=0.1;
    case 3
        vel=0.5;
end
global dev q;
b=[handles.txtPosArt1 handles.txtPosArt2 handles.txtPosArt3...
    handles.txtPosArt4 handles.txtPosArt5 handles.txtPosArt6];
pos=0;
calllib('m5apiw32','PCube_resetAll',dev);
for i=1:6
    calllib ('m5apiw32', 'PCube_moveRamp', dev, i, 0, vel, 0.2);
    [~, q(i)] = calllib('m5apiw32','PCube_getPos', dev, i, pos);
    while (q(i)*180/pi)<=-0.02 || (q(i)*180/pi)>=0.02 %Para que se quede en
ese rango
        [~, q(i)] = calllib('m5apiw32','PCube_getPos', dev, i, pos);
        set(b(i),'String',[num2str(round(q(i)*180/pi,2)), '°']);
        posyrot(handles);
        pause(0.05);
    end
end
%Tras terminar se actualizarán todos los valores ya que habrán sufrido
%algún pequeño cambio al terminar el proceso
[~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
[~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
[~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
[~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
[~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
[~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
set(b(1),'String',[num2str(round(q(1)*180/pi,2)), '°']);
set(b(2),'String',[num2str(round(q(2)*180/pi,2)), '°']);
set(b(3),'String',[num2str(round(q(3)*180/pi,2)), '°']);
set(b(4),'String',[num2str(round(q(4)*180/pi,2)), '°']);
set(b(5),'String',[num2str(round(q(5)*180/pi,2)), '°']);
set(b(6),'String',[num2str(round(q(6)*180/pi,2)), '°']);
posyrot(handles);
movAutomatico=0;
guidata(hObject, handles);


function posyrot(handles)
%Con esta función se actualizan tanto las posiciones x y z como la matriz
%de rotación, tomando el valor del vector q (posición de cada articuación)
%cada vez que se invoca. También se actualizarán los valores de las
%componentes de la velocidad del extremo del robot
global q Pm T dev;
%Matriz de transformación homogénea del robot
A0=[0 0 -1 0; -1 0 0 0; 0 1 0 0; 0 0 0 1]; %Para relacionar ejes principales
con ejes 0
A1=[sin(q(1)) 0 cos(q(1)) 0; -cos(q(1)) 0 sin(q(1)) 0; 0 -1 0 -241.1; 0 0 0
1];
A2=[-sin(q(2)) cos(q(2)) 0 -495*sin(q(2)); cos(q(2)) sin(q(2)) 0 ...
    495*cos(q(2)); 0 0 -1 0; 0 0 0 1];
A3=[sin(q(3)) 0 -cos(q(3)) 0; -cos(q(3)) 0 -sin(q(3)) 0; 0 1 0 0; 0 0 0 1];
A4=[cos(q(4)) 0 -sin(q(4)) 0; sin(q(4)) 0 cos(q(4)) 0; 0 -1 0 -372; 0 0 0 1];
A5=[cos(q(5)) 0 sin(q(5)) 0; sin(q(5)) 0 -cos(q(5)) 0; 0 1 0 0; 0 0 0 1];
A6=[-sin(q(6)) cos(q(6)) 0 0;cos(q(6))  sin(q(6)) 0 0; 0 0 -1 -183.2; 0 0 0
1];
T=A0*A1*A2*A3*A4*A5*A6;
%Posición de la muñeca del robot
Pm=T*[0;0;-183.2;1];
```

```
%Posiciones x y z
%Mostrar posiciones del extremo
set(handles.txtPosXExtremoRobot,'String',[num2str(round(T(1,4),2)), 'mm']);
set(handles.txtPosYExtremoRobot,'String',[num2str(round(T(2,4),2)), 'mm']);
set(handles.txtPosZExtremoRobot,'String',[num2str(round(T(3,4),2)), 'mm']);
%Mostrar posiciones de la muñeca
% set(handles.txtPosXExtremoRobot,'String',[num2str(round(Pm(1),2)), 'mm']);
% set(handles.txtPosYExtremoRobot,'String',[num2str(round(Pm(2),2)), 'mm']);
% set(handles.txtPosZExtremoRobot,'String',[num2str(round(Pm(3),2)), 'mm']);

