UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Desarrollo de una interfaz para el control del robot IRB120 desde Matlab"

Azahara Gutiérrez Corbacho

2014

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Desarrollo de una interfaz para el control del robot IRB120 desde Matlab"

Autor:	Azahara Gutiérrez Corbacho
Universidad:	Universidad de Alcalá
País:	España
Profesor Tutor:	María Elena López Guillén

Presidente: Dª. Marta Marrón Romera

Vocal 1: D. Manuel Ureña Molina

Vocal 2: Dª. María Elena López Guillén

CALIFICACIÓN:.....

FECHA:.....

Agradecimientos

Me gustaría dar las gracias a mis compañeros y profesores por la motivación y el respaldo recibido durante estos años, y en especial a mi tutora Elena.

También quiero agradecer a mis padres y a mi hermano todo el apoyo que me han dado en esta etapa de mi vida y el que hayan estado siempre animándome a seguir hacia delante y a luchar por lo que de verdad importa.

A mi novio David, por todo lo que hace por mí cada día, agradecerle todo su cariño y apoyo durante estos años. Sin él no habría sido posible llegar hasta aquí.

No me gustaría acabar sin mencionar una persona que desgraciadamente no se encuentra ya entre nosotros, abuelo, sé que estés donde estés eres partícipe de todo esto y nunca te olvidaremos.

Contenido general

Lista de figuras

Lista de tablas

Resumen

El objetivo de este proyecto es realizar la comunicación con el brazo robótico, IRB120 de ABB, a través de la herramienta de software matemático Matlab.

Para ello desarrollaremos un socket de comunicación, que se encargará enviar y procesar los datos. Para comprobar que la comunicación funciona y que el envío de datos se realiza correctamente, se implementarán en Matlab, una serie de interfaces de comunicación con el robot y una aplicación final.

La primera, será una interfaz gráfica realizada a través de la herramienta GUIDE. El diseño de la segunda interfaz será a través de la creación de clases en Matlab, que posteriormente utilizaremos para desarrollar la aplicación final. Para la tercera interfaz de comunicación, haremos uso de la herramienta Simulink.

Por último, diseñaremos una aplicación final basada en las interfaces creadas previamente y que consistirá en el trazado de números con el brazo robot.

Abstract

The objective of this project is to establish a reliable communication network for the ABB-IRB 120 industrial robot, through Matlab mathematical software.

Therefore, we will develop a socket with TCP/IP support for send and receive data. To Check this, we'll design, in Matlab, several interfaces for connect the robot with any remote control point.

The first interface will be performed through the GUIDE tool. The second one will be designed by developing classes. The latter, will be used later to create the final application. For the last communication interface, we will use Simulink tool to develop it.

Finally, we design our final application that based in previously created interfaces and will treat in plotting numbers with ABB industrial robot.

1 Introducción

1.1 Antecedentes

Los sistemas de producción modernos, son capaces de realizar operaciones industriales complejas con una mínima intervención manual. También pueden llegar a sustituir a los hombres en los procesos rutinarios como el embalaje o la manipulación.

Como la mayor parte de la industria en la actualidad, se basa en la fabricación y en el transporte de los productos fabricados, la tecnología que se aplica a estos procesos y más concretamente la robótica, minimiza sus costes de producción al plantear un sistema de trabajo continuo y con un reducido consumo de energía.

Los inicios más directos de los robots fueron los telemanipuladores que manipulaban elementos radiactivos con un dispositivo maestro-esclavo; el maestro lo manejaba directamente el operador y el esclavo que estaba directamente en contacto con el producto radioactivo y unido mecánicamente al maestro, era movido directamente por aquél.

La evolución de los robots industriales desde sus orígenes ha sido vertiginosa. La investigación y desarrollo sobre robótica industrial ha permitido que los robots tomen posiciones en casi todas las áreas productivas y tipos de industria.

Los futuros desarrollos de la robótica apuntan a aumentar su movilidad, destreza y autonomía de sus acciones. La mayor parte de los robots industriales actuales son con base estática, aunque ya existen otros tipos de aplicaciones que han hecho evolucionar tanto la concepción como su propia morfología. Entre los robots dedicados a aplicaciones no industriales destacan los robots espaciales, para aplicaciones submarinas y subterráneas, robots militares, aplicaciones agrícolas y robots móviles de servicios.

1.2 Objetivos del proyecto

Como ya se ha comentado antes, la robótica tiene un papel muy importante dentro de la ingeniería en general y la industria en particular. Es por lo que las asignaturas que

abarcan estos conocimientos han sido incluidas en los nuevos Planes de Estudio de Grado.

En nuestro caso es la asignatura de Sistemas Robotizados, la que nos proporciona un amplio estudio de la morfología y sistemas de percepción de los brazos robots industriales, al igual que el diseño de sistemas de control cinemático y dinámico para los mismos y la programación de este tipo de sistemas robóticos en aplicaciones de automatización industrial. Otro aspecto a destacar de esta asignatura es el laboratorio, debido a la posibilidad de trabajar e interactuar con un robot real: el robot irb120, proporcionado por la empresa ABB.

Es por tanto en este punto, cuando surgen distintas necesidades en cuanto a procesos de comunicación con el brazo robot. Los profesores del departamento de Electrónica plantearon diferentes maneras (a través de diversas plataformas) de llevar a cabo esa conexión. Por ejemplo, con RobotStudio, con leguaje Rapid o con leguaje C, mediante Matlab o mediante ROS.

El objetivo principal de este proyecto es realizar la comunicación con el robot a través de la herramienta de software matemático Matlab, para realizar un socket de comunicación entre el servidor y el cliente. En la figura siguiente podemos observar de forma esquematizada el objetivo a lograr en este proyecto.

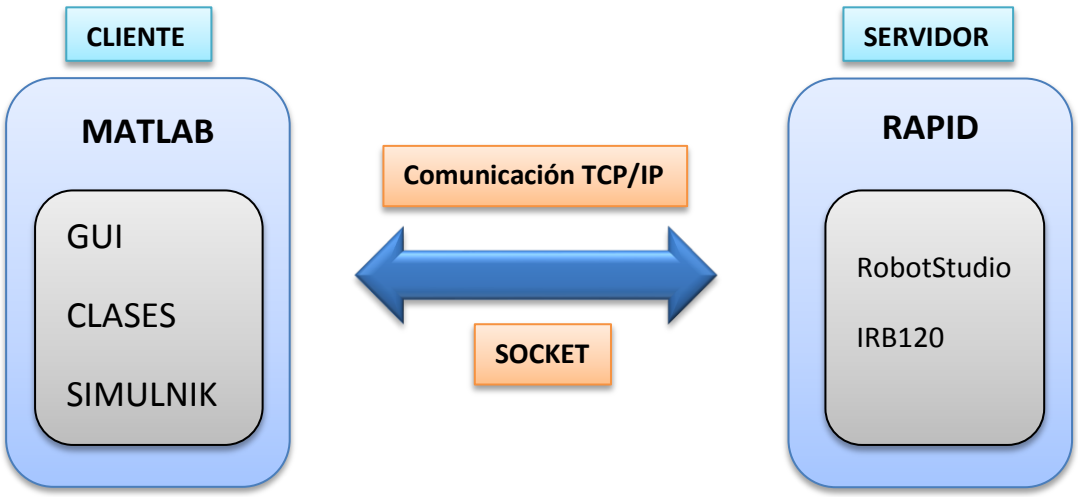


FIGURA 1.1 Esquema básico de los bloques principales del proyecto

Por un lado tenemos el servidor, que es un programa realizado en RAPID (lenguaje específico del software que proporciona la empresa ABB) que ya ha sido desarrollado en otro proyecto de título: "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station" [1], realizado por Marek Jerzy Frydrysiak.

Por otro lado tenemos el bloque correspondiente al cliente, implementado en Matlab. Para realizar la conexión del cliente con el servidor debemos crear un socket de comunicación, para lo que utilizaremos los protocolos de comunicación TCP/IP.

Tras conseguir la conexión entre cliente y servidor, desarrollaremos varias interfaces de comunicación a través de diversas herramientas de Matlab (GUIDE, clases y Simulink) que permitan realizar al usuario un control del robot desde una plataforma externa a la que proporciona la empresa ABB.

Por último, se llevará a cabo una aplicación final, que utilizará las interfaces anteriores y se encargará de pintar los números del cero al nueve, según requiera el usuario.

1.3 Estructura de la memoria

En este apartado explicaremos a grosso modo cómo está estructurada la memoria de este trabajo.

En el punto número uno tenemos la introducción, en la que encontraremos los antecedentes y los objetivos del proyecto.

El apartado número dos es el encargado de explicar a fondo cada una de las herramientas que se van a utilizar: Robot industrial ABB, socket de comunicación y la herramienta de software matemático, Matlab.

El desarrollo de las interfaces de comunicación lo encontramos en el apartado número tres, en el que tras una visión general de las tres interfaces desarrolladas, abordaremos cada una de ellas en profundidad.

En el punto número cuatro se implementa la aplicación final de trazado de números. En este apartado se incluyen resultados en Matlab, en simulación (con RobotStudio) y con el Robot real.

Para situar al lector, a continuación observaremos un esquema que muestra una vista general de los contenidos y apartados generales de este Trabajo Fin de Grado.

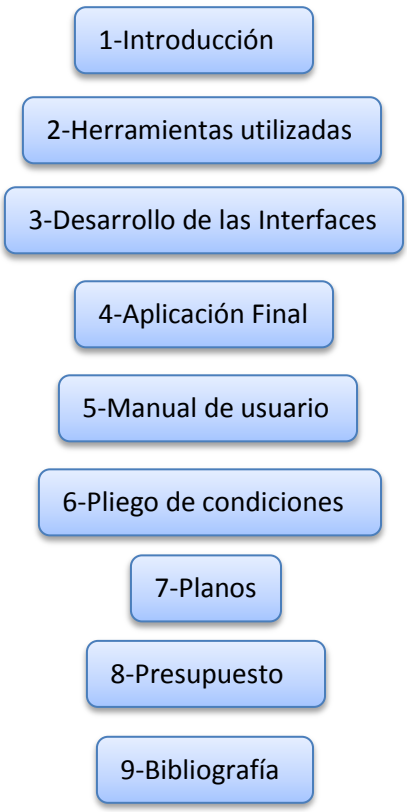


FIGURA 1.2 Esquema de la estructura de la memoria

2 Herramientas utilizadas

En este punto, se realizará un breve resumen acerca de las herramientas, tanto software como hardware, utilizadas para la realización del proyecto.

2.1 Robot Industrial de ABB

ABB es una empresa líder mundial en ingeniería eléctrica y automatización. La compañía es el producto de la unión en 1988 de Asea y BBC. Actualmente la sede central se encuentra en Zúrich (Suiza).

Tiene una gran oferta industrial en la que se pueden diferenciar cinco divisiones: Power Products, Power Systems, Discrete Automation and Motion y Low Voltage Products.

En estas divisiones, se ofrecen desde interruptores de iluminación, hasta robots industriales, grandes transformadores eléctricos o sistemas de control capaces de gestionar grandes redes eléctricas o industrias.

Como ya se ha mencionado antes, ABB ofrece a sus clientes robots industriales con los que se mejora la productividad de las empresas, cuenta con una oferta de más de 25 tipos de robots distintos para las diferentes utilidades que se le puede dar en cada tipo de industria. Este gran abanico de posibilidades la convierte en la empresa líder de suministro de robots en el mundo.

FIGURA 2.1 Robots de ABB en cadena de montaje

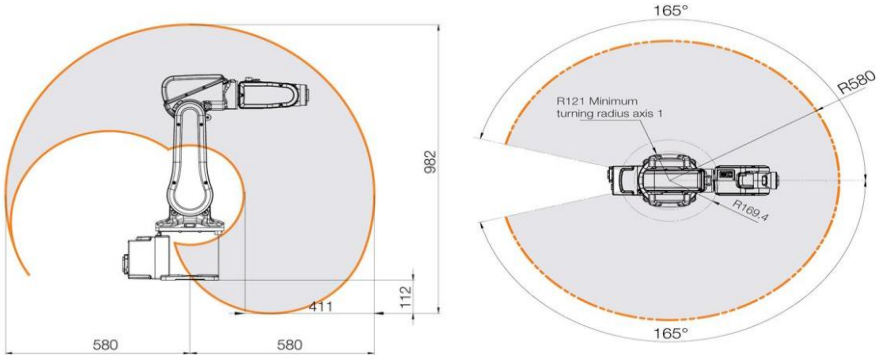
2.1.1 Brazo robótico IRB120

El brazo robótico IRB120 es el robot de ABB más pequeño y muy útil para muchas aplicaciones ya que pesa solamente 25 kg y puede manipular hasta 3 kg (4 kg para la muñeca en posición vertical), con un área de trabajo de 580mm. A pesar de su tamaño, para nuestra aplicación es más que suficiente; además posee 6 ejes, los tres primeros servirán para establecer la posición del efector final y los tres últimos para determinar la orientación del mismo.

En la primera figura observamos el robot IRB120 posicionado en diferentes planos (anclado al suelo, a una pared o al techo) en función de las necesidades de la aplicación que se vaya a desarrollar.

En la segunda figura podemos observar el área de trabajo de la muñeca con sus correspondientes dimensiones, lo que es muy útil a la hora de elegir qué robot se adapta mejor a nuestras necesidades.

FIGURA 2.2 Brazo Robótico IRB120



FIGURAS 2.3 Y 2.4 Área de trabajo del centro de la muñeca (eje 5)

2.1.2 Software RobotStudio

RobotStudio es el software de simulación y programación offline de ABB, que permite crear y simular estaciones robóticas industriales en 3D en un ordenador. Sus características aumentan la variedad de tareas posibles para llevar a cabo mediante un sistema robótico, como la capacitación, la programación o la optimización.

Este software aporta herramientas que aumentan la rentabilidad del sistema de robots, pues permite realizar tareas tales como programación y optimización de programas sin alterar la producción. Esto añade muchas ventajas, entre ellas:

- Reducción de riesgos
- Arranques más rápidos
- Menor tiempo para modificaciones
- Aumento de la productividad

RobotStudio está basado en el controlador virtual de ABB, que es una copia simulada del software real que utilizan los robots de ABB. Gracias a este software, se puede ejecutar en nuestro ordenador un sistema robótico que previamente haya sido diseñado antes de ser ejecutado en el robot real, con lo que se evitan costes innecesarios.

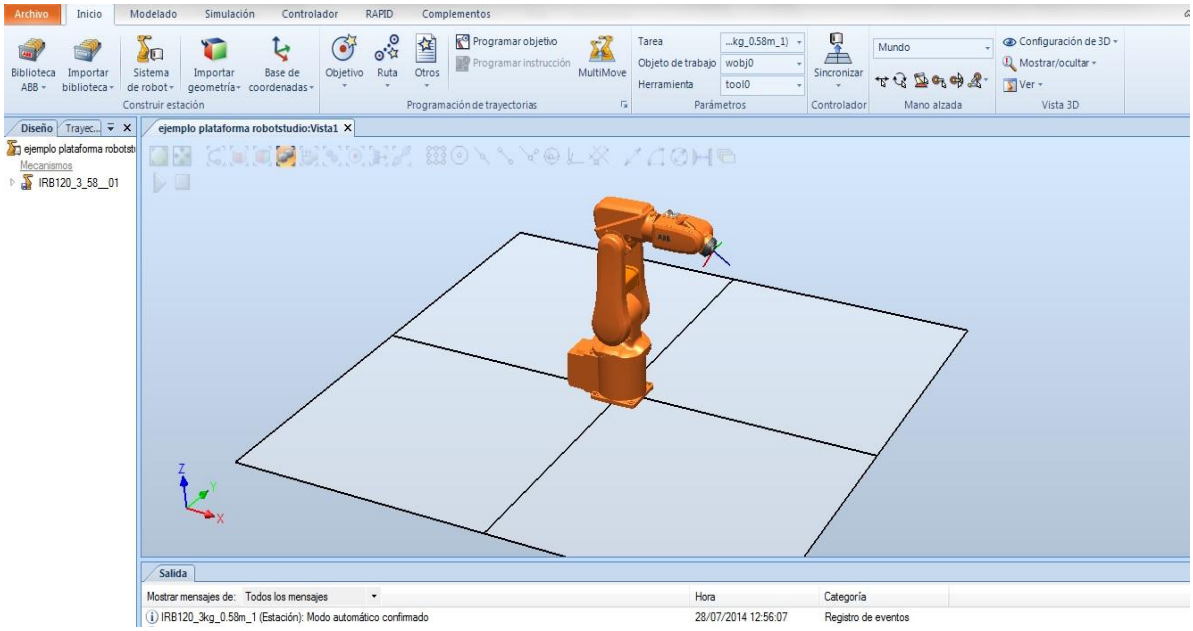


FIGURA 2.5 Interfaz principal de RobotStudio

El software de RobotStudio nos permite utilizar diversas herramientas. A continuación se mencionan algunas de las más importantes:

- ***Importar CAD***

RobotStudio nos permite importar fácilmente datos de la mayoría de formatos CAD incluyendo IGES, STEP, VRML, VDAFS, ACIS y CATIA. Al trabajar con este tipo de datos, el sistema permite al programador generar programas mucho más precisos.

- ***Tablas de Eventos***

Las tablas de eventos nos permiten verificar la estructura del programa y su lógica. Cuando ejecutamos el programa, se pueden visualizar los estados de las E/S desde la estación de trabajo.

- ***Detección de Colisión***

La detección de colisión es una herramienta más orientada a entornos reales de funcionamiento. Se utiliza para prevenir las posibles colisiones del robot con sus alrededores durante la ejecución del programa, debido por ejemplo, a algún fallo de programación.

- ***Visual Basic for Applications (VBA)***

El uso de Visual Basic for Applications nos permite crear y diseñar interfaces que faciliten al usuario la utilización de la aplicación.

- ***Carga y Descarga real***

Una importante característica que presenta el software de RobotStudio es que todos los programas creados se pueden cargar fácilmente al sistema real, sin que sea necesario utilizar medios externos adicionales.

2.1.3 Lenguaje de programación RAPID

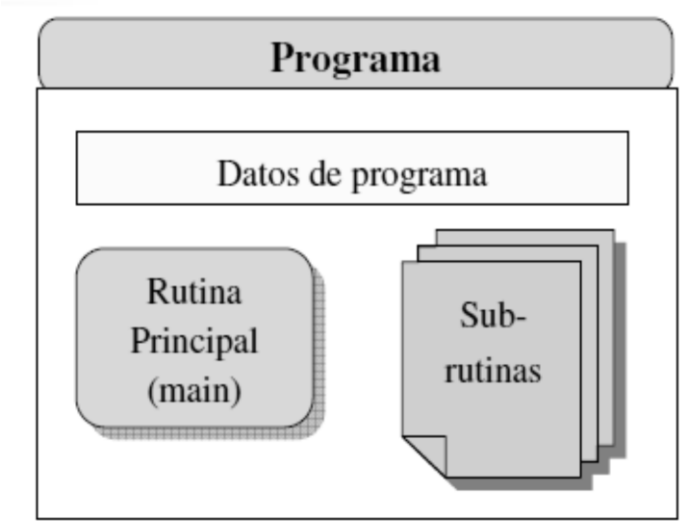
RAPID es un lenguaje de programación de alto nivel diseñado por la compañía ABB para el control de los robots industriales. Este lenguaje, proporciona un conjunto considerable de consignas (como funciones o rutinas), aritmética y expresiones lógicas, gestión de errores, multitarea, etc, por lo que se está utilizando con gran éxito para manejar algunas operaciones industriales complicadas y complejas en el ámbito robótico.

Una aplicación RAPID consta de un programa y una serie de módulos del sistema.

Podemos decir por tanto que básicamente, un programa de RAPID contiene un conjunto de instrucciones que describen cómo funciona un robot. Estas últimas en su mayoría, poseen una serie de argumentos que son utilizados para definir un movimiento del brazo robot.

Generalizando, podríamos deducir, que el programa es una secuencia de instrucciones que controlan el robot y que en general consta de tres partes: rutina principal, sub-rutinas y los datos del programa.

En la figura siguiente observamos la estructura básica de un programa desarrollado en lenguaje RAPID en el que localizamos cada una de las partes que acabamos de mencionar anteriormente.



FIFURA 2.6 Esquema básico de la estructura de un programa en Rapid

A continuación describiremos cada uno de los bloques principales que observamos en la imagen anterior y que forman el programa:

- *Una rutina principal (main):* Rutina donde se inicia la ejecución.

- *Un conjunto de sub-rutinas:* Sirven para dividir el programa en partes más pequeñas a fin de obtener un programa modular.

- *Los datos del programa:* Definen posiciones, valores numéricos, sistemas de coordenadas, etc.

Como ya se ha mencionado antes, uno de los bloques principales de los programas de RAPID son las rutinas, pero también existen otros como por ejemplo, los procedimientos, las funciones y las rutinas TRAP.

- Un ***procedimiento*** es básicamente un subprograma. Al llamar a un procedimiento desde prácticamente cualquier parte de un programa principal, se pueden ejecutar tareas adicionales para el cálculo de los datos necesarios, mediante comprobaciones del estado actual del robot o de una parte concreta del mismo, y así sucesivamente.

- Una ***función*** nos devuelve un valor de un tipo específico.

- Las ***rutinas TRAP*** son rutinas que se ejecutan debido a la aparición de las interrupciones.