%Matriz de rotación
set(handles.txtMatRotX6X,'String',num2str(round(T(1,1),2)));
set(handles.txtMatRotX6Y,'String',num2str(round(T(2,1),2)));
set(handles.txtMatRotX6Z,'String',num2str(round(T(3,1),2)));
set(handles.txtMatRotY6X,'String',num2str(round(T(1,2),2)));
set(handles.txtMatRotY6Y,'String',num2str(round(T(2,2),2)));
set(handles.txtMatRotY6Z,'String',num2str(round(T(3,2),2)));
set(handles.txtMatRotZ6X,'String',num2str(round(T(1,3),2)));
set(handles.txtMatRotZ6Y,'String',num2str(round(T(2,3),2)));
set(handles.txtMatRotZ6Z,'String',num2str(round(T(3,3),2)));

%Jacobiana directa de vlocidad lineal
Jv=[0,372*cos(q(2))*sin(q(3))-495*sin(q(2))-372*cos(q(3))*sin(q(2))+...
    (916*cos(q(5))*(cos(q(2))*sin(q(3))-cos(q(3))*sin(q(2))))/5+...
    (916*cos(q(4))*sin(q(5))*(sin(q(2))*sin(q(3)) +
cos(q(2))*cos(q(3))))/5,...
    372*cos(q(3))*sin(q(2))-372*cos(q(2))*sin(q(3))-
(916*cos(q(5))*(cos(q(2))*sin(q(3))...
    -cos(q(3))*sin(q(2))))/5-
(916*cos(q(4))*sin(q(5))*(sin(q(2))*sin(q(3))+cos(q(2))*cos(q(3))))/5,...
    (916*sin(q(4))*sin(q(5))*(cos(q(2))*sin(q(3))-cos(q(3))*sin(q(2))))/5,...
    -(916*sin(q(5))*(sin(q(2))*sin(q(3)) + cos(q(2))*cos(q(3))))/5-...
    (916*cos(q(4))*cos(q(5))*(cos(q(2))*sin(q(3))-
cos(q(3))*sin(q(2))))/5,0;... %Fin primera fila
    (916*sin(q(5))*(sin(q(1))*sin(q(4))-
cos(q(4))*(cos(q(1))*cos(q(2))*cos(q(3))+...
    cos(q(1))*sin(q(2))*sin(q(3)))))/5-
(916*cos(q(5))*(cos(q(1))*cos(q(2))*sin(q(3))-...
    cos(q(1))*cos(q(3))*sin(q(2))))/5 + 495*cos(q(1))*sin(q(2))-
372*cos(q(1))*cos(q(2))*sin(q(3))+...
    372*cos(q(1))*cos(q(3))*sin(q(2)),...

(916*cos(q(5))*(cos(q(2))*cos(q(3))*sin(q(1))+sin(q(1))*sin(q(2))*sin(q(3))))
/5+...
    495*cos(q(2))*sin(q(1))-
(916*cos(q(4))*sin(q(5))*(cos(q(2))*sin(q(1))*sin(q(3))-...
    cos(q(3))*sin(q(1))*sin(q(2))))/5+372*cos(q(2))*cos(q(3))*sin(q(1))+...
    372*sin(q(1))*sin(q(2))*sin(q(3)),...
    (916*cos(q(4))*sin(q(5))*(cos(q(2))*sin(q(1))*sin(q(3))-
cos(q(3))*sin(q(1))*sin(q(2))))/5-...

(916*cos(q(5))*(cos(q(2))*cos(q(3))*sin(q(1))+sin(q(1))*sin(q(2))*sin(q(3))))
/5-...
    372*cos(q(2))*cos(q(3))*sin(q(1)) - 372*sin(q(1))*sin(q(2))*sin(q(3)),...

(916*sin(q(5))*(sin(q(4))*(cos(q(2))*cos(q(3))*sin(q(1))+sin(q(1))*sin(q(2))*
sin(q(3)))-...
    cos(q(1))*cos(q(4))))/5,...
    (916*sin(q(5))*(cos(q(2))*sin(q(1))*sin(q(3))-
cos(q(3))*sin(q(1))*sin(q(2))))/5-...
```

```
(916*cos(q(5))*(cos(q(1))*sin(q(4))+cos(q(4))*(cos(q(2))*cos(q(3))*sin(q(1))+
...
    sin(q(1))*sin(q(2))*sin(q(3)))))/5,0;... %Fin segunda fila
    (916*cos(q(5))*(cos(q(2))*sin(q(1))*sin(q(3))-
cos(q(3))*sin(q(1))*sin(q(2))))/5-...
    495*sin(q(1))*sin(q(2))+(916*sin(q(5))*(cos(q(1))*sin(q(4))+...

cos(q(4))*(cos(q(2))*cos(q(3))*sin(q(1))+sin(q(1))*sin(q(2))*sin(q(3)))))/5+.
..
    372*cos(q(2))*sin(q(1))*sin(q(3))-372*cos(q(3))*sin(q(1))*sin(q(2)),...

(916*cos(q(5))*(cos(q(1))*cos(q(2))*cos(q(3))+cos(q(1))*sin(q(2))*sin(q(3))))
/5+...

495*cos(q(1))*cos(q(2))+372*cos(q(1))*cos(q(2))*cos(q(3))+372*cos(q(1))*sin(q
(2))*sin(q(3))-...
    (916*cos(q(4))*sin(q(5))*(cos(q(1))*cos(q(2))*sin(q(3))-
cos(q(1))*cos(q(3))*sin(q(2))))/5,...
    (916*cos(q(4))*sin(q(5))*(cos(q(1))*cos(q(2))*sin(q(3))-
cos(q(1))*cos(q(3))*sin(q(2))))/5-...
    372*cos(q(1))*cos(q(2))*cos(q(3))-372*cos(q(1))*sin(q(2))*sin(q(3))-...

(916*cos(q(5))*(cos(q(1))*cos(q(2))*cos(q(3))+cos(q(1))*sin(q(2))*sin(q(3))))
/5,...

(916*sin(q(5))*(cos(q(4))*sin(q(1))+sin(q(4))*(cos(q(1))*cos(q(2))*cos(q(3))+
...
    cos(q(1))*sin(q(2))*sin(q(3)))))/5,...
    (916*sin(q(5))*(cos(q(1))*cos(q(2))*sin(q(3))-
cos(q(1))*cos(q(3))*sin(q(2))))/5+...
    (916*cos(q(5))*(sin(q(1))*sin(q(4))-
cos(q(4))*(cos(q(1))*cos(q(2))*cos(q(3))+...
    cos(q(1))*sin(q(2))*sin(q(3)))))/5,0];%Fin última fila