RAPID nos permite también almacenar información en datos, ya sea de manera global (acceso permitido desde diferentes módulos del programa) o de manera local (acceso desde un único módulo). El número de datos está limitado sólo por la capacidad de la memoria utilizada.

Podemos definir por tanto tres tipos diferentes de datos:

- **Constantes** - Las constantes contienen valores, como cualquier variable, pero el valor se asigna siempre en el momento de la declaración y posteriormente no es posible cambiar el valor en ningún caso.
- **Variables** - Las variables contienen valores de datos. Al detener el programa y volverlo a poner en marcha, la variable conserva su valor, pero si se mueve el puntero de programa al programa principal (main), el valor del dato de la variable se pierde.

- **Variables persistentes** - Básicamente iguales a las variables normales, pero con una diferencia importante, las variables persistentes recuerdan el último valor que se les haya asignado, incluso si el programa es detenido y puesto en marcha de nuevo desde el principio.

2.1.4 Controlador IRC5 de ABB

Dentro de la familia de controladores que posee la empresa ABB, encontramos el controlador IRC5. Este controlador está disponible en diferentes tamaños y con distintas características en función de las necesidades de la aplicación que se vaya a desarrollar.

El que utilizamos nosotros junto con el Robot IRB120 es el controlador IRC5 Compact. Es un controlador de robot que contiene los elementos electrónicos necesarios para controlar el manipulador, los ejes adicionales y los equipos periféricos. Consiste en un único aparato de medidas 258x450x565mm y 27.5 kg de peso.

En la figura siguiente podemos observar el controlador IRC5 que acabamos de mencionar y el sistema FlexPendant, del que hablaremos a continuación.

FIGURA 2.7 Controlador IRC5 y sistema FlexPendant

El controlador está provisto de los siguientes módulos:

Módulo de accionamiento, que contiene el sistema de accionamiento que proporciona la energía necesaria a los motores.

Módulo de control, que contiene el ordenador, el panel de control, el interruptor de alimentación, las interfaces de comunicación, una tarjeta de entradas y salidas digitales, la conexión para *FlexPendant*, los puertos de servicio y cierto espacio libre para equipos del usuario. El controlador también contiene el software de sistema, es decir, que incluye todas las funciones básicas de manejo y programación (RAPID).

El FlexPendant es el elemento encargado de comunicar al hombre con la máquina y viceversa. Consiste en un mando, con una pantalla táctil y distintos botones con los que poder programar, configurar, e incluso monitorizar el estado del robot.

2.2 Socket TCP/IP

El flujo habitual de trabajo con el robot irb120 es realizar un programa en lenguaje RAPID, simularlo en RobotStudio y cargarlo en el controlador IRC5 para ejecutarlo posteriormente sobre el robot real.

Pero nosotros planteamos la posibilidad de realizar un control externo mediante una comunicación a través de un socket desde un sistema remoto. Esto permite realizar aplicaciones más complejas, como por ejemplo aquellas que incluyan sensores externos, típicamente una cámara de visión artificial, etc. Por lo que surge la necesidad de realizar un socket para comunicarnos con el servidor.

Los sockets (también llamados conectores) son un mecanismo de comunicación entre procesos que permiten la comunicación bidireccional tanto entre procesos que se ejecutan en una misma máquina como entre procesos lanzados en diferentes máquinas.

En este proyecto, realizaremos un socket de comunicación, utilizando el protocolo TCP/IP que a continuación se explicará, para realizar la comunicación del brazo robótico con el software matemático Matlab.

2.2.1 Comunicación TCP/IP

TCP/IP son las siglas de Protocolo de Control de Transmisión/Protocolo de Internet (en inglés Transmission Control Protocol/Internet Protocol), que es un sistema de protocolos que hace posible servicios como Telnet, FTP o E-mail entre ordenadores que no pertenecen a la misma red.

El **Protocolo de Control de Transmisión (TCP)** permite a dos anfitriones establecer una conexión e intercambiar datos. El TCP garantiza la entrega de datos, es decir, que los datos no se pierdan durante la transmisión y también garantiza que los paquetes sean entregados en el mismo orden en el cual fueron enviados.

El **Protocolo de Internet (IP)** utiliza direcciones que son series de cuatro números ocetetos (byte) con un formato de punto decimal, por ejemplo: 192.168.1.14

Los Protocolos de Aplicación como HTTP y FTP se basan y utilizan TCP/IP.

TCP/IP es un protocolo abierto, lo que significa que se publican todos los aspectos concretos del protocolo y cualquiera los puede implementar.

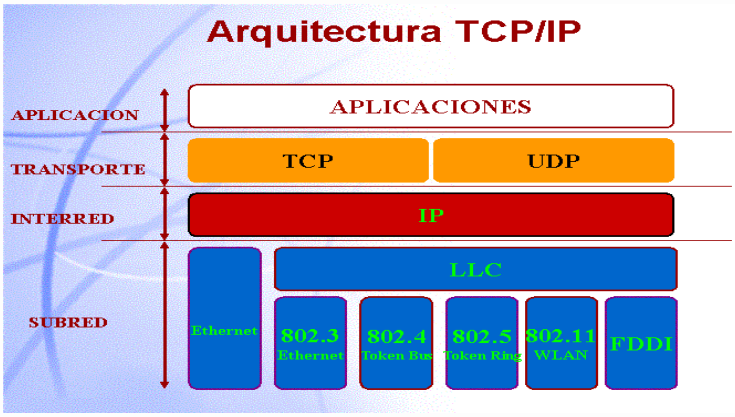


FIGURA 2.8 Arquitectura TCP/IP

Dentro de la comunicación TCP/IP, encontramos el denominado conjunto de Protocolos TCP/IP.

Todos estos servicios conforman TCP/IP, creando un protocolo potente y eficaz de red. Los diferentes protocolos dentro de TCP/IP, se mantienen de forma regular por un conjunto de estándares que son parte de la organización de Internet.

Los protocolos de transporte controlan el movimiento de datos entre dos máquinas.

- **TCP** Descrito anteriormente.
- **UDP** (User Datagram Protocol). Protocolo de Datagramas a nivel de Usuario. Permite el envío de datagramas a través de la red sin que se haya establecido previamente una conexión. Tampoco tiene confirmación ni control de flujo.
- **IP** (Internet Protocol). Protocolo de Internet. Gestiona la transmisión actual de datos.
- **ICMP** (Internet Control Message Protocol). Protocolo de Control de Mensajes de Internet. Gestiona los mensajes de estado para IP, como errores o cambios en el hardware de red que afecten a las rutas.
- **RIP** (Routing Information Protocol). Protocolo de Información de Rutas. Uno de los varios protocolos que determinan el mejor método de ruta para entregar un mensaje.
- **OSPF** (Open Shortest Path First). Abre primero el Path Mas Corto. Un protocolo alternativo para determinar la ruta.
- ARP, DNS, RARP, BOOTP, FTP, TELNET; entre otros.

Llegados a este punto, se nos planteó la posibilidad de utilizar el protocolo TCP o el protocolo UDP. A continuación realizaremos una comparativa entre ambos protocolos y posteriormente, justificaremos la decisión tomada para la realización del socket de comunicación.

Como ya hemos mencionado anteriormente, el Protocolo UDP es un protocolo de la capa de transporte para uso con el protocolo IP de la capa de red. El protocolo UDP provee un servicio de intercambio de datagramas a través de la red en modo *Best-Effort*, es decir, que no puede asegurar la entrega de los datagramas o paquetes. El servicio que provee el protocolo UDP no es fiable, ya que no garantiza la entrega o algún tipo de protección para evitar la duplicidad de los paquetes. La simplicidad de los paquetes UDP reduce la cantidad de información necesaria para poder ser utilizado, lo cual hace que el protocolo UDP sea la mejor opción en algunas aplicaciones. Un ordenador puede mandar paquetes UDP sin establecer una conexión con el dispositivo que los va a recibir. Es el propio ordenador quien completa los campos de información

en los encabezados en el paquete UDP y envía los datos junto con el encabezado a través de la capa de red mediante el protocolo IP.

Normalmente, UDP se utiliza en aplicaciones donde la confiabilidad no es crítica, pero la velocidad de transferencia sí lo es. Por ejemplo, puede ser mejor utilizar UDP en una aplicación de adquisición de datos a alta velocidad donde perder algunos datos es aceptable. Otra aplicación donde se utiliza UDP es cuando se desea transmitir un mensaje de difusión (*broadcast*) a cualquier ordenador que esté escuchando a un servidor.

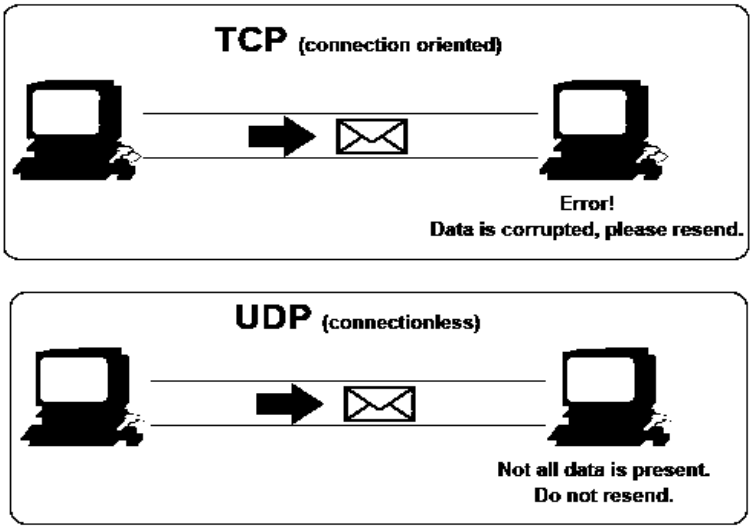


FIGURA 2.9 TCP/IP vs UDP

Podemos concluir que:

- UDP se usa cuando se buscan transmisiones con una cantidad de información baja en los paquetes y altas velocidades de transferencia, aunque se puedan perder algunos paquetes.
- TCP se usa cuando se requiere trasmisión de datos con mucha confiabilidad, es decir, que no se pierda información, por lo que es éste último, el protocolo seleccionado, debido a que no queremos perder ningún dato porque para nuestra aplicación, como veremos posteriormente, perder un dato puede implicar un error a la hora de enviar un datagrama y por tanto mandar una posición equívoca al robot o enviar de manera errónea alguna de sus características principales de funcionamiento.

2.2.2 *Implementación del servidor en lenguaje RAPID*

La figura 2.10 nos ayuda a situarnos dentro de nuestro esquema general de bloques.

En este apartado abordaremos la parte derecha de la figura, en particular el socket de servidor realizado en lenguaje RAPID.

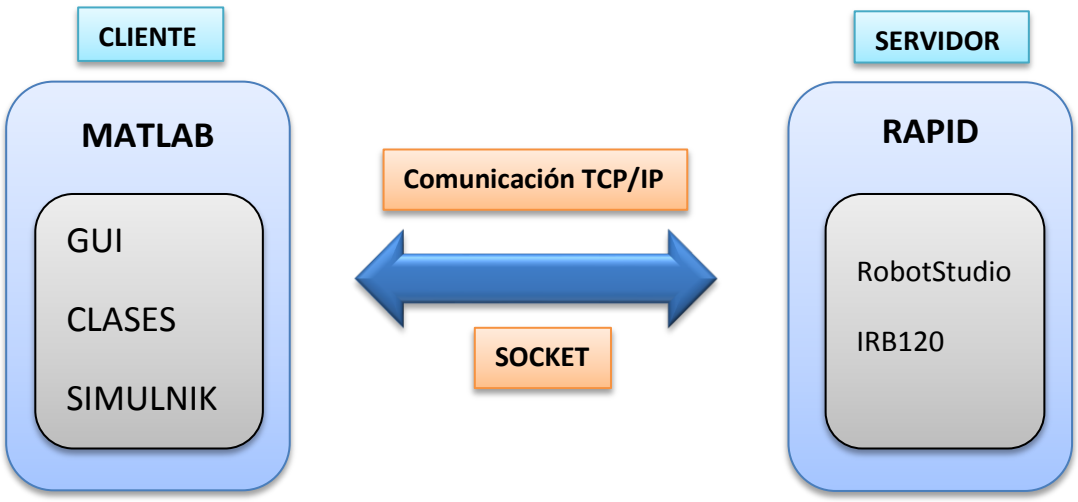


FIGURA 2.10 Esquema básico de los bloques principales del proyecto

Esta parte del proyecto ha sido realizada por Marek Jerzy Frydrysiak en su Trabajo Fin de Grado [1], que podemos consultar en la bibliografía de este documento.

En este proyecto se desarrolla un socket de comunicación de cliente programado en lenguaje C, y el servidor (programado en RAPID) que se ejecuta en el controlador del brazo robot.

Para nuestro trabajo se aprovecha el mismo socket de servidor y sin embargo se sustituirá el cliente por otro/s realizado/s en Matlab.

Es fundamental comprender la programación del servidor para poder desarrollar correctamente el socket del cliente.

A continuación detallaremos los puntos clave de este servidor en RAPID y que ha sido punto de partida para la realización de nuestro proyecto.

El lenguaje RAPID permite fundar una comunicación basada en socket entre un ordenador y un programa de control del robot. Como se mencionó antes, la mensajería

utilizada es del tipo protocolo TCP / IP con la garantía de entrega que este protocolo conlleva y que es necesaria en nuestro caso.

El código siguiente muestra la secuencia de las funciones que se han utilizado para crear un socket, para establecer una conexión, para la escucha de los datos y para volver a enviar datos.

```
MODULE TCPIPConnectionHandler
VAR socketdev socketServer;
VAR socketdev socketClient;
VAR bool      connectionStatus := FALSE;

LOCAL VAR string    ipAddress := "127.0.0.1";
LOCAL VAR num       portNumber := 1024;
LOCAL VAR string    receivedString;

   PROC EstablishConnection()
      SocketCreate socketServer;
      SocketBind socketServer, ipAddress, portNumber;
      SocketListen socketServer;
     SocketAccept socketServer,socketClient,\ClientAddress:=ipAddress,\Time:=WAIT_MAX;
   connectionStatus := TRUE;
 ENDPROC
   PROC DataProcessing()
      SocketSend client_socket \Str := "Connection successfully established!";
      WHILE connectionStatus DO
       SocketReceive client_socket \Str := receivedString;
```

```
    processData receivedString;
  ENDWHILE
  ERROR
    IF ERRNO=ERR_SOCK_CLOSED THEN
      RETURN;
    ENDIF
  ENDPROC
```

El código se localiza en un módulo separado denominado “TCPipConnectionHandler”. Es posible acceder a los procedimientos “EstablishConnection” o “DataProcesing” desde diferentes módulos que se encuentran dentro del programa que hemos creado.

Supongamos que el ejemplo de programa dado, se ha ejecutado desde el principio; lo que quiere decir que se ha llamado al procedimiento “EstablishConnection”.

Estudiaremos paso a paso el funcionamiento del código y de las funciones que lo integran. Pero antes, debemos mencionar algunas variables que se usan para realizar la conexión y el procesamiento de datos de red.

```
VAR socketdev socketServer;
VAR socketdev socketClient;
```

Las variables del tipo “socketdev” son necesarias para controlar una comunicación con otro socket dentro de una red. Estas variables también se denominan *“socket devices”* ya que representan el fin del flujo de datos entre los procesos a través de una red.

La característica principal de las variables “socketdev” es que son variables de tipo no valor, por lo que no es posible utilizar más de 32 sockets al mismo tiempo. Este tipo de variables está siendo utilizado por las funciones de RAPID para la comunicación TCP/IP, como el envío o la recepción de datos; de ahí que se hayan definido como variables globales, para que cualquier módulo del programa tenga acceso a ellas.

```
  LOCAL VAR string   receivedString;
```

En el procedimiento “DataProcessing” una parte de la información, que ha sido recibida de la red a través de la función “SocketReceive”, se almacena en la variable

"receiveString" que es de tipo string. Sin embargo para la trasmisión de otros tipos de datos a través de la red, es mejor utilizar otro formato debido a la poca eficiencia a la hora de encapsular datos o a la legibilidad del protocolo implementado.

Una vez declaradas las variables, el programa puede comenzar la conexión mediante el procedimiento denominado "EstablishConnection()". Este procedimiento utiliza las instrucciones principales de RAPID para la creación y comunicación mediantes sockets. Son las que aparecen a continuación:

```
SocketCreate socketServer;

SocketBind socketServer, ipAddress, portNumber;

SocketListen socketServer;

SocketAccept socketServer,socketClient,\ClientAddress:=ipAddress,\Time:=WAIT_MAX;
```

En primer lugar, se crea un socket de conexión, Después de eso, el socket se vincula a una dirección IP y a un número de puerto determinado. Es en este punto cuando el socket está a la espera de recibir cualquier conexión entrante, como lo haría un servidor.

Finalmente cuando se obtiene una petición de un cliente, el servidor asocia una variable "socketClient" con el punto final de la red. Este procedimiento se usa para permitir el envío y recepción de instrucciones.

Las instrucciones de envío y recepción mencionadas podemos verlas en el procedimiento "DataProcessing()".

```
SocketSend client_socket \Str := "Connection successfully established!";
```

La instrucción anterior realiza la operación de envío de datos por la red. Se escribe una cadena de datos que son enviados al cliente que previamente hemos definido. Si el programa cliente está configurado correctamente, recibirá un datagrama que contenga dicha cadena.

Sin embargo, no es el único tipo posible de datos que se pueden enviar. La instrucción de envío, al igual que la de recepción, proporciona una útil colección de parámetros que se pueden modificar para adaptarse a las necesidades del programador. Por lo tanto, en un programa adecuado de control de movimiento del robot, en vez de utilizar el tipo de dato string, se implementará una matriz de bytes.

Pata recibir los datos se debe utilizar la siguiente instrucción:

```
SocketReceive client_socket \Str := receivedString;
```

Al igual que en el procedimiento “SocketSend”, el servidor escucha los datos entrantes del cliente mediante el uso de la variable “client_socket”. En este ejemplo en concreto, el valor esperado tiene que ser de un tipo string. Sin embargo, se pueden definir otros tipos de datos según los requerimientos del programador.

A continuación abordaremos el programa principal de RAPID, responsable del control de movimiento del robot a través de la comunicación basada en el socket creado.

En la siguiente imagen podemos observar cada uno de los módulos de programa que trataremos a continuación:

FIGURA 2.11 Módulos del programa principal de RAPID

Analizaremos los principales módulos que forman el socket, con sus correspondientes procedimientos y también cómo interactúan, cómo se pasan los datos dentro del programa de RAPID y qué ventajas nos proporcionan.

Para profundizar más en cada uno de ellos debemos consultar el Trabajo Fin de Grado que aparece en el último apartado (Bibliografía) [1], ya que no es objeto de este proyecto entrar más en detalle en este punto.

MainModule

La tarea *MainModule* se crea sólo para iniciar los procedimientos que permiten establecer la conexión, una vez realizado este paso, el módulo se encarga de ejecutar instrucciones para el procesamiento de datos y para lograr el movimiento del robot.

El módulo consta de un procedimiento main () que es imprescindible para ejecutar el programa de RAPID.

TCPIPConnectionHandler

En el módulo "TCPIPConnectionHandler" encontramos el procedimiento "EstablishConnection()". Su función es establecer una conexión fiable a través de una red mediante el protocolo TCP / IP.

IRB120_Control

El módulo IRB120_Control es un núcleo principal del programa. Proporciona procedimientos muy importantes para el procesamiento de datos recibidos (reconocimiento de cabeceras, movimiento del TCP o de los ejes del robot a una posición definida), y también se ocupa de algunos errores que pueden ocurrir. Además, se ha implementado en este módulo, un bucle para realizar el control de los datos entrantes y redirigirlos al procedimiento adecuado.

SendDataHandler

El módulo "SendDataHandler" se ha creado para responder a un estado actual de la ejecución del programa. Este módulo contiene un procedimiento, llamado "SendDataProcedure", que envía los datos a una aplicación del cliente. En función de un estado del proceso, el programa servidor puede informar a un usuario (a través del programa # cliente C) de que se ha establecido la conexión, de que se ha producido un error o de que se han superado las limitaciones de espacio de trabajo.

SecurityModule

El módulo "SecurityModule" posee características que ofrecen seguridad al usuario durante la ejecución del programa. En él comprobamos las limitaciones de los movimientos del robot. También nos permite realizar cambios en los límites mediante la aplicación del cliente si fueran necesarias ciertas modificaciones.

CalibData

El módulo "CalibData" nos permite realizar la elección de la herramienta, modificar su velocidad y elegir la precisión y orientación deseadas.

Como ya hemos comentado antes, nuestro socket de servidor está desarrollado en lenguaje RAPID, por lo que debemos tener en cuenta para realizar nuestra interfaz de comunicación, el formato en el que recibe los datos ese socket del servidor.

Otro factor a tener en consideración es que estamos haciendo uso del protocolo TCP/IP, que realiza el envío de los datos a través de datagramas.

Un datagrama es un paquete de datos preparados para ser enviados a través de una red. El protocolo utilizado para la comunicación, define ciertas normas de organización de los datagramas para normalizar el diseño de los mismos. Gracias a ello, será posible crear una gran colección de diversas aplicaciones que podrán procesar los datagramas recibidos y enviarlos de vuelta a través de una red.

El socket del servidor desarrollado en código RAPID, se ha diseñado para recibir cada uno de los datagramas descritos a continuación y para realizar, por tanto, una acción específica.