%Jacobiana directa de velocidad angular
Jw=[0,...
    0,-sin(q(2))*sin(q(3))-cos(q(2))*cos(q(3)),...
    -sin(q(4))*(cos(q(2))*sin(q(3))-cos(q(3))*sin(q(2))),...
    cos(q(4))*sin(q(5))*(cos(q(2))*sin(q(3))-cos(q(3))*sin(q(2)))-...
    cos(q(5))*(sin(q(2))*sin(q(3))+cos(q(2))*cos(q(3))),...
    cos(q(5))*(sin(q(2))*sin(q(3))+cos(q(2))*cos(q(3)))-...
    cos(q(4))*sin(q(5))*(cos(q(2))*sin(q(3))-cos(q(3))*sin(q(2)));...%Fin
primera fila
    -cos(q(1)),...
    cos(q(1)),...
    cos(q(2))*sin(q(1))*sin(q(3))-cos(q(3))*sin(q(1))*sin(q(2)),...
    cos(q(1))*cos(q(4))-sin(q(4))*(cos(q(2))*cos(q(3))*sin(q(1))+...
    sin(q(1))*sin(q(2))*sin(q(3))),...
    cos(q(5))*(cos(q(2))*sin(q(1))*sin(q(3))-
cos(q(3))*sin(q(1))*sin(q(2)))+...
    sin(q(5))*(cos(q(4))*(cos(q(2))*cos(q(3))*sin(q(1))+...
    sin(q(1))*sin(q(2))*sin(q(3)))+cos(q(1))*sin(q(4))),...
    -cos(q(5))*(cos(q(2))*sin(q(1))*sin(q(3)) -...
    cos(q(3))*sin(q(1))*sin(q(2)))-...
    sin(q(5))*(cos(q(4))*(cos(q(2))*cos(q(3))*sin(q(1))+...
    sin(q(1))*sin(q(2))*sin(q(3)))+cos(q(1))*sin(q(4)));...%Fin segunda fila
    sin(q(1)),...
    -sin(q(1)),...
    cos(q(1))*cos(q(2))*sin(q(3))-cos(q(1))*cos(q(3))*sin(q(2)),...
    -sin(q(4))*(cos(q(1))*cos(q(2))*cos(q(3))+...
```

```
    cos(q(1))*sin(q(2))*sin(q(3)))-cos(q(4))*sin(q(1)),...
    cos(q(5))*(cos(q(1))*cos(q(2))*sin(q(3))-...
    cos(q(1))*cos(q(3))*sin(q(2)))+....
    sin(q(5))*(cos(q(4))*(cos(q(1))*cos(q(2))*cos(q(3))+...
    cos(q(1))*sin(q(2))*sin(q(3)))-sin(q(1))*sin(q(4))),...
    -cos(q(5))*(cos(q(1))*cos(q(2))*sin(q(3))-...
    cos(q(1))*cos(q(3))*sin(q(2)))-...
    sin(q(5))*(cos(q(4))*(cos(q(1))*cos(q(2))*cos(q(3))+...
    cos(q(1))*sin(q(2))*sin(q(3)))-sin(q(1))*sin(q(4)))];%Fin última fila

%Lectura de la velocidad de cada articulación
vel=0;
[~, qd(1)] = calllib('m5apiw32','PCube_getVel', dev, 1, vel);
[~, qd(2)] = calllib('m5apiw32','PCube_getVel', dev, 2, vel);
[~, qd(3)] = calllib('m5apiw32','PCube_getVel', dev, 3, vel);
[~, qd(4)] = calllib('m5apiw32','PCube_getVel', dev, 4, vel);
[~, qd(5)] = calllib('m5apiw32','PCube_getVel', dev, 5, vel);
[~, qd(6)] = calllib('m5apiw32','PCube_getVel', dev, 6, vel);

%Componentes de la velocidad del extremo del robot
J1=Jv*qd';
J2=Jw*qd';
set(handles.txtVelXExtremoRobot,'String',[num2str(round(J1(1),2)),' mm/s']);
set(handles.txtVelYExtremoRobot,'String',[num2str(round(J1(2),2)),' mm/s']);
set(handles.txtVelZExtremoRobot,'String',[num2str(round(J1(3),2)),' mm/s']);
set(handles.txtVelPhiExtremoRobot,'String',[num2str(round(J2(1),2)),'
rad/s']);
set(handles.txtVelThetaExtremoRobot,'String',[num2str(round(J2(2),2)),'
rad/s']);
set(handles.txtVelPsiExtremoRobot,'String',[num2str(round(J2(3),2)),'
rad/s']);


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

%Al pulsar este botón se guarda la posición en la que está el robot para
%que, posteriormente, se mueva (en orden) por todas las posiciones guardadas
%En c se guardarán las distintas posiciones de la articulación 1, en d las
%de la articulación 2, y así sucesivamente
global c d e f g h cont dev;
cont=cont+1;
set(handles.lblNumPosGuardadas,'String',['N° de posiciones guardadas:
',num2str(cont)]);
pos=0;
[~, c(cont)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
[~, d(cont)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
[~, e(cont)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
[~, f(cont)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
[~, g(cont)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
[~, h(cont)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);