Por otro lado, el socket del cliente (programa desarrollado en MATLAB), contiene una serie de funciones que se encargarán de construir los datagramas y enviarlos a través de la red al servidor.

Según las diferentes operaciones que el robot IRB120 puede realizar, es necesario definir un conjunto único de datagramas. Como mencionamos antes, el programa de RAPID identifica la cabecera (ID) de un datagrama entrante y según su valor, realiza una acción u otra. A continuación se explicarán en detalle los diferentes tipos de datagramas, sus estructuras, las limitaciones que presentan y las acciones que el robot debe realizar en ausencia de errores de ejecución.

❖ **Datagrama de Valores Articulares**

Este datagrama se encarga de enviar la información de la posición (ángulos) de los ejes del robot. Dado que el robot ABB IRB120 tiene seis grados de libertad, el datagrama

contiene seis campos (J1-J6) para los valores absolutos (en grados) de los ángulos de las articulaciones y seis campos para sus correspondientes signos.

FIGURA 2.12 Estructura del Datagrama de Valores Articulares

En la tabla siguiente podemos observar el significado de cada campo:

ID	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Identificador del datagrama Byte 1 Contiene el identificador del datagrama Joint Values
S	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor del signo Byte 0 or 1 $sgn\ (J_{n+1}) = \begin{cases} -Jn+1 & if\ S_n = 0 \\ Jn+1 & if\ Sn = 0 \end{cases}$ Contiene el signo del siguiente valor de Joint
Jn	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor Joint aboluto Byte 0 – 255 Valor absolute del ángulo Joint (en grados, n – número del Joint)

TABLA 1: Tabla descripción del Datagrama de Valores Articulares

❖ Datagrama de Orientación del TCP (Tool Central Point)

Este datagrama contiene datos de la orientación del TCP en referencia al sistema de coordenadas global, obtenido a través de los ángulos de Euler ZYX.

El procedimiento de rotación mantiene la posición actual del TCP. Por tanto, para algunos puntos en el espacio de trabajo, no será posible orientar el sistema local del TCP.

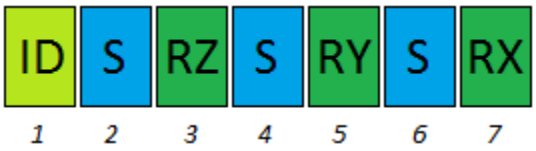


FIGURA 2.13 Estructura del Datagrama de Orientación del TCP

En la tabla siguiente podemos observar el significado de cada campo:

ID	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Identificador del datagrama Byte 2 Contiene el identificador del datagrama de Orientación del TCP
S	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor del signo Byte 0 or 1 $sgn\ (J_{n+1}) = \begin{cases} -Jn+1 & if\ S_n = 0 \\ Jn+1 & if\ Sn = 0 \end{cases}$ Contiene el signo del siguiente valor de orientación
R_{ZYX}	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor aboluto de la rotación Byte 0 – 255 Un valor absoluto de la rotación alrededor del eje EulerZYX elegido, en referencia al sistema cartesiano TCP local. Sin embargo, el sistema local sin rotar corresponde a la global (los ejes paralelos). De este modo, el robot en la posición por defecto (valores de las articulaciones 'ponen a 0) tiene la orientación del sistema local del TCP que es igual a (0, 90, 0).

		Nota: El orden de la rotación no es despreciable.

TABLA 2: Tabla descripción del Datagrama de Orientación del TCP

❖ Datagrama de Posicionamiento del TCP

Este datagrama, contiene datos del proceso de posicionamiento del TCP. Permite realizar movimientos de tipo lineal o tipo Joint.

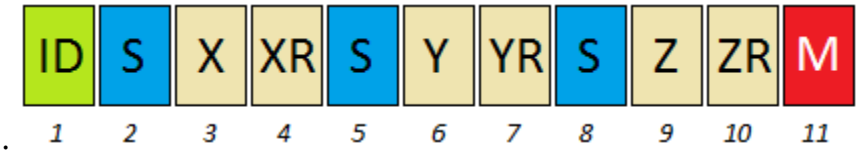


FIGURA 2.14 Estructura del Datagrama de Posicionamiento del TCP

En la tabla siguiente podemos observar el significado de cada campo:

ID	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Identificador del datagrama Byte 3 Contiene el identificador del datagrama de Posición del TCP
S	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor del signo Byte 0 or 1 $sgn\ (J_{n+1}) = \begin{cases} -Jn+1 & if\ S_n = 0 \\ Jn+1 & if\ Sn = 0 \end{cases}$ Contiene el signo del siguiente valor de Posición
X/Y/Z **XR** **YR** **ZR**	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor aboluto de la posición Byte 0 – 255 X / Y / Z contienen los valores en el rango de 0 - 255 XR / YR / ZR contienen los valores en el rango de 0 - 99 Debido a la limitación del tipo byte, es necesario dividir el valor

		de la posición (en mm, en referencia al robot de sistema de coordenadas local) si se quiere alcanzar, por el TCP, cualquier posición dentro de la zona de trabajo. Los campos X y XR se refieren a la coordenada X, Y y YR a la coordenada y así sucesivamente. Fórmula matemática para la coordenada X: $X_{real} = 100 \cdot X_{frame} + XR$ Donde, X_{real} – coordenada X referente al sistema del robot X_{frame} - define la cantidad de cientos de X_{real} XR - el resto del valor (>100)
M	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Tipo de movimiento Byte 0 or 1 Contiene el tipo de movimiento del TCP $Para \begin{cases} M=0 \ \ joint \\ M=1 \ \ lineal \end{cases}$

TABLA 3: Tabla descripción del Datagrama de Posicionamiento del TCP

❖ Datagrama de Orientación y Posicionamiento del TCP

Este datagrama es una combinación de los datagramas dos y tres. Permite controlar al mismo tiempo, la posición del TCP y su orientación.

En esta versión del protocolo de datagrama de recepción (código de RAPID), el tipo de movimiento no puede elegirse. Debido a las dificultades relacionadas con los puntos de singularidad (algunos valores confdata y las configuraciones de la muñeca) el movimiento TCP es de tipo Joint (MoveAbsJ).

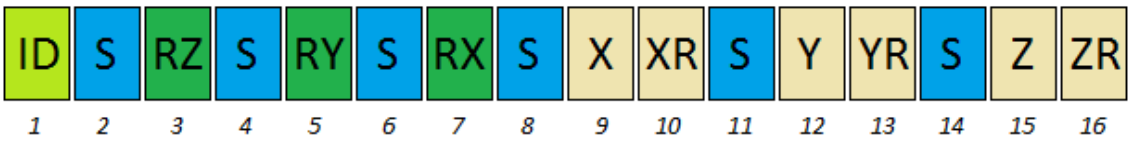


FIGURA 2.15 Estructura del Datagrama de Orientación y Posicionamiento del TCP

En la tabla siguiente podemos observar el significado de cada campo:

ID	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Identificador del datagrama Byte 4 Contiene el identificador del datagrama de Orientación del TCP
S	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor del signo Byte 0 or 1 $sgn\ (J_{n+1}) = \begin{cases} -Jn+1 & if\ S_n = 0 \\ Jn+1 & if\ Sn = 0 \end{cases}$ Contiene el signo del siguiente valor de orientación
R_{ZYX}	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor aboluto de la rotación Byte 0 – 255 Un valor absoluto de la rotación alrededor del eje EulerZYX elegido, en referencia al sistema cartesiano TCP local. Sin embargo, el sistema local sin rotar corresponde a la global (los ejes paralelos). De este modo, el robot en la posición por defecto (valores de las articulaciones 'ponen a 0) tiene la orientación del sistema local del TCP que es igual a (0, 90, 0). *Nota: El orden de la rotación no es despreciable*.
X/Y/Z **XR** **YR** **ZR**	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor aboluto de la posición Byte 0 – 255 X / Y / Z contienen los valores en el rango de 0 - 255 XR / YR / ZR contienen los valores en el rango de 0 - 99 Debido a la limitación del tipo

		byte, es necesario dividir el valor de la posición (en mm, en referencia al robot de sistema de coordenadas local) si se quiere alcanzar, por el TCP, cualquier posición dentro de la zona de trabajo. Los campos X y XR se refieren a la coordenada X, Y y YR a la coordenada y así sucesivamente. Fórmula matemática para la coordenada X: $X_{real} = 100 \cdot X_{frame} + XR$ Donde, X_{real} – coordenada X referente al sistema del robot X_{frame} - define la cantidad de cientos de X_{real} XR - el resto del valor (>100)

TABLA 4: Tabla descripción del Datagrama de Orientación y Posicionamiento del TCP

❖ Datagrama de velocidad del TCP, herramienta y limitaciones

El último datagrama se utiliza para ajustar el movimiento del robot a las preferencias del usuario y a los requisitos de la tarea actual. En otras palabras, se puede ajustar a la velocidad del TCP y de la herramienta Por otra parte, gracias a este datagrama, también es posible modificar las limitaciones de movimiento.

Las limitaciones sólo afectan a la posición del TCP, no a toda la estructura del robot. Por tanto, hay que tener en cuenta, que en algunos puntos en el espacio de trabajo del robot IRB120, el efector final sí puede llegar con el margen de seguridad, pero uno o más eslabones del robot pueden estar fuera de los límites o en colisión con la jaula de seguridad actual.

FIGURA 2.16 Estructura del Datagrama de Velocidad del TCP, herramienta y limitaciones

En la tabla siguiente podemos observar el significado de cada campo:

ID	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Identificador del datagrama Byte 5 Contiene el identificador del Datagrama de Velocidad del TCP, de la herramienta y limitaciones.
T	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Herramienta del efector Byte 0 or 1 El identificador para la herramienta seleccionada. Este cuadro afecta a la posición del TCP. $Para \begin{cases} T=0 \ \ sin\ herramienta \\ T=1 \ \ rotulador \end{cases}$
V_T **V_{TR}**	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Velocidad del TCP Byte 0 – 1999 [mm/s] V_T contiene valores en el rango de 0 – 19 V_{TR} contiene valores en el rango de 0 – 99 Estos cuadros contienen la velocidad elegida para el TCP. Debido al tamaño máximo del tipo byte, es necesario dividir el valor en dos partes, que se describen en la fórmula siguiente: $V_{TCP} = 100 \cdot V_T + V_{TR}$
V_R **V_{RR}**	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Velocidad de Orientación Byte 0 – 420 [º/s] V_R contiene valores en el rango de 0 – 4 V_{RR} contiene valores en el rango de 0 – 99 Estos cuadros contienen la velocidad de orientación.

		Debido al tamaño máximo del tipo byte, es necesario dividir el valor en dos partes, que se describen en la fórmula siguiente: $V_{Orientación} = 100 \cdot V_R + V_{RR}$
S	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor del signo Byte 0 or 1 $sgn\ (J_{n+1}) = \begin{cases} -Jn+1 & if\ S_n = 0 \\ Jn+1 & if\ Sn = 0 \end{cases}$ Contiene el signo del siguiente valor de orientación
X_L / Z_L **X_LR** **Z_LR**	**Nombre del cuadro:** **Tipo de datos:** **Valor:** **Anotaciones:**	Valor absoluto de la limitación Byte X_L / Z_L contiene valores en el rango de 0 – 255 X_LR / Z_LR contiene valores en el rango de 0 – 99 Estos cuadros contienen la localización (en mm en los ejes X y Z) de las superficies con las que limta el TCP. Debido al tamaño máximo del tipo byte, es necesario dividir el valor en dos partes. Los campos X y XR se refieren a la coordenada X, y los campos Y y YR a la coordenada Y. La fórmula matemática para la coordenada X: Xreal= 100 • Xframe + XR Donde, Xreal – coordenada X de la superficie límite en el sistema de referencia del robot Xframe - define la cantidad de cientos de Xreal XR - el resto del valor (>100)

TABLA 5: Tabla descripción del Datagrama de Velocidad del TCP, herramienta y limitaciones

2.3 Matlab / Simulink

En este apartado hablaremos sobre la herramienta de software matemático MATLAB y sobre los distintos componentes que son utilizados en este proyecto.

2.3.1 Generalidades

MATLAB es un lenguaje de alto nivel y un entorno interactivo para el cálculo numérico, la visualización y la programación. Mediante Matlab, es posible analizar datos, desarrollar algoritmos y crear modelos o aplicaciones. El lenguaje, las herramientas y las funciones matemáticas incorporadas, permiten explorar diversos enfoques y llegar a una solución antes que con hojas de cálculo o lenguajes de programación tradicionales, como pueden ser C/C++ o Java.

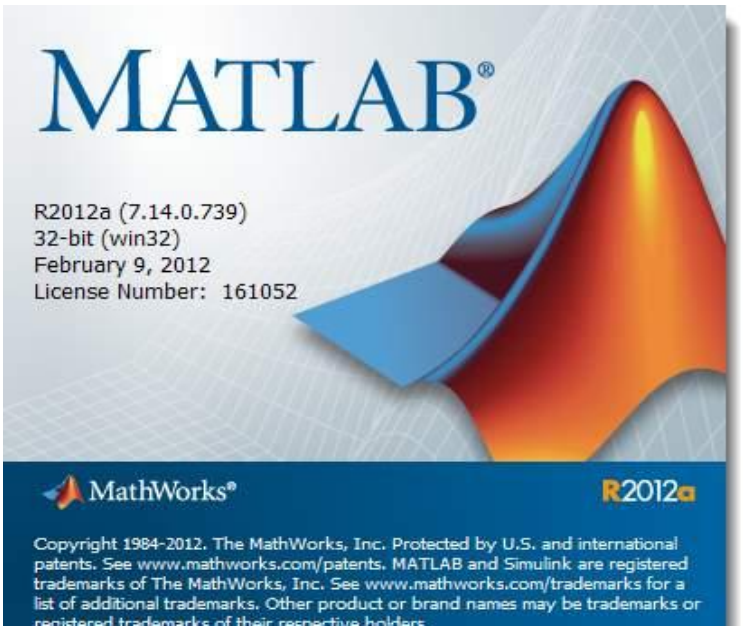


Figura 2.17 MATLAB R2012a

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete MATLAB dispone, entre otras muchas, de dos herramientas adicionales que expanden sus prestaciones: **Simulink** (plataforma de simulación multidominio) y **GUIDE** (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes) y las de Simulink con los paquetes de bloques (blocksets).

Es cierto que en sus orígenes estaba centrado en el cálculo matemático y en concreto el matricial; poco a poco se ha ido extendiendo hasta abarcar prácticamente todos los ámbitos de la ingeniería. Algunas de estas extensiones pertenecen al propio programa, mientras otras, han sido implementadas por terceros y a día de hoy continúan realizando su desarrollo y adaptación a las versiones sucesivas de Matlab.

En este proyecto utilizaremos sobre todo las funciones básicas del software y también haremos uso de las herramientas adicionales que ofrece Matlab como Simulink y GUIDE para el desarrollo de la aplicación final y de la interfaz gráfica, respectivamente.

Otra de las opciones que incluye Matlab y que utilizaremos serán las cajas de herramientas (toolboxes); mencionadas anteriormente y en concreto la Toolbox de Robótica (Robotics Toolbox) [4] desarrollada por Peter Corke y la "*Instrument Control Toolbox*" [5].

En apartados posteriores de este documento, entraremos más en detalle de esta Toolbox y todas sus funciones, orientada a nuestra aplicación final de pintar números.

2.3.2 Comunicación TCP/IP con MATLAB

En este punto volvemos a recordar la figura que mostrábamos al principio del documento en la que vemos la estructura de bloques principal del proyecto (Figura 2.18).

Para volver a situarnos, nos fijaremos ahora en la parte izquierda de dicha figura, en la que encontramos el bloque del cliente. El socket del cliente ha sido desarrollado en Matlab y se encargará de comunicarse a través del protocolo TCP/IP con el servidor, programado en lenguaje RAPID.

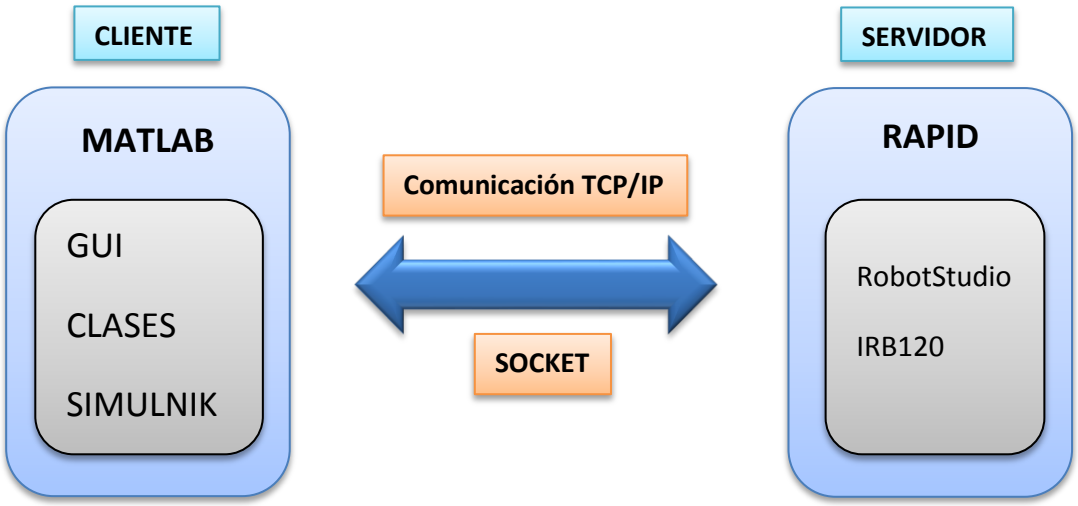


FIGURA 2.18 Esquema básico de los bloques principales del proyecto

Para realizar la comunicación TCP/IP con Matlab, tenemos que hablar antes de una de las toolbox más importantes utilizadas en este proyecto; la Toolbox de Control de Instrumentos (Instrument Control Toolbox).

La "*Instrument Control Toolbox*" nos permite controlar y comunicarnos con dispositivos externos a través de los protocolos de comunicación GPIB y VXI directamente desde Matlab.

Proporciona también interfaces intuitivas para conectar y comunicarse con el correspondiente instrumento. También podemos usar estas interfaces con dispositivos a través del puerto serie.

En nuestro caso, hemos utilizado esta toolbox para poder llevar a cabo el control de nuestro brazo robot a través del envío de datagramas por el puerto serie mediante el protocolo TCP/IP Cómo ya hemos comentado, la comunicación en Matlab se realiza programando un cliente y un servidor. El servidor ya está implementado en lenguaje Rapid por lo que será el cliente el que se desarrollará en Matlab.

Para realizar la comunicación TCP/IP con Matlab tenemos que crear un objeto a través del uso de una instrucción perteneciente a la "*Instrument Control Toolbox"*. La sintaxis de estas instrucciones tiene diferentes variantes, a continuación se muestran algunas de ellas:

- *obj = tcpip('rhost')* - Crea un objeto TCPIP, obj, asociado a un host remoto (rhost) y el valor por defecto del Puerto remoto es 80.
- *obj = tcpip('rhost',rport)* - Creat un objeto TCPIP con valor del Puerto remoto, rport.
- *obj = tcpip(...,'PropertyName',PropertyValue,...)* - Crea un objeto TCPIP con los valores de nombre / valor de propiedad, especificados. Si se especifica un nombre de propiedad o un valor de la propiedad no válido, no se crea el objeto.
- *obj = tcpip('localhost', 30000, 'NetworkRole', 'client')* - Crea un objeto TCPIP, obj, que es una interfaz de cliente para un socket servidor.

También debemos hablar de una serie de instrucciones que son utilizadas en el proceso de comunicación:

- **fopen**(obj) - conecta el objeto creado con el instrumento.
- **fclose**(obj) - desconecta el objeto del instrumento
- **fwrite** (obj,A,'precision','mode') - escribe datos binarios (A) con precisión especificada por el acceso (‘precision’) y precisión de línea de comandos especificada por el modo (‘mode’).
- A = **fread** (obj,size,'precision') - lee los datos binarios con precisión especificada (‘precision’). La precisión controla el número de bits leídos para cada valor y la interpretación de esos bits como un dato entero (int).

A continuación se muestra un ejemplo de código:

```
% Creamos el objeto, definiendo sus propiedades.

>> t=tcpip('localhost', 30000, 'NetworkRole', 'client');

% Abrimos la conexión.

>> fopen(t)

% Escribimos los datos que queramos, en este caso, data.

>> fwrite(t, data)
```

```
% Cerramos la conexión.

>> fclose(t)
```

Estas son las instrucciones que se utilizan básicamente en la implementación del código, para realizar la comunicación TCP/IP y conseguir conectar Matlab con el servidor implementado en RAPID, ya sea ejecutándose en el software específico RobotStudio (simulación) o en el Robot real.

2.3.3 Desarrollo de Interfaces Gráficas (GUIs)

Una interfaz gráfica de usuario (GUI), es una interfaz construida a través de objetos gráficos, tales como menús, botones, listas y barras de desplazamiento.

Esos objetos formarán una interfaz en el momento que se defina una acción a realizar cuando se produzca algún cambio o acción en los mismos (una pulsación de ratón, arrastrar un objeto con el puntero del ratón, etc.).

Para diseñar una interfaz gráfica efectiva, se deben escoger los objetos gráficos adecuados que se desean visualizar en la pantalla, distribuirlos y colocarlos de una manera lógica para que la interfaz sea fácil de usar y determinar las acciones que se realizarán al ser activados estos objetos gráficos por el usuario.

Matlab permite desarrollar y definir un conjunto de elementos (botones, menús, ventanas...) que permiten utilizar de manera fácil e intuitiva, programas realizados en este entorno. Los objetos de una GUI en Matlab se dividen en dos clases:

- ***Controles***
- ***Menús***

Los gráficos de Matlab tienen una estructura jerárquica, formada por objetos de distintos tipos.

En la siguiente figura podemos observar la forma que presenta esta jerarquía:

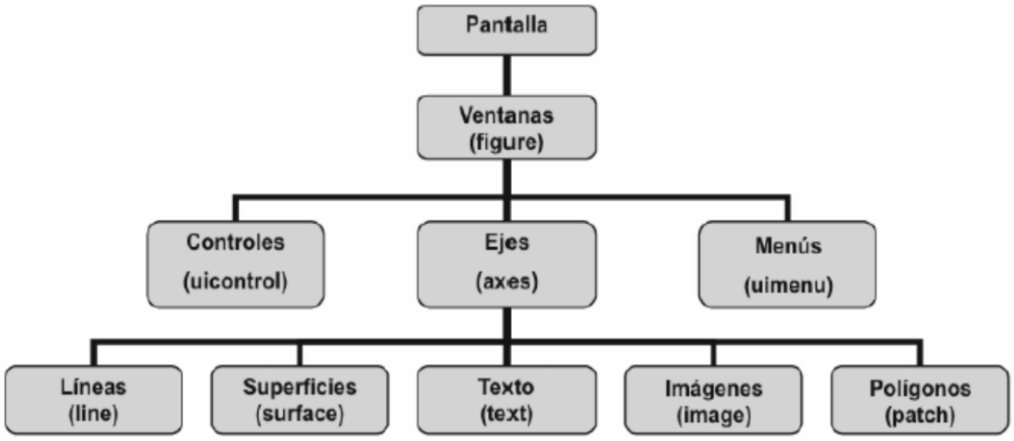


FIGURA 2.19 Jerarquía gráfica de Matlab

Tal y como se indica en la figura anterior, la pantalla es el objeto más general y del que dependen todos los demás y únicamente puede haber un objeto pantalla. Una pantalla puede contener una o más ventanas (*figures*). A su vez, cada una de las ventanas puede tener uno o más ejes de coordenadas (*axes*) con los que representar otros objetos de más bajo nivel. Una ventana puede tener también controles (*uicontrol*s), tales como botones, barras de desplazamiento, botones de selección o de opción, etc. y menús (*uimenu*s). Finalmente, los ejes pueden contener los cinco tipos de elementos gráficos que permite Matlab y que se indican en el gráfico anterior (*line, surface, text, image* y *patch*).

De todo esto se deduce que en el interface gráfico de Matlab hay objetos padres e hijos. Cuando se borra un objeto de Matlab, automáticamente también se borran todos los objetos que sean sus descendientes. El comando a utilizar será: *delete (identificador).*

Además, cada uno de los objetos de Matlab tiene un identificador único denominado *handle*. Si una pantalla tiene muchos objetos, cada uno de ellos tendrá asignado un *handle*. El objeto raíz (Pantalla), es siempre único y su identificador es cero. El identificador de las ventanas es un entero, que aparece en la barra de nombre de dicha ventana. Los identificadores de otros elementos gráficos, son número flotantes, que pueden ser obtenidos como valor de retorno y almacenados en variables de Matlab.

Así mismo, Matlab puede tener varias ventanas abiertas, pero siempre hay una única activa. También una ventana puede tener varios ejes, pero sólo uno activo. Matlab dibuja en los ejes activos de la ventana activa.

Se pueden obtener los identificadores de la ventana activa, de los ejes activos y del objeto activo con los siguientes comandos: gcf *(get current figure),* gca (*get current axes)* y gco *(get current object).*

La herramienta GUIDE (GUI Development Environment) es una herramienta de Matlab para el desarrollo de aplicaciones GUI bajo un entorno gráfico. GUIDE facilita al desarrollador, un conjunto de herramientas de uso sencillo que simplifican mucho el diseño y la programación de GUIs.

Actualmente, las herramientas que nos ofrece GUIDE, permiten además de diseñar la ventana, generar un archivo de extensión .M que contiene el código para controlar el lanzamiento e inicialización de la aplicación GUI.

Se puede abrir GUIDE desde el Lanch Path de Matlab haciendo doble clic sobre GUIDE o tecleando desde la línea de comandos "guide", abriendo la siguiente pantalla, donde se seleccionará crear un nuevo GUI o abrir uno ya existente.

Para crear una nueva GUI, seleccionaremos la opción "Blank BUI (Default)" y nos aparecerá una ventana que servirá como editor de nuestro nuevo GUI. En ella podremos poner los objetos gráficos que necesitemos para nuestra aplicación.

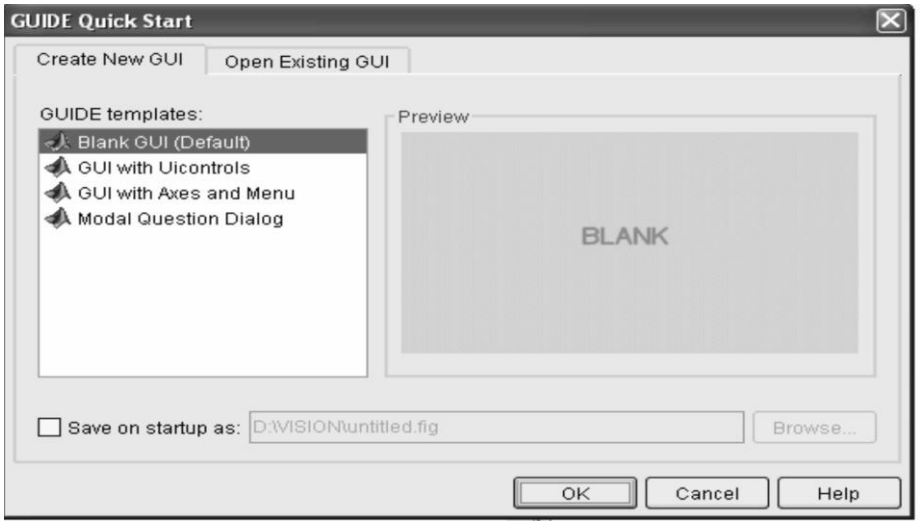


FIGURA 2.20 GUIDE Start

El desarrollo de una aplicación GUI, implica diseñar la ventana (colocando cada uno de los objetos gráficos dentro del área de trabajo) y programar los componentes para definir cuáles son las funciones que realizará cada uno de los objetos cuando se actúe sobre ellos.

A continuación, se muestra el formato del área de trabajo donde vamos a desarrollar la nueva GUI.

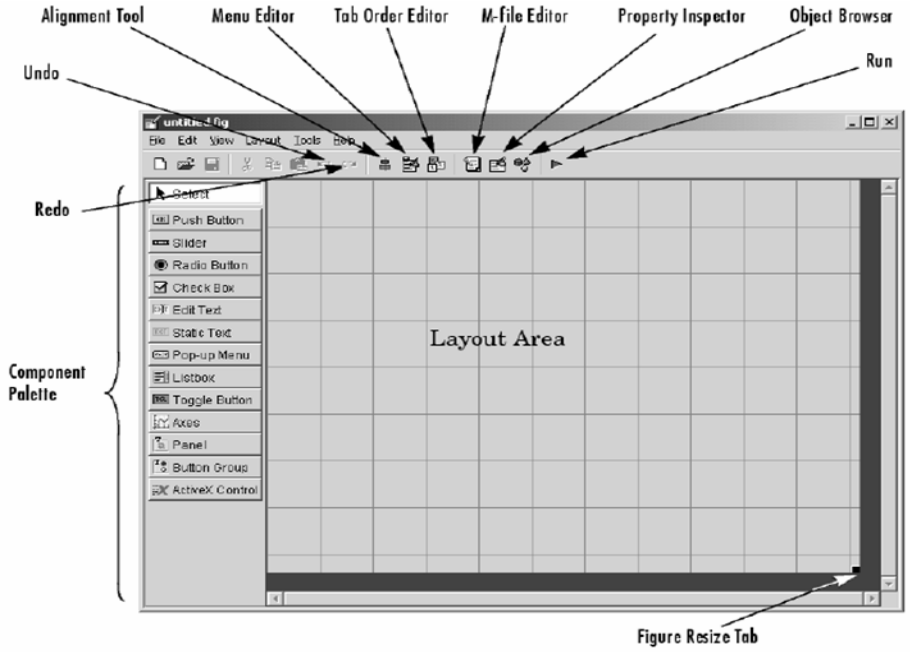


FIGURA 2.21 Layout Area

Arrancando la herramienta GUIDE, tenemos la posibilidad de configurar las opciones del GUI en el menú Tools opción GUI Options

Una vez que se configura la herramienta GUIDE, se procede a crear en el layout, los objetos gráficos que compondrán el GUI de nuestra aplicación. Se muestran a continuación las opciones más empleadas en el editor de layout:

- ***Herramienta de Alineación (Alignment Tool)***

Esta herramienta permite ajustar con respecto a una referencia, la posición de un objeto dentro del layout. Las operaciones de alineación se realizarán sobre los objetos seleccionados, una vez pulsado *"Apply"*. Esta herramienta proporciona dos tipos de alineaciones, tanto en horizontal como en vertical:

- ***Buscador de Objetos (Object Browser)***

Esta herramienta permite mostrar el orden que se ha seguido a la hora de ir insertando los distintos controles (*uicontrols*) u objetos gráficos. Es muy útil cuando se tiene un GUI muy complejo (con muchos objetos insertados). Además, haciendo doble clic con

el ratón sobre cualquier elemento, se accede a sus características propias, el *Property Inspector*.

- ***Menú Editor***

Esta herramienta permite diseñar los menús que contendrá el GUI cuando se ejecute. Matlab es capaz de crear dos tipos de menús:

- *Menu Bar:* Son los menús típicos de las barras de herramienta (Archivo, Edición, Insertar...).
- *Context Menu:* Son los menús que saldrán en cada objeto al pulsar el botón derecho del ratón. La forma de crearlos es igual a los anteriores, salvo que hay que asociar los menús al elemento gráfico en cuestión.

- ***Uicontrols***

Los *uicontrols* son los denominados controles de usuario del Interface (*User Interface Controls*). En el área de *Layout* se pueden insertar los siguientes uicontrols:

- ***Push Buttons***: Permiten pasar de un estado al siguiente en la aplicación, cuando el usuario pulsa sobre ellos (bien con el ratón o bien con la tecla de tabulador y el enter). Son los típicos botones de Activación: Cancelar, OK...
- ***Toggle Buttons***: Generan una acción e indican un estado binario (on/off). Para ejecutar la callback asociada a este botón, se necesita leer el valor con el comando: get(gcbo,'value').
- ***Radio Buttons:*** Este tipo de botones sólo tiene dos estados: activado o desactivado.
- ***Edit Text:*** Sirve para modificar y escribir cadenas de texto.
- ***Static Text:*** Sirve para insertar un texto fijo en controles, botones, etc. Dicho texto no se puede modificar en ejecución.
- ***Check Boxes:*** Cajas de selección que se activan pulsando sobre el cuadrado y aparece un stick. Tiene dos posibles valores: seleccionado o no seleccionado.

- ***Sliders:*** Se suelen emplear para que el usuario de la aplicación seleccione valores numéricos. Aparece una barra, que puede tener una orientación vertical u horizontal; dicha orientación se establece con el property inspector. El slider tiene asignadas cuatro variables: value, max, min y sliderstep.

- ***List Boxes:*** Aparece una lista con varias opciones, donde el usuario deberá seleccionar una de ellas.

- ***Pop-Up Menús:*** Listas desplegables donde se muestran varias opciones para que el usuario de la aplicación pueda seleccionar alguna/s de ellas.

- ***Axes y figures:*** Permiten insertar ejes y figuras en el GUI

Llegados a este punto debemos mencionar una parte muy importante de la creación de una interfaz GUI que son las funciones callbacks.

Cada objeto gráfico y/o figura que se crea en el GUI, lleva asociado un callback que permite definir la acción que llevará a cabo la aplicación cuando se actúe sobre dicho objeto.

Existen dos tipos de callback, según se trate de objetos gráficos o figuras*:*

- *Callbacks para objetos gráficos*

  Todos los objetos gráficos que se añaden al GUI, tienen una serie de propiedades que permiten al programador definir funciones callbacks para asociarlas a ellas.

- *Callbacks para figuras*

  Las figuras o ventanas tienen otras propiedades que pueden asociarse con sus respectivas funciones callbacks tras las correspondientes acciones del usuario.

Cuando salvamos la aplicación GUI que hemos creado, se nos generan automáticamente dos: *archivo.FIG* y *archivo.M*.

El *archivo.FIG* contiene la descripción del GUI que se ha diseñado. En él se guarda la figura con los objetos gráficos insertados en ella, para su posterior modificación o uso. También contiene la información relativa a las propiedades de cada objeto.

El *archivo.M*, es un fichero que contiene las funciones necesarias para controlar y ejecutar el GUI y las funciones *callbacks*. En este fichero, aparece una función callback para cada objeto gráfico diseñado en el archivo.FIG. El programador, dependiendo de lo que quiera que haga dicho objeto, introducirá el código necesario.

Por último nos queda comentar las estructuras handles: concepto importante para el uso de esta herramienta. Como acabamos de explicar anteriormente, la herramienta GUIDE, genera automáticamente código en un fichero Matlab con extensión .m, donde se encuentra las funciones que controlan y ejecutan el GUI y las funciones callbacks.

Dicho código está basado en la estructura handles y en él se puede añadir el código para ejecutar los hilos de GUI. La estructura handles es pasada como una entrada a cada callback. La estructura handles puede usarse para:

- Compartir datos entre callbacks
- Acceder la data en el GUI

Por ejemplo, para almacenar los datos contenidos en una variable X, se fija un campo de la estructura handles igual a X y se salva la estructura con guidata como se muestra a continuación:

```
handles.x=X

guidata(hObject, handles)
```

En cualquier momento se puede recuperar la data en cualquier callback con el comando:

```
X=handles.x
```

Es muy importante dominar el uso de la estructura handles para implementar y desarrollar aplicaciones GUI que requieran continuamente cambios en los parámetros de los botones o user interface controls (uicontrols) según los mandatos del usuario. Se puede decir que con la estructura handles, tenemos el control absoluto de lo que pueda pasar en la aplicación GUI desarrollada.

2.3.4 Creación de clases en Matlab

Para permitir la comunicación con el robot desde cualquier programa de Matlab, es necesario crear algún tipo de función que permita enviar datos por el socket desde cualquier fichero .m.

Se ha optado por la creación de una clase para poder así beneficiarnos de las ventajas que ya conocemos, como es por ejemplo su facilidad de utilización así como las ventajas asociadas a la programación orientada a objetos.

La creación de aplicaciones de software, normalmente implica el diseño de la forma de representación de los datos de la aplicación y por tanto, nos permite determinar cómo implementar las operaciones realizadas sobre dichos datos.

Los programas de procedimiento pasan los datos a las funciones, las cuales llevan a cabo las operaciones necesarias para el tratamiento de los datos. La programación orientada a objetos, se encarga de encapsular datos y realizar operaciones en los objetos que interactúan entre sí, a través de la interfaz propia del objeto.

El lenguaje de MATLAB le permite crear programas utilizando ambas técnicas: los programas de procedimiento y la programación orientada a objetos y por lo tanto realizar cualquier aplicación utilizando objetos y funciones comunes en sus programas.

Una clase describe un conjunto de objetos que posee características comunes. Los objetos son instancias específicas de una clase. Los valores contenidos en las propiedades del objeto son los que hacen que un objeto sea diferente de otros objetos de la misma clase.

Las funciones definidas por la clase son los llamados métodos, que se encargan de unificar las propiedades de los objetos que son comunes a todos los de una misma clase.

Tras esta introducción, explicaremos en profundidad a través de un ejemplo de Matlab [7], la creación de una clase utilizando la programación orientada a objetos.

Para explicar las clases en MATLAB, debemos conocer previamente ciertos conceptos que utilizan las siguientes instrucciones para describir las diferentes partes de una clase y para la definición de la misma:

- **Definición de clase** - Descripción de lo que es común a todas las clases.
- **Propiedades** - Almacenamiento de los datos que utilizará la clase.
- **Métodos** - Funciones especiales que implementan las operaciones que normalmente hacen referencia sólo a una parte de la clase.
- **Eventos** - Mensajes que son definidos por las clases y ejecutados cuando se produce alguna acción específica.
- **Atributos** - Valores que modifican el funcionamiento de las propiedades, métodos, eventos y clases.
- **Oyentes** - Objetos que responden a un evento específico mediante la ejecución de una función de devolución de llamada cuando se emite el aviso de evento.
- **Objetos** – Los componentes de las clases, los cuales contienen los valores de los datos reales almacenados en las propiedades de los objetos.
- **Subclases** - Clases que derivan de otras clases y que heredan los métodos, propiedades y eventos de esas mismas clases (las subclases facilitan la reutilización del código definido en la superclase de la que derivan).
- **Superclases** - Clases que se utilizan como base para la creación de clases definidas más específicamente (es decir, subclases).
- **Paquetes** - Carpetas que definen un ámbito de la clase y la función de denominación.

A continuación, utilizaremos un ejemplo de Mathworks (almacenamiento y uso de unos datos de tracción de ciertos materiales a través de la definición de una clase).

% Creamos la clase TensileData y en las propiedades ponemos las características necesarias del material que vamos a evaluar.

```
classdef TensileData
  properties
    Material = '';
    SampleNumber = 0;
    Stress
    Strain
    Modulus = 0;
  end
```

```
end
```

% Creamos un ejemplo y le asignamos los datos.

```
td = TensileData;

td.Material = 'Carbon Steel';
td.SampleNumber = 001;
td.Stress = [2e4 4e4 6e4 8e4];
td.Strain = [.12 .20 .31 .40];
td.Modulus = mean(td.Stress./td.Strain);
```

% Añadimos métodos. La función que definimos será para asegurarnos que el usuario introduce correctamente el material.

```
methods
    function obj = set.Material(obj,material)
      if ~(strcmpi(material,'aluminum') ||...
        strcmpi(material,'stainless steel') ||...
        strcmpi(material,'carbon steel'))
        error('Material must be aluminum, stainless steel, or carbon steel')
      end
```

% Para simplificar la creación de la clase creamos un constructor.

```
function td = TensileData(material,samplenum,stress,strain)
  if nargin > 0 % Support calling with 0 arguments
    td.Material = material;
    td.SampleNumber = samplenum;
    td.Stress = stress;
    td.Strain = strain;
  end
end % TensileData
```

% Este constructor se utilizaría de la siguiente forma:

```
>> td = TensileData('carbon steel',1,[2e4 4e4 6e4 8e4],[.12 .20 .31 .40]);
```

Se pueden añadir tantas funciones como queramos dentro de los métodos.

En este caso se podría realizar otras dos funciones: una por ejemplo para calcular el módulo del material a partir de los datos introducidos por el usuario y otra para representar gráficamente por medio de un plot diferentes datos del material.

%Ejemplo del código de la función para obtener el módulo.

```
methods
  function modulus = get.Modulus(obj)
 [..]

end % Modulus get method
end % methods
```

%Ejemplo de código de la función para representar los datos de forma gráfica mediante la sentencia plot.

```
function plot(td,varargin)
  plot(td.Strain,td.Stress,varargin{:})
  title(['Stress/Strain plot for Sample',...
end % plot
end % methods
```

2.3.5 Creación de bloques para Simulink

Simulink, es una *toolbox* especial de MATLAB que sirve para simular el comportamiento de los sistemas dinámicos. Puede simular sistemas lineales y no lineales, modelos en tiempo continuo y tiempo discreto y sistemas híbridos de todos los anteriores.

Proporciona al usuario un entorno gráfico que facilita enormemente el análisis, diseño y simulación de sistemas (de control, electrónicos, etc.) al incluir una serie de rutinas que resuelven los cálculos matemáticos de fondo, junto con una sencilla interfaz para su uso. También le permite dibujar los sistemas como diagramas de bloques tal como se haría sobre un papel.

El conjunto de componentes incluidos en el programa Simulink, incluye también bibliotecas de fuentes de señal, dispositivos de presentación de datos, sistemas lineales y no lineales, conectores y funciones matemáticas. En caso de que sea necesario, se pueden crear nuevos bloques a medida por el usuario.

Con las nuevas versiones, Simulink ha ido ampliando sus librerías de bloques (*blocksets*) y capacidades. En concreto, destaca el paquete *Stateflow*, que permite la simulación de máquinas de estados

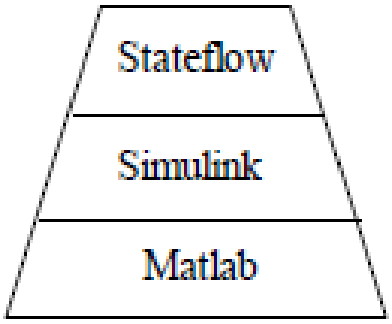


Figura 2.22 Jerarquía de Matlab, Simulink, Stateflow

Además algunas *toolboxes* de MATLAB incorporan también bloques de Simulink. Es el caso por ejemplo, de la *Toolbox de Robótica* o de la *Control System Toolbox*. La primera de ellas será base importante de este proyecto, sobre todo en el desarrollo de la aplicación final de pintar números para realizar el control del brazo robot.