% --- Executes on button press in btnMoverRobot.
function btnMoverRobot_Callback(hObject, eventdata, handles)
% hObject    handle to btnMoverRobot (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

%Al pulsar este botón, el robot se mueve por todas las posiciones guardadas
global movAutomatico;
movAutomatico=1;
pos=0;
val=get(handles.pmnSeleccionVelocidad,'Value');
switch val
    case 1
        vel=0.05;
    case 2
        vel=0.1;
    case 3
        vel=0.5;
end
global c d e f g h cont dev q;
calllib('m5apiw32','PCube_resetAll',dev);
for i=1:cont
    %Se declara la variable maxDif, en la que se guarda la maxima diferencia
    %entre la posición a la que tiene que llegar y la posicion en la que se
    %encuentra cada articulación, dándole principalmente el valor arbitrario
    %de c(i)-q(1) en valor absoluto. Esto hará que la máxima velocidad sea
    %la elegida en el pop-up menu
    maxDif=abs(c(i)-q(1));
    if abs(d(i)-q(2))>maxDif
        maxDif=abs(d(i)-q(2));
    end
    if abs(e(i)-q(3))>maxDif
        maxDif=abs(e(i)-q(3));
    end
    if abs(f(i)-q(4))>maxDif
        maxDif=abs(f(i)-q(4));
    end
    if abs(g(i)-q(5))>maxDif
        maxDif=abs(g(i)-q(5));
    end
    if abs(h(i)-q(6))>maxDif
        maxDif=abs(h(i)-q(6));
    end
    %La velocidad especificada sale de la relación de velocidades angulares
    %hallada entre la máxima velocidad (la del pop-up menu) y la velocidad
    %de cada articulación. Así, la articulación que tenga la diferencia de
    %posiciones más grande coincidirá con la máxima velocidad
    calllib ('m5apiw32', 'PCube_moveRamp', dev, 1, c(i), vel*abs(c(i)-
q(1))/maxDif, 0.2);
    calllib ('m5apiw32', 'PCube_moveRamp', dev, 2, d(i), vel*abs(d(i)-
q(2))/maxDif, 0.2);
    calllib ('m5apiw32', 'PCube_moveRamp', dev, 3, e(i), vel*abs(e(i)-
q(3))/maxDif, 0.2);
    calllib ('m5apiw32', 'PCube_moveRamp', dev, 4, f(i), vel*abs(f(i)-
q(4))/maxDif, 0.2);
    calllib ('m5apiw32', 'PCube_moveRamp', dev, 5, g(i), vel*abs(g(i)-
q(5))/maxDif, 0.2);
    calllib ('m5apiw32', 'PCube_moveRamp', dev, 6, h(i), vel*abs(h(i)-
q(6))/maxDif, 0.2);
    %Hasta que no coincida la posición de cada articulación con la posición
    %a la que tienen que llegar estas, no se pasará a mover el robot a la
    %siguiente posición guardada
    while round(q(1)*180/pi)~=round(c(i)*180/pi) || ...
            round(q(2)*180/pi)~=round(d(i)*180/pi) || ...
            round(q(3)*180/pi)~=round(e(i)*180/pi) || ...
```