Como ya hemos mencionado antes, todos los componentes básicos de Simulink se pueden encontrar en su biblioteca. Nosotros nos vamos a centrar en un bloque en concreto, ya que más adelante utilizaremos para desarrollar una interfaz y conseguir comunicarnos con el robot.

El bloque al que nos referimos es el bloque ***Interpreted Matlab Function,*** perteneciente al grupo *User- Defined Functions.* Este bloque aplica la función de MATLAB especificada o la expresión que tenga en su entrada.

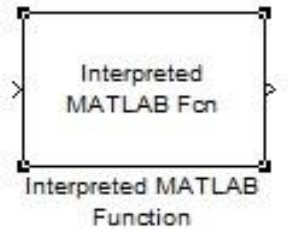


Figura 2.23 Bloque Simulink Interpreted Matlab Function

La salida de la función debe coincidir con las dimensiones de la salida del bloque o de lo contrario se producirá un error.

Algunas expresiones válidas para este bloque son:

sin (u) // atan2 (u (1), u (2)) // u (1) ^ u (2)

A parte de una expresión, también se puede escribir en ese campo el nombre de un archivo.m en el que tengamos el código correspondiente asociado a esa función.

El bloque *Interpreted Matlab Function* acepta una entrada real o compleja de tipo double y genera una salida real o compleja de tipo doble, dependiendo como ajustemos el parámetro de *Output Signal Type* (tipo de señal de salida).

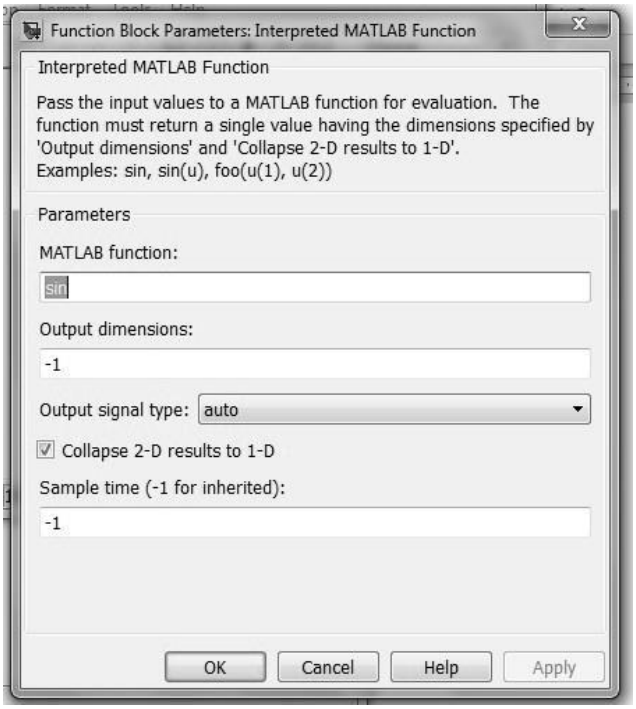


Figura 2.24 Parámetros del bloque Interpreted Matlab Fcn

En la figura de arriba podemos observar el cuadro de diálogo que aparece al hacer doble clic sobre el bloque. En él ajustaremos los parámetros necesarios para conseguir la funcionalidad deseada para nuestra aplicación.

- *MATLAB function.* Especifica la función o expresión que hemos asignado al bloque.
- *Output dimensions.* Indica las dimensiones de la señal de salida de este bloque. Si las dimensiones de la salida son iguales que las de entrada, debemos poner -1. Si esto no ocurre debemos introducir las dimensiones de la señal de salida. En cualquier caso, las dimensiones de salida deben coincidir con las dimensiones del valor retornado por la función.

- *Output signal type*. Indica el tipo de señal de salida. Podemos elegir entre: real, complejo o auto (mismo tipo que la señal de entrada).
- *Collapse 2-D results to 1-D.* Selecciona esta casilla de verificación para convertir una matriz 2-D a una matriz 1-D que contendrá los elementos de la matriz 2-D ordenados por columnas.
- *Sample Time.* Indica el intervalo de tiempo entre las muestras.

Por último y no menos importante nos queda mencionar una función importante de la herramienta Simulink; la creación de subsistemas y máscaras.

Los subsistemas son utilizados cuando nuestro modelo se hace complicado, ya que aumenta de tamaño y complejidad. El agrupamiento es útil por una serie de razones:

- Ayuda a reducir el número de bloques visualizados.
- Permite mantener juntos los bloques que están funcionalmente relacionados.
- Hace posible establecer un diagrama de bloques jerárquico.

La manera más rápida de crear un subsistema, es ir seleccionado los bloques que queremos unir y haciendo clic en el botón derecho del ratón seleccionamos la opción de crear subsistema.

Automáticamente se crean, rotulan y numeran los puertos de entrada y salida del subsistema y al hacer doble clic sobre el bloque, se nos abrirá una ventana en la que podemos ver los bloques que componen el subsistema por si necesitamos cambiar algún parámetro de esos bloques.

También podemos crear una máscara para nuestro subsistema. La máscara la utilizaremos sobre todo con el fin de unificar todos los parámetros de los bloques y acceder a ellos más fácilmente. También nos proporcionará una mejora en la estética del bloque.

Para enmascarar el bloque subsistema, seleccionamos el bloque y haciendo clic en el botón derecho elegimos Crear máscara.

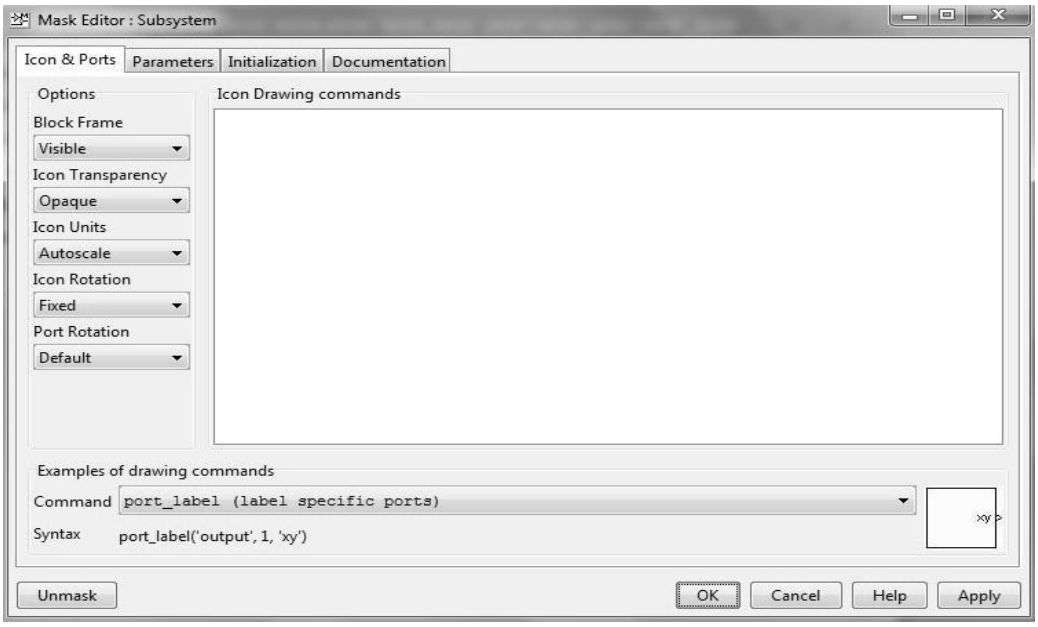


Figura 2.25 Cuadro diálogo Parámetros Create Mask

El cuadro de diálogo del enmascaramiento, que podemos observar en la figura anterior, nos permite describir el bloque enmascarado definiendo sus campos de parámetros, las órdenes de inicialización, el icono y el texto de ayuda.

El título del cuadro de diálogo y el nombre y tipo del bloque se derivan del bloque que está siendo enmascarado.

3 Desarrollo de las interfaces de comunicación

3.1 Visión general de las interfaces desarrolladas

En este punto hablaremos en detalle de las interfaces desarrolladas para conseguir la comunicación con el brazo Robot. Se han realizado tres interfaces a través de tres herramientas de Matlab. En la siguiente figura observamos el bloque correspondiente y las tres herramientas mencionadas anteriormente:

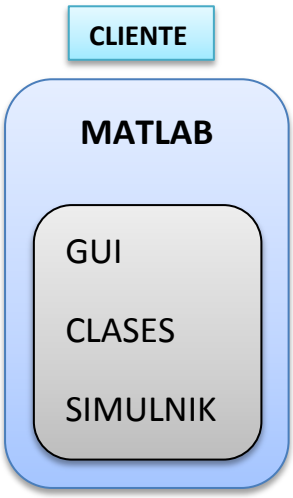


FIGURA 3.1 Esquema básico de los bloques principales del proyecto

La primera de la que hablaremos será la **interfaz GUI**. Una interfaz gráfica GUI tiene como característica principal que es *intuitiva*. Una persona que desconozca el funcionamiento de este tipo de interfaces podrá realizar fácilmente un uso de ella después de recibir unas breves instrucciones. También facilita al usuario el manejo de los procedimientos. Otra de sus ventajas es que proporciona mecanismos estándar de control, como ventanas y cuadros de diálogo.

Por tanto este tipo de interfaces permiten al usuario interactuar con el sistema de una forma más fácil que otros sistemas.

Por otro lado tiene alguna que otra desventaja, cómo su dificultad de desarrollo frente a otro tipo de interfaces o también otro punto a tener en cuenta a la hora de elegir este tipo de interfaz para desarrollar una aplicación, es que utiliza demasiados recursos del sistema y puede no interesarnos por complejidad y coste.

También debemos mencionar un problema importante que tenemos con esta interfaz, que no nos sirve para realizar un control automático del robot. Simplemente nos sirve para enviar órdenes específicas al brazo robótico para que éste realice ciertos movimientos o posicione el efector final en una posición concreta.

La segunda interfaz que describiremos, será la **interfaz a través de una clase de Matlab.** Como ya hemos descrito anteriormente, un Objeto es una entidad que contiene información y un conjunto de acciones que operan sobre los datos.

Para que un objeto realice una de sus acciones, se le manda una petición de invocación a una función que pertenece a un objeto en particular. Por tanto, podemos decir que la ventaja principal de este tipo de interfaces a través del desarrollo de clases en Matlab es la *encapsulación* de datos y operaciones que presenta y por tanto la inclusión en otros programas de Matlab (ficheros .m).

Otra de sus ventajas, es que la escritura del código es más fácil que la de otras interfaces, debido en gran parte a que las propias definiciones de la clase nos estructuran el código, por lo que si hemos desarrollado una clase con muchos objetos o métodos, no tendremos problema en manejar un código extenso.

Por otro lado, entre las desventajas que presenta podemos destacar, que el hecho de tener un único bloque de programa puede complicarnos sobre todo la detección de fallos y errores.

Por último, nos queda comentar **la interfaz desarrollada a través de un bloque de Simulink**. Una de sus ventajas más destacables en general, es que posee un entorno interactivo con un conjunto de librerías con bloques personalizables que permiten simular, implementar y probar una serie de sistemas variables con el tiempo.

Como desventaja (en comparación con las demás interfaces), podríamos destacar que no es tan sencilla de manejar y entender como aquellas, debido a la configuración de los parámetros de los bloques y de los parámetros de simulación.

3.2 Interfaz a través de GUI

La interfaz desarrollada a través de la herramienta de Matlab GUIDE, es uno de los puntos clave que se desarrollan este proyecto.

Como ya conocemos el formato de datos que el cliente enviará al servidor (datagramas), pasaremos a explicar en detalle la interfaz y el entorno de funcionamiento de la misma.

Nuestra Interfaz Gráfica GUI, se compone de dos archivos: un archivo denominado Interfaz.m, en el que encontramos todo el código referente a la programación de los controles y el archivo Interfaz.fig, en el que encontramos los controles propiamente dichos.

Analizaremos los dos archivos a la par, ya que van ligados el uno al otro; es decir, si ponemos un control en nuestro archivo.fig en blanco, se generará una serie de instrucciones referentes a ese control seleccionado.

En la imagen siguiente podemos observar la apariencia final de la interfaz desde la pantalla de edición de la herramienta GUIDE. Ahí podemos hacer todas las modificaciones necesarias para conseguir la estética deseada para nuestra interfaz.

Además de modificaciones estéticas, también podemos realizar ciertas configuraciones y seleccionar que unciones que queremos implementar *(callbacks)* en el archivo.m.

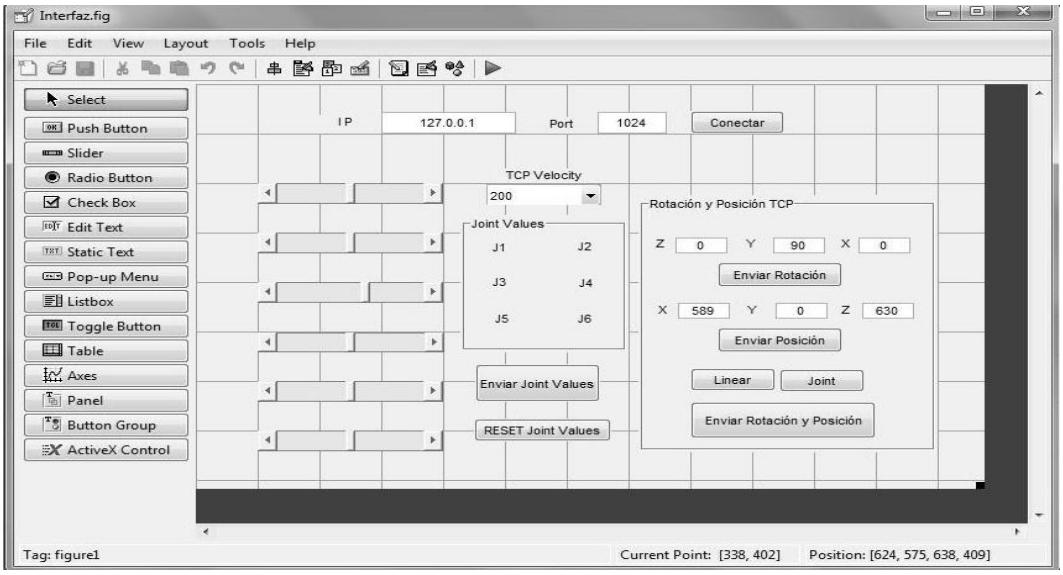


FIGURA 3.2 Interfaz Gráfica en pantalla de edición GUIDE

A partir de la figura anterior, comenzaremos a explicar los diferentes controles utilizados y el código implementado para cada uno de ellos.

IP y Port

En estos dos cuadros de texto, se le solicita al usuario que introduzca la dirección IP y el Puerto para realizar la comunicación TCP/IP con el servidor.

A continuación, analizaremos el código empleado. La funcionalidad de cada línea de código será comentada e irá precedida de % en letra cursiva para mejor visualización. Tenemos que mencionar también, que se ha hecho uso de variables globales, debido a que se deben utilizar ciertas variables para diversos controles y como ya sabemos, las variables utilizadas en una función (y no declaradas como globales), son consideradas por el programa como locales y por tanto no serían detectadas por otras funciones de nuestro código.

```
function edit1_Callback(hObject, eventdata, handles)
%Definición de variables globales utilizadas para este control.
global IP
global t

global Port
IP = get(hObject,'String');  % guardamos en la variable IP el valor del edit text que introduzca el usuario
guidata(hObject, handles);   %Salvar datos de la aplicación
```

Conectar

El push button “Conectar” se encarga de crear la variable t,que es el manejador (handle) de la conexión TCP/IP. Esta variable se utiliza para realizar la comunicación mediante el protocolo TCP/IP con sus correspondientes comandos. El código asociado a este control se muestra a continuación:

```
function pushbutton1_Callback(hObject, eventdata, handles)
%Definición de variables globales utilizadas para este control.
```

```
global t
global IP
global Port

% Establecemos la conexión mediante el protocolo TCP/IP utilizando las funciones de la toolbox de Matlab correspondiente.
t=tcpip(IP, Port); fopen(t);
msgbox('La Conexión se ha establecido'); % Abrimos un cuadro de diálogo al haberse establecido de forma correcta la conexión
guidata(hObject, handles); %Salvar datos de la aplicación
```

Barras de desplazamiento J_i

Las barras de desplazamiento (“Sliders”), se utilizan para que el usuario elija el número de grados que desea mover cada una de las seis articulaciones que posee el robot. El código correspondiente al primer slider se muestra a continuación:

```
function slider1_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.
global J1
global sj1

% cogemos el valor del slider y nos quedamos con su valor absoluto y su signo para asignárselo a las variables J y sj
    ang =get(hObject,'Value');
    signo = sign(ang);
    J1 = uint8(round(abs(ang)));
    if signo == -1
       sj1 = 0;
    else
       sj1=1;
     end
```

```
    angulo =round(ang);

% actualizamos el valor del static text     correspondiente a este Joint Value
    set(handles.text2,'String',angulo);
    guidata(hObject, handles); %Salvar datos de la aplicación
```

Nota: Para los otros cinco valores de Joint restantes, se utiliza el mismo fragmento de código. Para evitar la redundancia, omitiremos la descripción de los demás.

Enviar Joint Values

El push button "Enviar Joint Values" se encarga de crear el "Datagrama de Valores Articulares" y enviarlo por el socket. Para ello hace uso tanto de los valores obtenidos a través de los sliders, como de la variable global t, creada para establecer la comunicación TCP/IP. El código perteneciente a este botón es el siguiente:

```
function pushbutton2_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.
global J1
global J2
global J3
global J4
global J5
global J6
global t
global sj1
global sj2
global sj3
global sj4
global sj5
global sj6
```

```
 % Construimos el datagrama
D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5 sj6 J6]);


%Enviamos el datagrama realizando una pausa de dos segundos, para dar tiempo al robot a realizar la instrucción ordenada.
fwrite(t, D1);
pause(2);
guidata(hObject, handles); %Salvar datos de la aplicación
```

Reset Joint Values

Este control se ha implementado para llevar al robot a una posición de reposo y facilitar al usuario un posible reseteo de las posiciones de las diferentes articulaciones del robot.

El código que se presenta a continuación realiza las funciones descritas anteriormente:

```
function pushbutton3_Callback(hObject, eventdata, handles)
%Definición de variables globales utilizadas para este control.
global J1
global J2
global J3
global J4
global J5
global J6
global sj1
global sj2
global sj3
global sj4
global sj5
global sj6
global t
% Enviamos el robot a la posición de Reposo
```

```
set(handles.slider1,'Value',0) ; set(handles.text2,'String',0);
set(handles.slider2,'Value',0);  set(handles.text3,'String',0);
[…]
%Y así sucesivamente con los demás valores Joint del robot.
sj1=1; J1=0; sj2=1; J2=0; sj3=1; J3=0; sj4=1; J4=0; sj5=1; J5=0; sj6=1;J6=0;
% asignamos los varales  de reset y enviamos el datagrama

D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5 sj6 J6]);
fwrite(t, D1);
pause(2);
guidata(hObject, handles); % Salvar datos de la aplicación
```

Cuadros de texto editable Z, Y, X

En este cuadro de texto se le solicita al usuario que introduzca los valores en grados de los ángulos de Euler Z, Y, X que desea que gire el efector final del robot.

El código asociado al cuadro de texto Z es el siguiente:

```
function edit3_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.
global sz
global Rz
% Obtenemos de los cuadros de texto las variables de Rotación para construir el datagrama
vble = str2double(get(hObject,'String'));
signo = sign (vble);
if signo == -1
   sz=0;
else
   sz=1;
end
```

```
Rz= uint8(round(abs(vble))); % Obtenemos la variable de rotación Rz.
guidata(hObject, handles); %Salvar datos de la aplicación
```

Para el resto de variables de rotación (Y, X) se utilizan las mismas instrucciones, siendo la única diferencia, la denominación de las variables intermedias y de las variables finales obtenidas.

Cuadro de texto editable X, Y, Z

Este control es similar al anterior con la diferencia de que en este caso al usuario se le solicita la posición del efector final (en mm) en vez de la rotación.

El código asociado se presenta a continuación:

```
function edit6_Callback(hObject, eventdata, handles)
%Definición de variables globales utilizadas para este control.
global spx
global X
global xr

%Obtenemos las variables en este caso correspondientes a la coordenada x para construir el datagrama
x=str2double(get(hObject,'String'));
xr=rem(x,100);
X=floor(x/100);
signo = sign (x);

if signo == -1
   spx=0;
else
spx=1;
```

```
end
guidata(hObject, handles); %Salvar datos de la aplicación
```

Para el resto de variables de posición (Y,Z) se utilizan las mismas instrucciones, siendo la única diferencia la denominación de las variables intermedias y de las variables finales obtenidas.

Enviar rotación, Enviar Posición y Enviar rotación y posición

Para estos controles nos bastará con explicar uno sólo de ellos, ya que el resto son similares.

Cada uno de los *push button* nombrados anteriormente tiene como función crear el datagrama correspondiente y enviarlo por el socket. El push button " Enviar rotación" creará y enviará el datagrama dos, el push button "Enviar posición" el datagrama tres y el push button "Enviar rotación y posición" se encargará de crear y enviar el datagrama cuatro (que como ya sabemos es una mezcla de los datagramas dos y tres; es decir, posee variables tanto de rotación como de posición).

El código del control correspondiente a la orden "Enviar la rotación del TCP" es el siguiente:

```
function pushbutton4_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.
global sz
global sx
global sy
global Rz
global Rx
global Ry
global t

D2=uint8([2 sz Rz sy Ry sx Rx]); % Construimos el datagrama
```

```
fwrite(t, D2); %Enviamos datagrama realizando una pausa de 2 segundos entre medias
pause(2);
guidata(hObject, handles); %Salvar datos de la aplicación
```

TCP Velocity

Este control es un "pop up" menu, es decir, una lista desplegable en la que podremos elegir la velocidad del efector final.

Para este datagrama no es necesario un control externo que se encargue de crear y enviar el datagrama, lo realizamos todo en el mismo control.

El cógido asociado a este control se muestra a continuación:

```
function popupmenu2_Callback(hObject, eventdata, handles)

%definimos las variables globales utilizadas en este control.
global t;
global Vt;
global Vtr;
global Vr;
global Vrr;
sxl=1; xl=0; xlr=0; szl=1; zl=0; zlr=0;
vel= get(hObject,'Value'); % Ajustamos valores de velocidad a partir del control del popup menu
Vtr=rem(vel,100);
Vt=floor(vel/100);
Vrr=rem(vel,100);
Vr=floor(vel/100);

D5=uint8([5 1 Vt Vtr Vr Vrr sxl xl xlr szl zl zlr]);
% construimos y enviamos en datagrama.

fwrite(t,D5);
```

```
pause(2);
guidata(hObject, handles); %Salvar datos de la aplicación
```

En la siguiente imagen, podemos ver la apariencia final de la interfaz (la que verá el usuario) tras hacer clic en el botón de ejecutar.

Tras conocer el funcionamiento de la interfaz, procedemos a realizar una demostración gráfica. En las siguientes imágenes observaremos la apariencia final de nuestra interfaz gráfica GUI.

En figuras posteriores podremos observar el funcionamiento de la interfaz, para la que se ha hecho uso de la Estación de RobotStudio creada en el TFG [1] in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station" [1] para poder así comprobar que el brazo robot realiza correctamente las instrucciones que le enviamos desde nuestra interfaz.

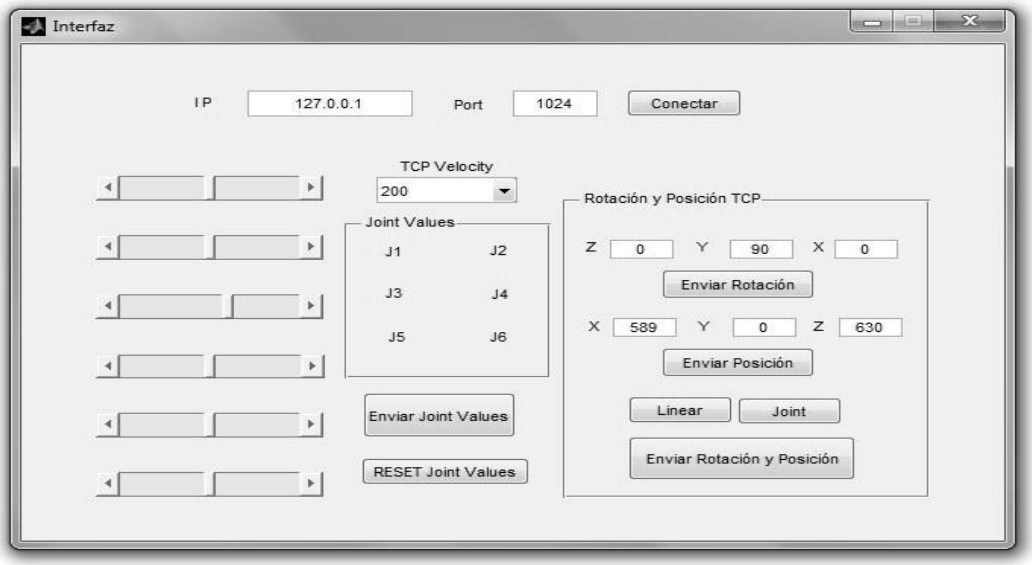


FIGURA 3.3 Apariencia final Interfaz GUI

Paso 1 de la demostración de la Interfaz GUI:

- *Establecemos conexión con la estación de RobotStudio*

Paso 2 de la demostración de la Interfaz GUI:

- *Elegimos unos valores de Joint y se los mandamos al Robot*

Paso 3 de la demostración de la Interfaz GUI:

- Elegimos unos valores de Rotación del TCP y se los mandamos al Robot

Paso 4 de la demostración de la Interfaz GUI:

- *Elegimos unos valores de Posición del TCP y se los mandamos al Robot*

Paso 5 de la demostración de la Interfaz GUI:

- *Elegimos unos valores de Rotación y Posición del TCP y se los mandamos al Robot de manera simultánea*

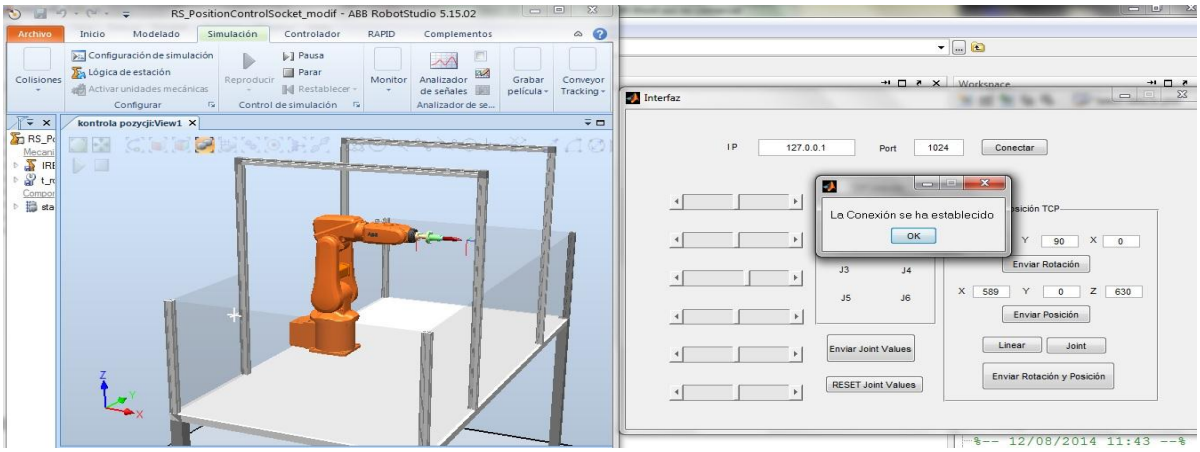


FIGURA 3.4 Paso 1 de la demostración de la Interfaz GUI

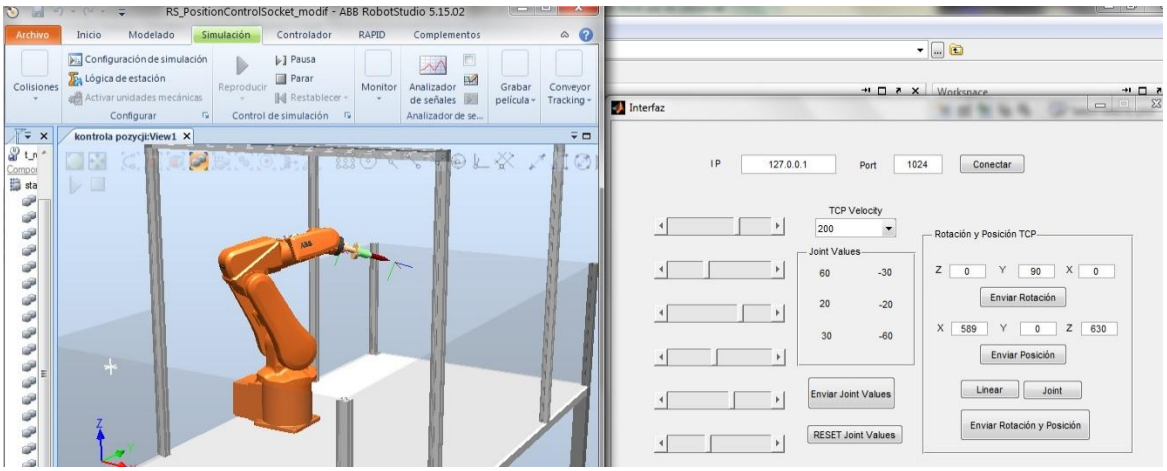


FIGURA 3.5 Paso 2 de la demostración de la Interfaz GUI:

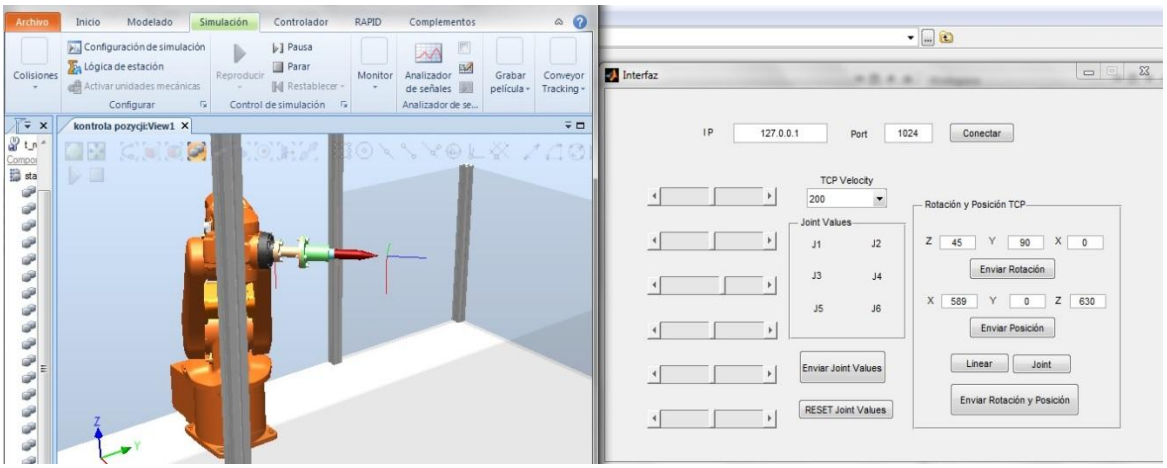


FIGURA 3.6 Paso 3 de la demostración de la Interfaz GUI:

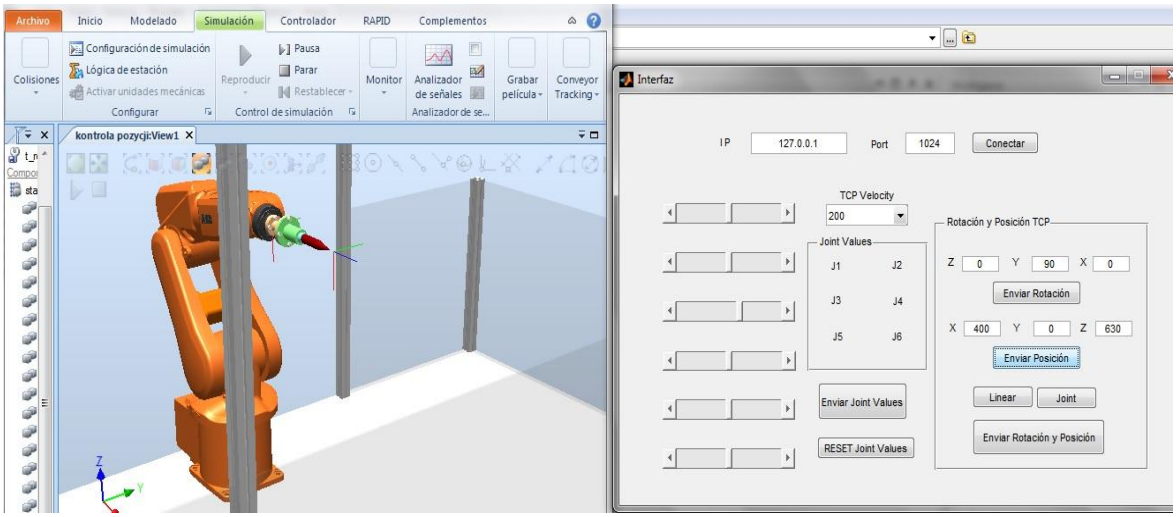


FIGURA 3.7 Paso 4 de la demostración de la Interfaz GUI:

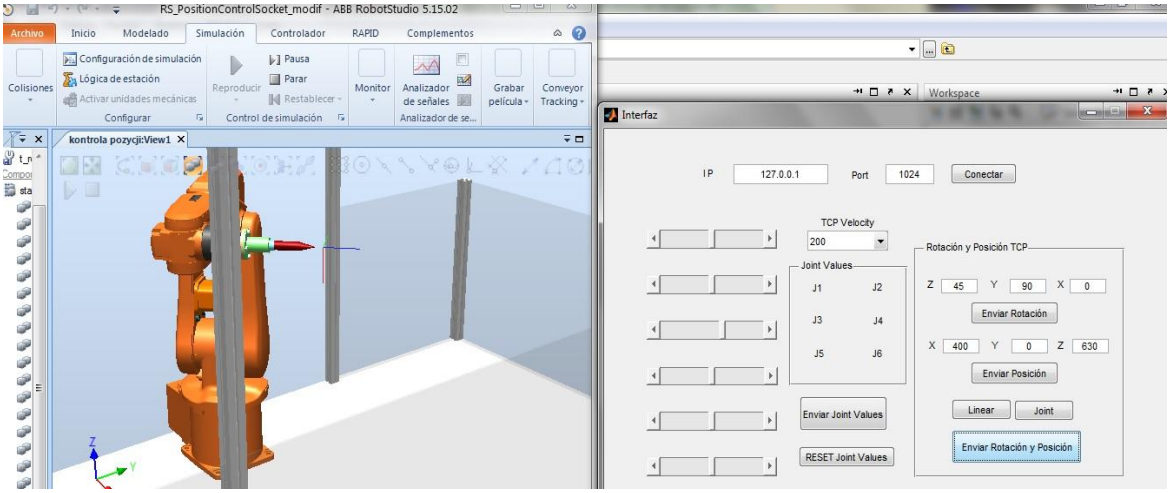


FIGURA 3.8 Paso 5 de la demostración de la Interfaz GUI:

3.3 Interfaz a través de clases en Matlab

El objetivo principal de esta interfaz a través de la creación de clases en Matlab, es facilitar al usuario el manejo de la comunicación con el brazo robot y también poder utilizar la clase creada para usos posteriores desde cualquier fichero .m de Matlab. Como ya sabemos, la creación de clases en Matlab puede ser más o menos complicada, pero su utilización es de lo más sencilla para el usuario. De ahí que hayamos elegido este método para crear nuestra interfaz.

Por tanto, para desarrollar esta interfaz, hemos definido una clase denominada "irb120". Esta clase posee una serie de propiedades y unos métodos, compuestos por funciones que serán los encargados de realizar las acciones asociadas a esta clase.

En primer lugar, utilizando el comando habitual, hemos definido la clase:

```
classdef irb120 < handle
```

Nuestra clase pertenece a la clase handle, que es la clase superior en Matlab por definición. No ha sido necesario implementar ningún tipo de subclases para esta interfaz, con la clase principal ha sido suficiente.

A continuación, si seguimos analizando el código desarrollado, nos encontramos con la definición de las propiedades:

```
Properties
    IP;
    port;
    conexion;
end
```

Las propiedades las utilizamos para asignar valores a los campos que posee nuestra clase y que son necesarios para que las funciones definidas posteriormente los utilicen.

A partir de este punto, entramos en el bloque de los métodos en el que tenemos definidas una serie de funciones que comentaremos a continuación:

- **Función que obtiene los valores de IP y Puerto**

Esta función es el denominado constructor de la clase. La clase la creamos escribiendo el nombre de la misma e introduciendo los valores que sean necesarios para su creación. Más adelante observaremos un ejemplo de cómo crear la clase paso a paso.

A continuación mostramos el código correspondiente al constructor que crea la clase:

```
function r=irb120(ip,puerto)

        r.IP=ip;

        r.port=puerto;

end
```

Esta función se encarga de obtener el valor de la dirección IP elegida por el usuario y el valor del puerto por el que se va a establecer la conexión con el servidor.

Tras estas líneas de código, ya estaría creada nuestra clase y tendría el siguiente aspecto:

```
>> robot= irb120('127.0.0.1',1024)
```

FIGURA 3.9 Creación de la clase

```
robot =

  irb120 handle

  Properties:
          IP: '127.0.0.1'
        port: 1024
    conexion: []

  Methods, Events, Superclasses
```

FIGURA 3.10 Clase irb120

- **Función que realiza la conexión y la desconexión**

Esta función es la encargada de realizar la conexión o desconexión de la comunicación según requiera la aplicación.

El código correspondiente es el que aparece a continuación:

```
function connect(r)
        r.conexion=tcpip(r.IP, r.port);
        fopen(r.conexion);
end
function disconnect(r)
       fclose(r.conexion);
end
```

En estas dos funciones, abrimos y cerramos respectivamente la comunicación con el robot. Para realizar la conexión, basta con seguir la línea de comando básica de la comunicación TCP/IP en Matlab, ajustando los parámetros a los de nuestra clase.

El código ejecutado en la ventana de comandos de Matlab sería el siguiente:

```
>> robot.connect
```

FIGURA 3.11 Uso de funciones de la clase creada

- **Función que envía la configuración articular**

Esta función es la encargada de enviar los valores de los ángulos de cada una de las seis articulaciones del robot. Para explicar cómo se ha implementado, nos basta con analizar el primer valor de articulación, ya que los siguientes son similares salvo por la denominación de las variables (J1, J2…etc).

```
function jointvalues(r,q)

    J1=uint8(round(abs(q(1))));
```

```
      signo = sign(q(1));

         if signo == -1
            sj1 = 0;
         else
           sj1=1;
         end
                  […]
end
```

El usuario introduce desde la ventana de comandos de Matlab los seis ángulos necesarios para crear el datagrama de Valores Articulares en forma de vector. Este fragmento de código, recoge el primer elemento del vector, obtiene su valor absoluto y su signo y los guarda en las variables correspondientes (J1, sj1 respectivamente).

Para los valores de los ángulos restantes, el proceso es el mismo, pero accediendo a elementos sucesivos del vector de ángulos introducido por el usuario.

Una vez obtenidos todos los valores de los ángulos, creamos y enviamos el datagrama:

```
D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5 sj6 J6]);

%Enviamos el dato
fwrite(r.conexion,D1);
```

La manera de ejecutar en Matlab esta función es como se muestra en la figura siguiente:

```
>> robot.jointvalues([20 -40 0 0 0 0])
```

FIGURA 3.12 Uso de funciones de la clase creada

Con la instrucción anterior estaríamos solicitando al robot que realice un giro de 20 grados positivos en la primera articulación, y otro de cuarenta grados negativos en al segunda articulación. El resto de articulaciones se quedarían en la posición de reposo.

- **Función que envía la Rotación del TCP**

Como su nombre indica, esta función se encarga de enviar los valores de los ángulos de rotación del TCP. Al igual que ocurre con la configuración articular, el usuario introduce en la ventana de comandos un vector de tres elementos correspondientes a los tres ángulos Z, Y, X (en ese orden y en grados).

Analizamos el código referente al ángulo Z y obviamos por similitud el código de los restantes. Tras obtener los tres valores, se crea y envía el datagrama.

```
function TCProtation(r,zyx)

    Rz=uint8(round(abs(zyx(1))));

    signoz = sign (zyx(1));

      if signoz == -1

         sz=0;

      else

         sz=1;

      end

   […]

D2=uint8([2 sz Rz sy Ry sx Rx]);

fwrite(r.conexion, D2);
```

La manera de ejecutar correctamente ese comando en Matlab sería la siguiente:

```
>> robot.TCProtation([45 0 90])
```

FIGURA 3.13 Uso de funciones de la clase creada

Con la instrucción anterior estaríamos pidiéndole al robot que gire 45 grados positivos respecto a su eje Z. Los otros dos ángulos se quedarían en la posición de reposo.

- **Función que envía los valores de posición del TCP**

Para la posición del TCP, se realiza el mismo proceso que para la rotación: un vector de tres elementos (en este caso la posición viene dada en mm) y el código que veremos a continuación (correspondiente a la coordenada X) se encargará de seleccionar el primer elemento de ese vector y obtener la posición X, Y, Z del TCP.

```
function TCPposition(r,xyz)
    xr=rem(xyz(1),100);
     X=floor(xyz(1)/100);
     signox = sign (xyz(1));

          if signox == -1
             spx=0;
          else
             spx=1;
          end
[…]

   D3=uint8([3 spx X xr spy Y yr spz Z zr 0]);

   fwrite(r.conexion,D3);
```

Con la instrucción que se muestra a continuación, realizamos un cambio de la posición del efector final. En concreto estaríamos pidiéndole al robot que se desplace 400 mm en X (positivo), y que mantenga el reposo para sus otras dos coordenadas.

```
>> robot.TCPposition([400 0 630])
```

FIGURA 3.14 Uso de funciones de la clase creada

- **Función que envía la rotación y la posición del TCP**

Esta última función, al igual que el datagrama del mismo nombre, se encarga de enviar de manera simultánea la rotación y la posición del TCP. El usuario introduce en la ventana de comandos de Matlab un vector en este caso de seis elementos: [rotación Z, rotación Y, rotación X, posición X, posición Y, posición Z].