```
            round(q(4)*180/pi)~=round(f(i)*180/pi) ||  ...
            round(q(5)*180/pi)~=round(g(i)*180/pi) || ...
            round(q(6)*180/pi)~=round(h(i)*180/pi)
        [~, q(1)] = calllib('m5apiw32','PCube_getPos', dev, 1, pos);
        [~, q(2)] = calllib('m5apiw32','PCube_getPos', dev, 2, pos);
        [~, q(3)] = calllib('m5apiw32','PCube_getPos', dev, 3, pos);
        [~, q(4)] = calllib('m5apiw32','PCube_getPos', dev, 4, pos);
        [~, q(5)] = calllib('m5apiw32','PCube_getPos', dev, 5, pos);
        [~, q(6)] = calllib('m5apiw32','PCube_getPos', dev, 6, pos);
        set(handles.txtPosArt1,'String',[num2str(round(q(1)*180/pi,2)),
'°']);
        set(handles.txtPosArt2,'String',[num2str(round(q(2)*180/pi,2)),
'°']);
        set(handles.txtPosArt3,'String',[num2str(round(q(3)*180/pi,2)),
'°']);
        set(handles.txtPosArt4,'String',[num2str(round(q(4)*180/pi,2)),
'°']);
        set(handles.txtPosArt5,'String',[num2str(round(q(5)*180/pi,2)),
'°']);
        set(handles.txtPosArt6,'String',[num2str(round(q(6)*180/pi,2)),
'°']);
        posyrot(handles);
        pause(0.05);
    end
    %En cuanto el robot llegue a la posicion guardada, el robot se parará
    %y, por tanto, la velocidad de su extremo será nula
    set(handles.txtVelXExtremoRobot,'String','0 mm/s');
    set(handles.txtVelYExtremoRobot,'String','0 mm/s');
    set(handles.txtVelZExtremoRobot,'String','0 mm/s');
    %El robot se mantiene 1 segundo en la posición sin moverse antes de
    %pasar a la siguiente posición guardada
    pause(1);
end
posyrot(handles);
movAutomatico=0;


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

%Al pulsar este botón se borran todas las posiciones guardadas para volver
%a empezar a guardar valores desde 0
global cont;
cont=0;
%También se actualiza el número de posiciones guardadas
set(handles.lblNumPosGuardadas,'String',['N° de posiciones guardadas: '...
    ,num2str(cont)]);


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

%Este botón simplemente invocará a la ventana de las posiciones guardadas
PosicionesGuardadas;
```

```
% --- Executes on mouse motion over figure - except title and menu.
function figure1_WindowButtonMotionFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global cont;
%Nada más moverse el ratón tras cerrar el GUI de PosicionesGuardadas se
%actualizará el valor de las posiciones guardadas si ha sido modificado
set(handles.lblNumPosGuardadas,'String',['N° de posiciones guardadas: '...
    ,num2str(cont)]);
```

Capítulo 10 : Bibliografía

[1] MCKERROW, Phillip John (1991). *Introduction to Robotics*. Massachusetts: Addison-Wesley.

[2] RENTERIA, Arantxa; RIVAS, María (2000). *Robótica Industrial. Fundamentos y aplicaciones*. Madrid: McGraw-Hill.

[3] BARRIENTOS, Antonio (2007). *Fundamentos de Robótica*. Madrid: McGraw-Hill.

[4] Manuales del brazo modular Robotnik. 2008-2009.

[5] Schunk. *Programmers guide for PowerCube*. 2007.

[6] http://encoder.com/blog/encoder-basics/que-es-un-encoder/

[7] https://www.dspace.espol.edu.ec/bitstream/123456789/10740/11/MATLAB_GUIDE.pdf

[8] https://ocw.upc.edu/sites/all/modules/ocw/estadistiques/download.php?file=51427/2011/1/54511/tema_4_interficie_grafica_con_el_usuario-5154.pdf

[9] http://www.cs.cornell.edu/courses/cs1115/2012fa/GUIDocs/buildgui.pdf

[10] http://www.ehu.eus/izaballa/Ana_Matr/Matlab/guia.pdf

[11] https://es.mathworks.com/products/symbolic/features.html#key-features

[12] https://stackoverflow.com/questions/44342930/matlab-guide-adding-deleting-items-from-listbox

[13] https://es.slideshare.net/morones.om/guia-rapida-de-matlab-comandos-basicos-graficacion-y-programacion

[14] http://mirror.umd.edu/roswiki/doc/fuerte/api/schunk_libm5api/html/m5apiw32_8h_source.html

[15] http://docs.ros.org/diamondback/api/libm5api/html/globals_func.html