El código de esta función, es un conjunto de las dos anteriores. La creación y envío del datagrama se realiza de la misma manera que los demás datagramas.

```
function TCProtandpos(r,tcp)
[…]

D4=uint8([4 sz Rz sy Ry sx Rx spx X xr spy Y yr spz Z zr]);

fwrite(r.conexion,D4);
```

En la figura siguiente ejecutaremos el comando realizando las dos instrucciones anteriores de manera simultánea:

```
>> robot.TCProtandpos([45 0 90 400 0 630])
```

FIGURA 3.15 Uso de funciones de la clase creada

A continuación, veremos un ejemplo gráfico a través de imágenes del funcionamiento de esta interfaz junto con el simulador de RobotStudio para poder comprobar los resultados con el Robot.

Paso 1 de la demostración de la Interfaz a través de clases en Matlab:

- *Creamos la clase, establecemos la conexión, y enviamos un valor de Joint*

Paso 2 de la demostración de la Interfaz a través de clases en Matlab:

- *Enviamos un valor de Joint para cada uno de los ángulos*

Paso 3 de la demostración de la Interfaz a través de clases en Matlab:

- *Elegimos unos valores de Rotación del TCP y se los mandamos al Robot*

Paso 4 de la demostración de la Interfaz a través de clases en Matlab:

- *Elegimos unos valores de Posición del TCP y se los mandamos al Robot*

Paso 5 de la demostración de la Interfaz a través de clases en Matlab:

- *Elegimos unos valores de Rotación y Posición del TCP y se los mandamos al Robot de manera simultánea*

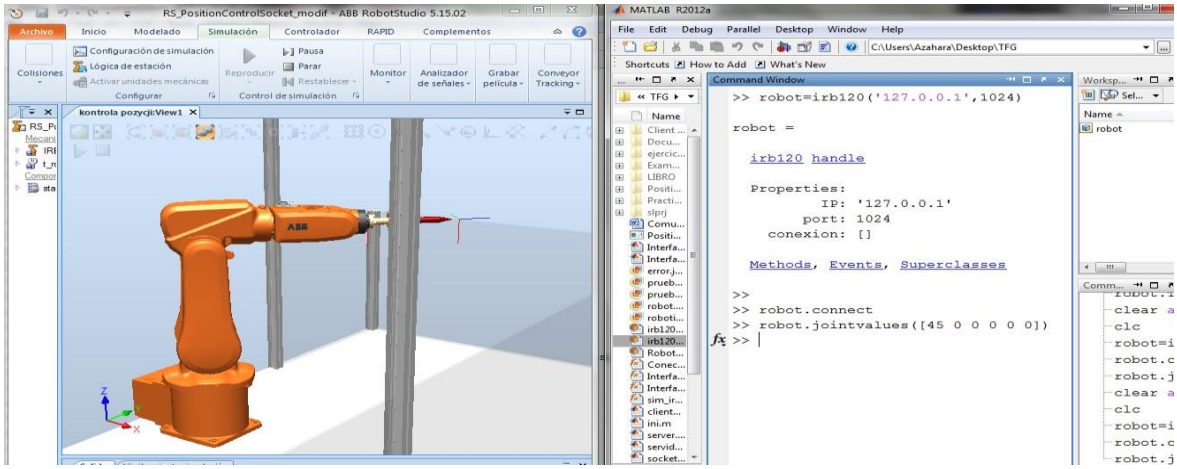


FIGURA 3.16 Paso 1 de la demostración de la Interfaz a través de clases en Matlab

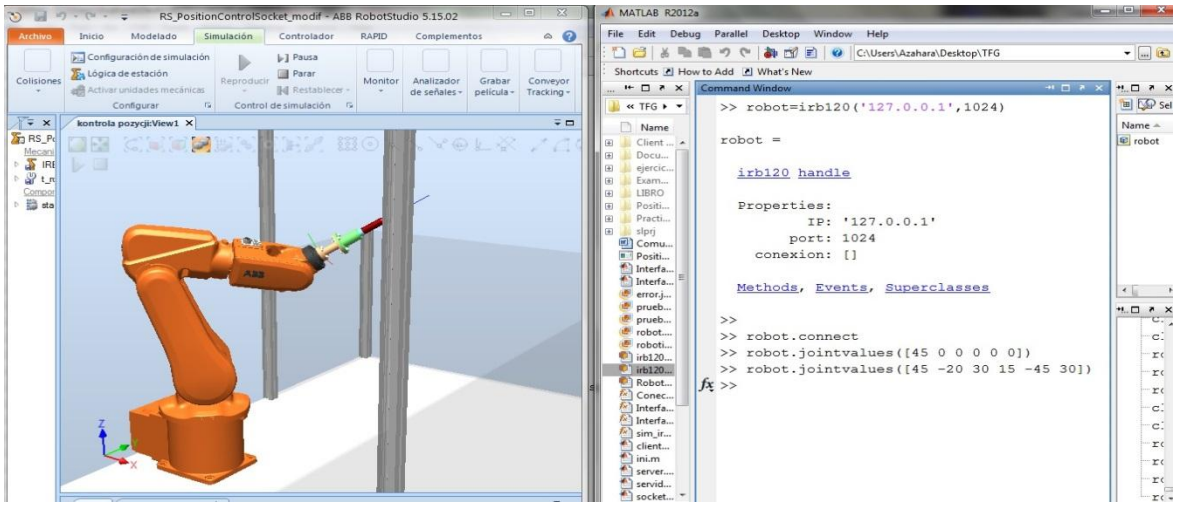


FIGURA 3.17 Paso 2 de la demostración de la Interfaz a través de clases en Matlab

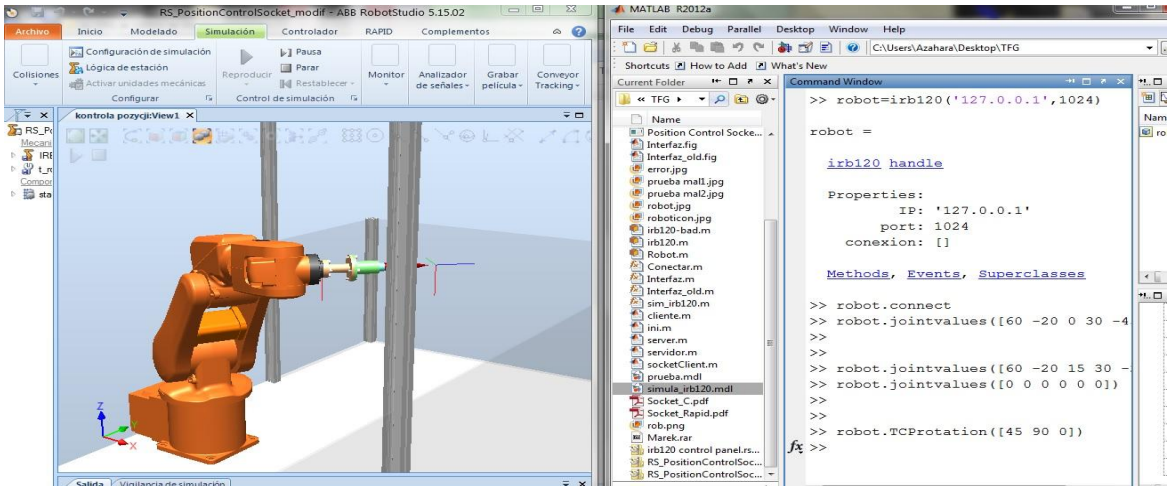


FIGURA 3.18 Paso 3 de la demostración de la Interfaz a través de clases en Matlab

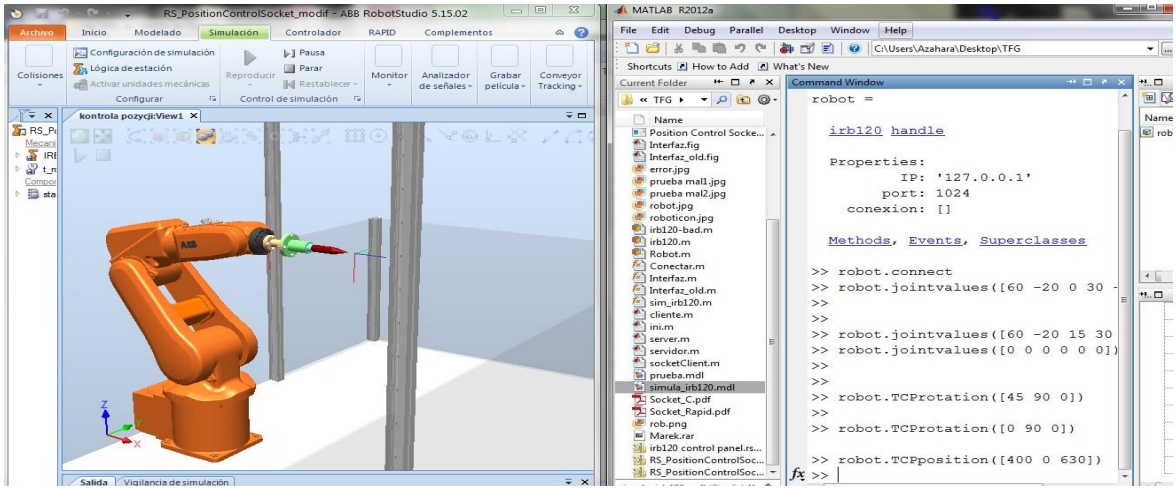


FIGURA 3.19 Paso 4 de la demostración de la Interfaz a través de clases en Matlab

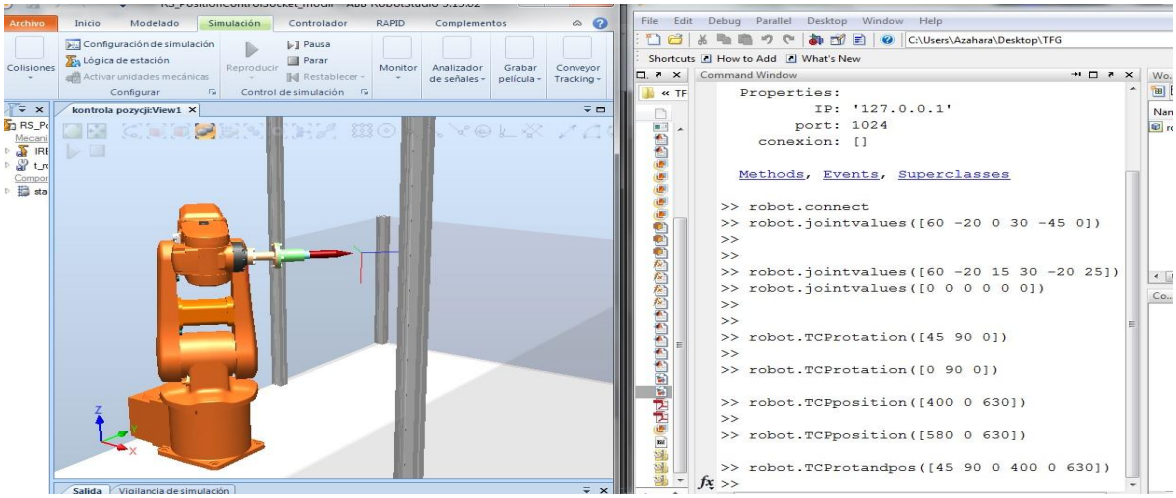


FIGURA 3.20 Paso 5 de la demostración de la Interfaz a través de clases en Matlab

3.4 Interfaz a través de un bloque de Simulink

Para la realización de esta interfaz, utilizaremos la herramienta de Matlab, Simulink y un archivo de inicialización .m.

Esta interfaz consta por tanto, de dos archivos principales: el archivo .m de inicialización y el archivo .mdl en el que se encuentra el bloque de Simulink creado. En el primero, definimos una serie de variables globales que se utilizarán después y en el archivo de Simulink encontramos el bloque que hemos creado para conectar con el robot y enviarle los datos que creamos convenientes.

También debemos mencionar que en el archivo de inicialización, se realiza la apertura de la comunicación con el servidor. A través de Simulink, procesamos y enviamos la información.

En la siguiente figura, observamos el contenido del archivo de Simulink creado. Consta de dos elementos: el primero es un vector de constantes en el que introduciremos los valores de los ángulos Joint que queremos enviar al robot y en el segundo encontramos el bloque irb120 creado.

El bloque irb120 puede utilizarse en cualquier otro diagrama de bloques de Simulink.

Esta interfaz ha sido diseñada para realizar sólo el envío de la configuración articular del robot.

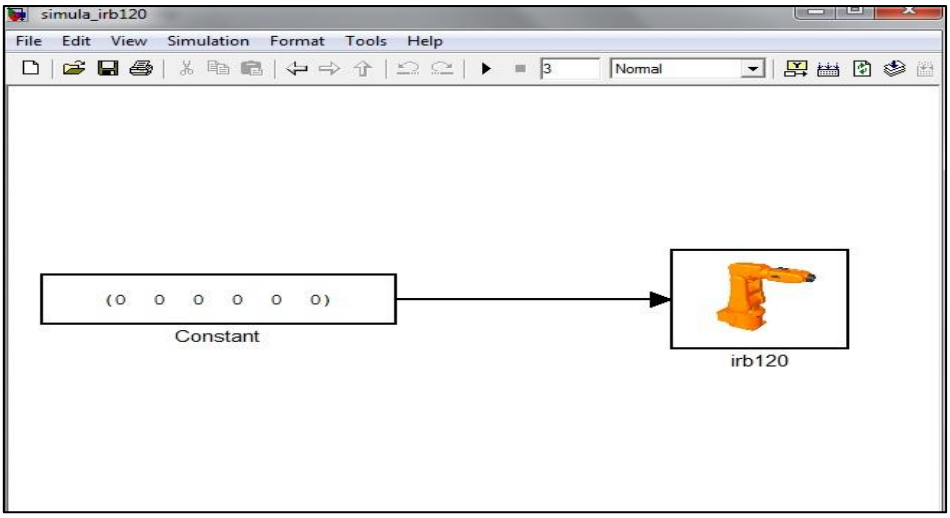


FIGURA 3.21 Contenido del archivo .mdl

El bloque mencionado anteriormente ha sido enmascarado y posee en su interior un subsistema formado a su vez por otro bloque denominado "*Interpreted Matlab Function*".

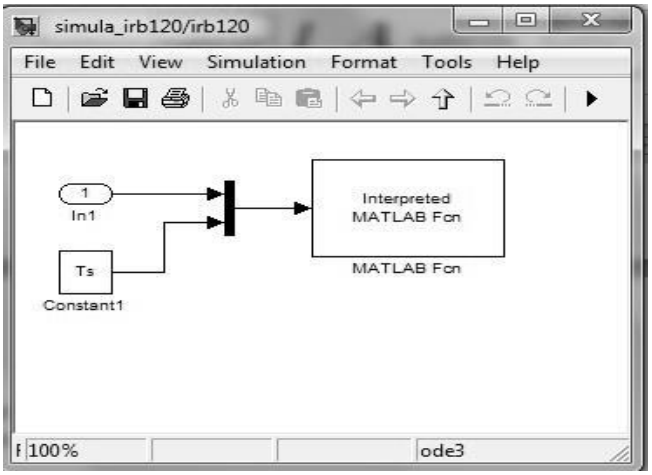


FIGURA 3.22 Contenido del interior de la máscara del bloque irb120

El bloque *Interpreted Matlab Function* posee un archivo .m asociado, en el que hemos implementado un código que realiza el procesamiento de los datos, la creación del datagrama y el envío del mismo.

El parámetro *Ts* (Time Step), lo utilizaremos para unificar el tiempo de muestreo de cada bloque y para ajustar los tiempos de simulación de nuestro programa. Gracias al enmascaramiento del bloque, accedemos de forma rápida y sencilla a este parámetro simplemente con hacer doble clic sobre el bloque irb120.

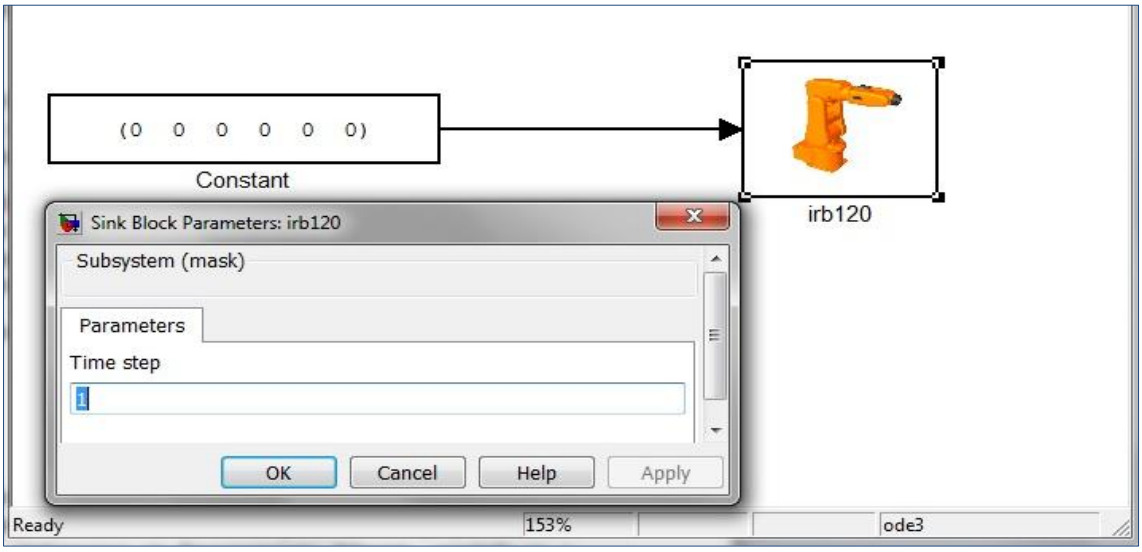


FIGURA 3.23 Parámetro Ts de la máscara

En la imagen anterior podemos comprobar la manera de acceder al parámetro “Time Step”.

El desarrollo actual de esta interfaz, implica que el bloque de Simulink obtiene la configuración articular directamente del vector de constantes que le estamos proporcionando a su entrada.

En apartados posteriores, analizaremos un funcionamiento diferente, orientado ya a nuestra aplicación final de pintar números.

El bloque de Simulink recibirá la configuración articular de un conjunto de bloques con el que realizamos un control diferencial.

A continuación, el funcionamiento de esta interfaz:

Paso 1 de la demostración de la Interfaz a través de Bloque de Simulink:

- *Ejecutamos el archivo de inicialización y se establece la comunicación*

Paso 2 de la demostración de la Interfaz a través de Bloque de Simulink:

- *Elegimos un vector de valores y ejecutamos nuestro .mdl*

Paso 3 de la demostración de la Interfaz a través de Bloque de Simulink:

- *Modificamos el vector de valores y añadimos más ángulos*

Paso 4 de la demostración de la Interfaz a través de Bloque de Simulink:

- *Volvemos a modificar los valores y ejecutamos el archivo comprobando el resultado*

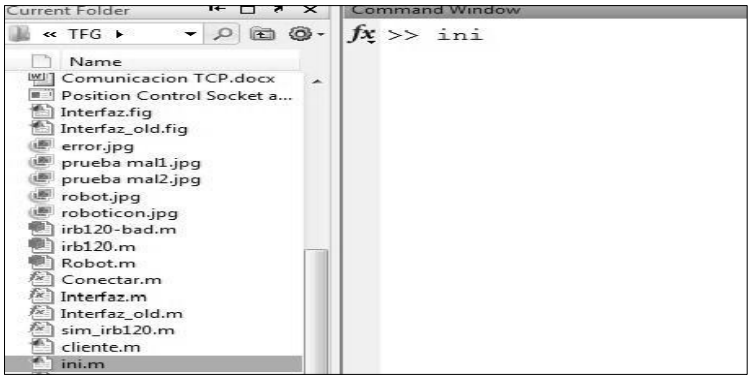


FIGURA 3.24 Paso 1 de la demostración de la Interfaz a través de Bloque de Simulink:

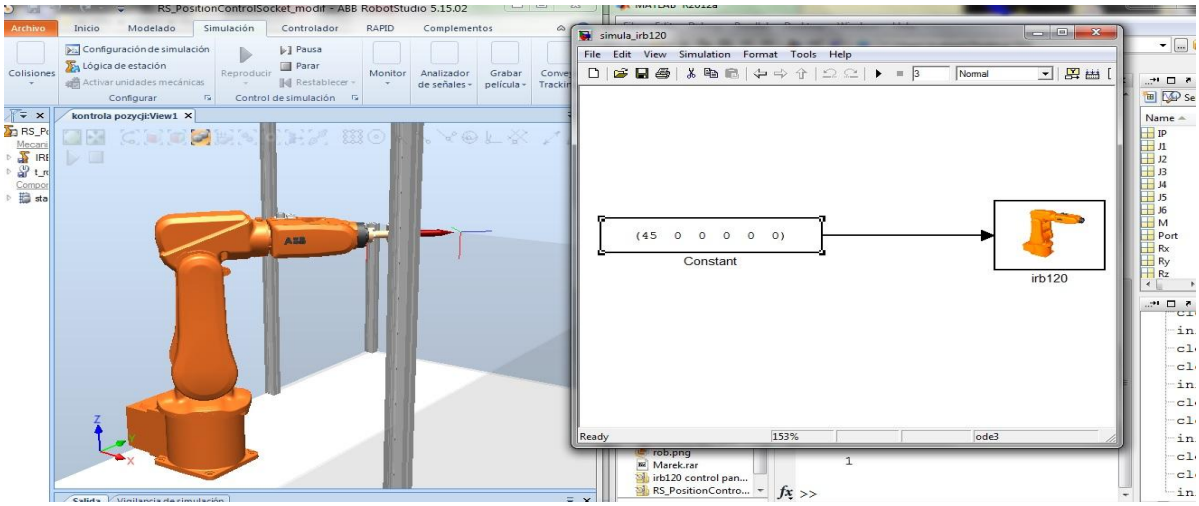


FIGURA 3.25 Paso 2 de la demostración de la Interfaz a través de Bloque de Simulink:

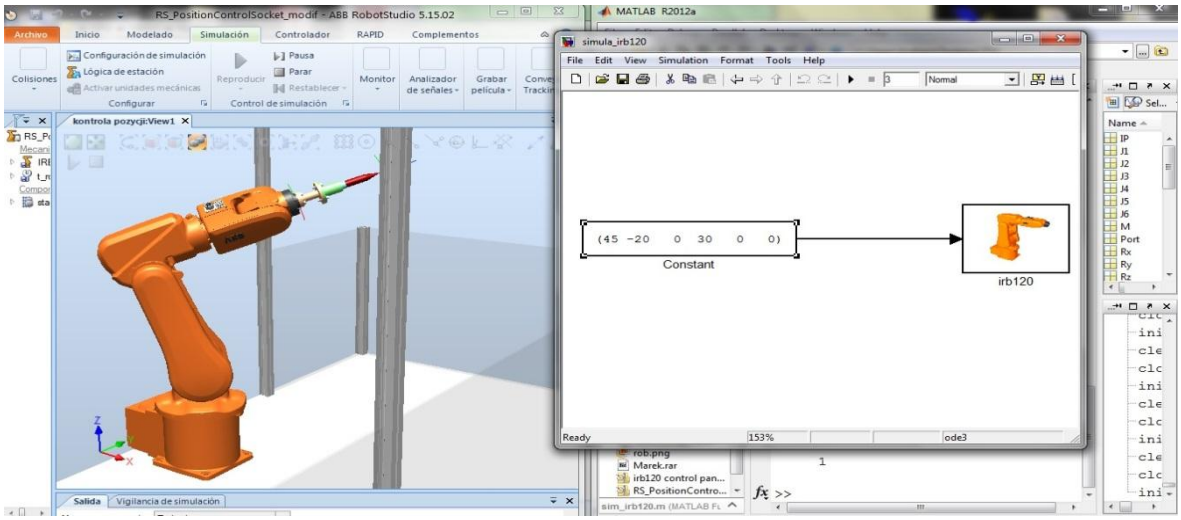


FIGURA 3.26 Paso 3 de la demostración de la Interfaz a través de Bloque de Simulink:

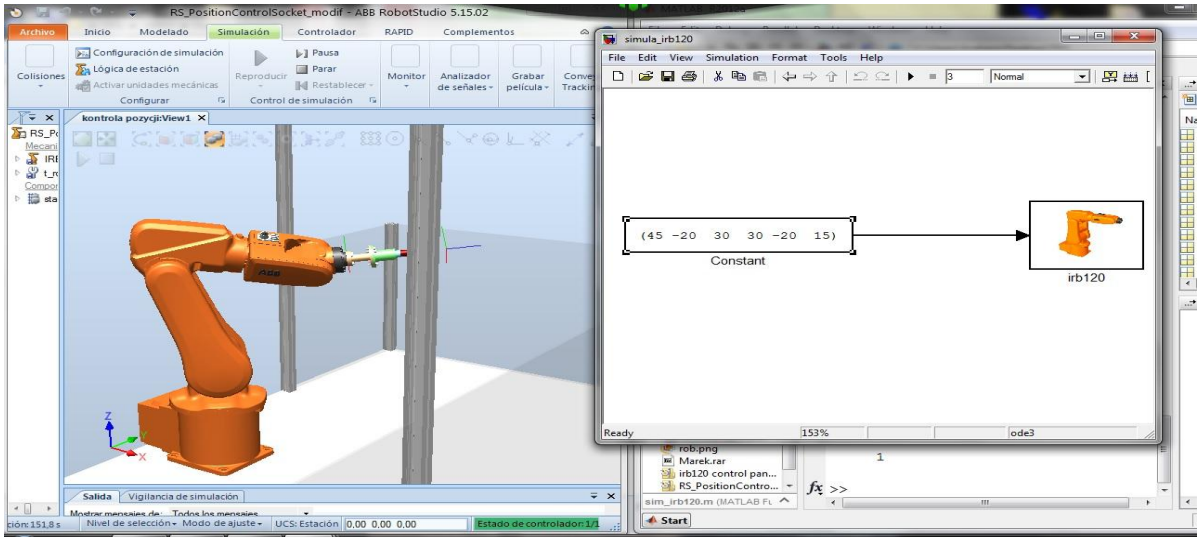


FIGURA 3.27 Paso 4 de la demostración de la Interfaz a través de Bloque de Simulink

4 Aplicación: trazado de números

4.1 Objetivo de la aplicación

El objetivo principal de esta aplicación, es conseguir que nuestro brazo robot pinte números del cero al nueve sobre un panel numérico creado, a petición del usuario, utilizando dos de las interfaces de comunicación desarrolladas: la interfaz a través de la creación de clases en Matlab y la interfaz a través del bloque "irb120 de Simulink.

En el desarrollo de esta aplicación se utilizarán los conceptos sobre modelado y control cinemático y dinámico de robots industriales, aprendidos durante la asignatura de "Sistemas Robotizados".

Para ello, se realizarán diferentes simulaciones en RobotStudio y también realizaremos pruebas en el robot real, con la *Toolbox de Robótica* instalada específicamente para esta asignatura. Como ya sabemos, el robot a modelar y simular es el modelo IRB120 de ABB.

Al realizar esta aplicación, conseguimos profundizar en conceptos de modelado cinemático y dinámico, generación de trayectorias y control de un robot industrial, al igual que conseguiremos demostrar el correcto funcionamiento de las interfaces desarrolladas como objetivo principal del proyecto.

4.2 Modelado del Robot IRB120 en Matlab

La **cinemática** de un brazo robot estudia la geometría del movimiento de un robot con respecto a un sistema de coordenadas fijo como una función del tiempo, sin considerar las fuerzas/momentos que originan dicho movimiento.

Esta cinemática trata de resolver dos tipos de problemas:

- El problema cinemático directo.
- El problema cinemático inverso.

✓ **Problema cinemático directo**

Su resolución permite conocer la posición (x,y,z) y orientación (ϕ, θ, ψ) del extremo final del robot, con respecto a un sistema de coordenadas de referencia, conocidos los ángulos/desplazamientos de las articulaciones (qi – variables articulares) y los parámetros geométricos de los elementos del robot.

Nuestro robot posee seis grados de libertad, por lo que tendremos seis variables articulares (q_1-q_6). En la siguiente figura podemos observar los ejes del robot irb120:

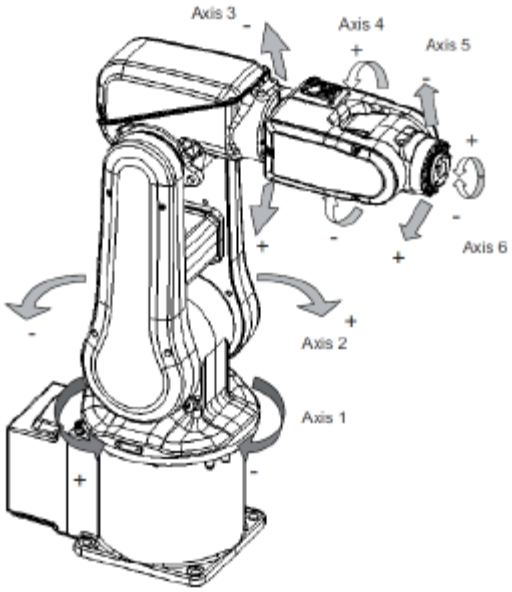


FIGURA 4.1 Ejes del robot IRB120

Para calcular la cinemática directa, necesitamos una serie de parámetros y sistemas de coordenadas de referencia que se obtienen de forma manual a través del Procedimiento de Denavit-Hartenberg (D-H) [2], que es un método matricial, que establece la localización que debe tomar cada sistema de coordenadas ligado a cada eslabón i de una cadena articulada, para poder sistematizar la obtención de las ecuaciones cinemáticas de la cadena completa.

Gracias al conocido algoritmo Denavit-Hartember, mencionado anteriormente, obtenemos la tabla de parámetros D-H que vemos a continuación:

ESLABÓN	MATRIZ	θ	d	a	α
1	A01	q1	l1	0	-π/2
2	A12	q2 – π/2	0	l2	0
3	A23	q3	0	-l3	-π/2
4	A34	q4	L4	0	π/2
5	A45	q5	0	0	-π/2
6	A56	q6+π	l5	0	0

TABLA 6: Tabla de parámetros D – H

Con estos parámetros y las dimensiones del robot (que veremos en la siguiente figura), podemos modelar nuestro robot en Matlab gracias a la clase *SerialLink de la "Toolbox de Robótica"*.

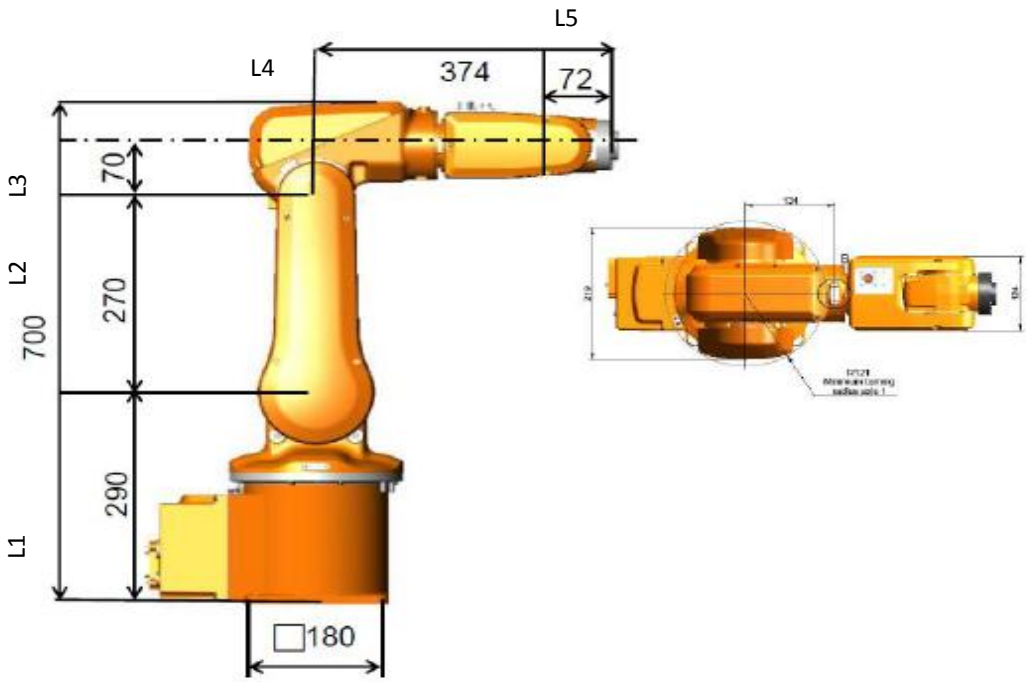


FIGURA 4.2 Robot en la posición de reposo. Dimensiones principales

Como ya hemos comentado, utilizando la clase *SerialLink* podemos realizar el modelado de nuestro robot en Matlab. El siguiente código muestra la forma de hacerlo:

```
L(1)=Link([0 l1 0 -pi/2]);

L(2)=Link([0 0 l2 0]);
```

```
L(3)=Link([0 0 l3 -pi/2]);
L(4)=Link([0 l4 0 pi/2]);
L(5)=Link([0 0 0 -pi/2]);
L(6)=Link([0 l5 0 0]);

irb120=SerialLink(L,'name','IRB120');
```

Utilizando la función "irb120.plot" ([q,s]) obtenemos los resultados mostrados en las siguientes imágenes, para diferentes configuraciones articulares del robot.

En la primera observamos al robot en la posición de reposo y en la segunda observamos el robot con una rotación de 45 º negativos en su primera articulación y 50º positivos en su segunda articulación, quedándose el resto de articulaciones en posición de reposo.

```
>> irb120.plot([0 0 0 0 0 0])
```

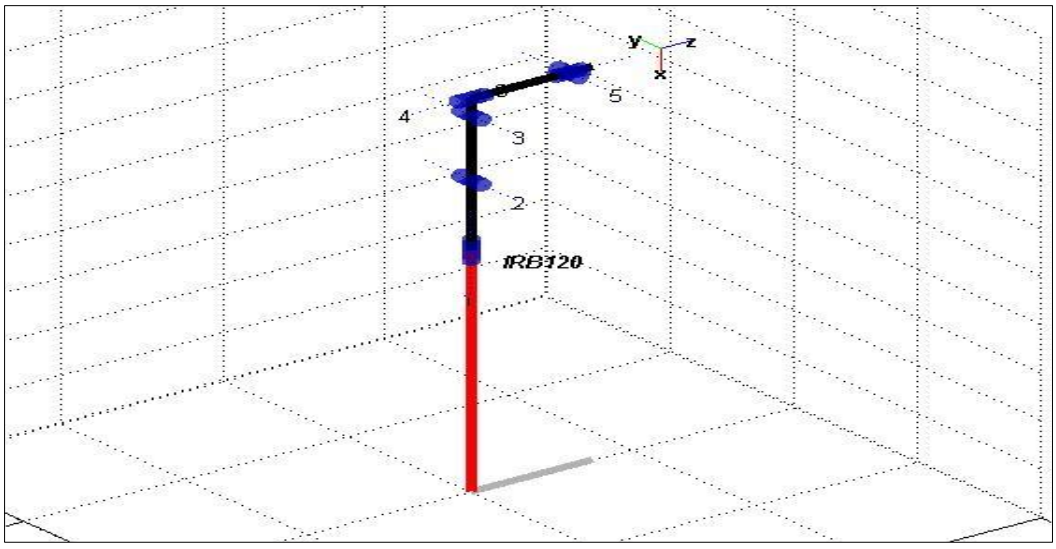


FIGURA 4.3 Modelo del robot en la posición de reposo

```
>> irb120.plot([-45 50 0 0 0 0])
```

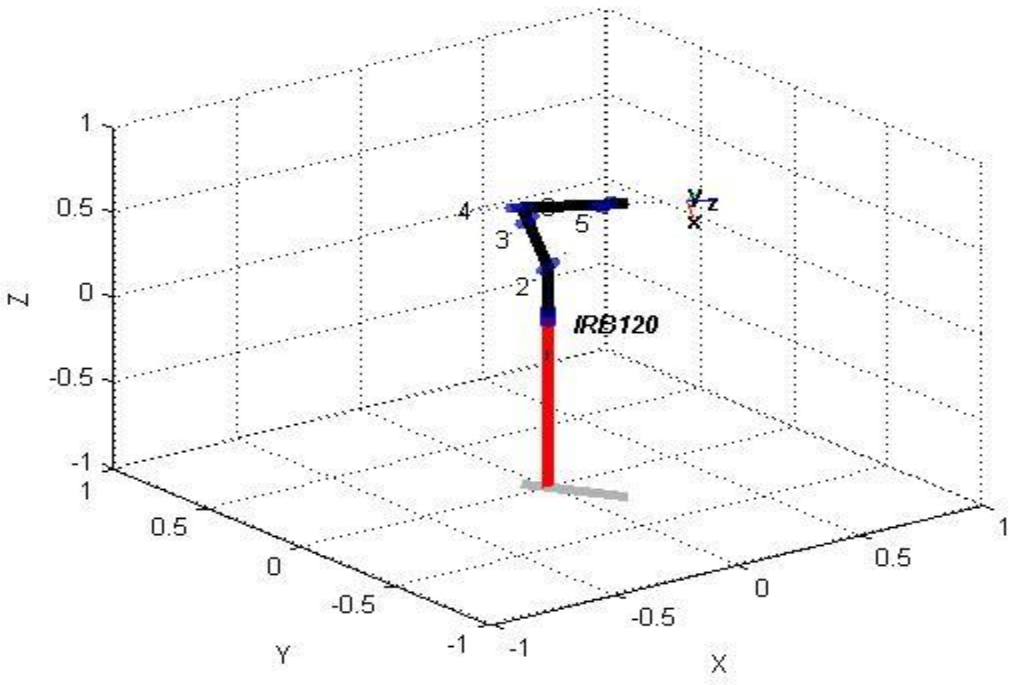


FIGURA 4.4 Modelo del robot en una posición en concreto

✓ **Problema cinemático Inverso**

El objetivo de este problema cinemático es encontrar los valores que deben adoptar las coordenadas articulares del robot $q = [q1, q2,\ldots qn]^T$ para que su extremo se posicione y oriente según una determinada localización espacial (p,[n, o, a]).

La resolución no es sistemática y depende de la configuración del robot, por lo que pueden existir múltiples soluciones.

La mayor parte de los robots poseen cinemáticas relativamente simples que facilitan la resolución del problema cinemático inverso. Algunas de ellas son:

–Tres primeros grados de libertad con estructura planar (elementos contenidos en un plano).

–Tres últimos grados de libertad (para orientar el extremo) corresponden a giros sobre ejes que se cortan en un punto.

Para el cálculo de la cinemática inversa, hacemos uso de una función de la "*Toolboxde Robótica"* que se denomina *"ikine",* la cual tiene una variante denominada *"ikine6s".*

Desgraciadamente ninguno de estos dos métodos funciona correctamente con nuestro robot. El problema es que el robot irb120 presenta un offset en el hombro que no contemplan las funciones descritas anteriormente.

Para solucionar este problema optamos por obtener las ecuaciones finales de la cinemática inversa mediante un proceso denominado desacoplo cinemático [2].

Las ecuaciones finales del cálculo del problema cinemático inverso se presentan a continuación:

Suponiendo que la posición del efector final viene dada por la matriz de transformación homogénea T, la posición de la muñeca puede calcularse conociendo el valor de L_5 (distancia entre la muñeca y el efector final) mediante la siguiente expresión:

$$pm = T \cdot [\,0 \;\; 0 \;\; -L_5 \;\; 1]' = [\,pmx \;\; pmy \;\; pmz \;\; 1]$$

La ecuación correspondiente a la primera articulación es:

$$q1 = atg\,\frac{pmy}{pmx}$$

Llamamos A, B, t, β y K1 a:

$$A = pmx \cdot \cos\, q1 \;\; + pmy \cdot sen(q1)$$

$$B = pmz - L_1$$

$$t = \frac{{L_2}^2 \; + \; {L_3}^2 - A^2 - B^2}{2 \cdot L_2 \cdot \quad \overline{{L_3}^2 + {L_4}^2}}$$

$$\beta = atg\frac{L_3}{L_4}$$

$$K_1 = \frac{L_2 + L_3 \cdot \cos\ q3\ - L_4 \cdot sen(q3)}{L_4 \cdot \cos\ q3\ + L_3 \cdot sen(q3)}$$

Siendo L_i las dimensiones mostradas en la figura 4.2.

Las ecuaciones correspondientes a las articulaciones 2 y 3 son:

$$q2 = atan2\ \frac{A \cdot K_1 - B}{A + B \cdot K_1}$$

$$q3 = atg\ \frac{t}{\sqrt{1 - t^2}} + \beta$$

Una vez conocidos los valores de las articulaciones (q1, q2, q3) y por tanto la rotación entre la primera y la tercera articulación (R03), la rotación entre la tercera y la sexta articulación será:

$$R_{36} = (R_{03})^{-1} \cdot R_{06} = \begin{matrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{matrix}$$

Conocidos todos los datos, podemos obtener de manera sencilla los valores de las tres articulaciones restantes:

$$q4 = atg\ \frac{r_{23}}{r_{13}}$$

$$q5 = arcos\ (r_{33})$$

$$q6 = atg\frac{r_{32}}{-r_{31}}$$

A parte de realizar el modelado del robot, vamos a desarrollar un control cinemático para nuestro brazo robot. El control cinemático presenta, entre otras, las siguientes funciones:

- Convertir la especificación de movimiento dada en el programa, en una trayectoria analítica en espacio cartesiano.
- Muestrear la trayectoria cartesiana, obteniendo un número finito de puntos de dicha trayectoria
- Utilizar la cinemática inversa, convertiendo cada uno de estos puntos en sus correspondientes coordenadas articulares (q1, q2, q3, q4, q5, q6).
- Muestreo de la trayectoria articular para generar referencias al control dinámico.
- Interpolación de los puntos articulares obtenidos, generando para cada variable articular, una expresión qi(t) que pase o se aproxime a ellos

En apartados posteriores observaremos cómo se realiza la implementación de tres posibles variantes del control cinemático:

- ❖ *Control mediante interpolación articular*
- ❖ *Control mediante interpolación cartesiana*
- ❖ *Control diferencial*

4.3 Creación del panel numérico

En este apartado, vamos a desarrollar el panel numérico sobre el que el robot pintará los números que le solicitemos a través de la ventana de comandos de Matlab.

El esquema del panel podemos observarlo en la figura situada al final de la página.

Los vértices (7, 8, 9, 10,11 ,12), separados L/10 del plano en el que se localiza el panel, permiten que la herramienta se separe del panel para alcanzar las nuevas posiciones a partir de las cuales, pueda seguir trazando los números.

Al pintar, la herramienta se encontrará siempre perpendicular al plano del panel (con la misma orientación que el sistema {Sp}).

La posición del panel no es única, puede ser modificada según requiera la aplicación.

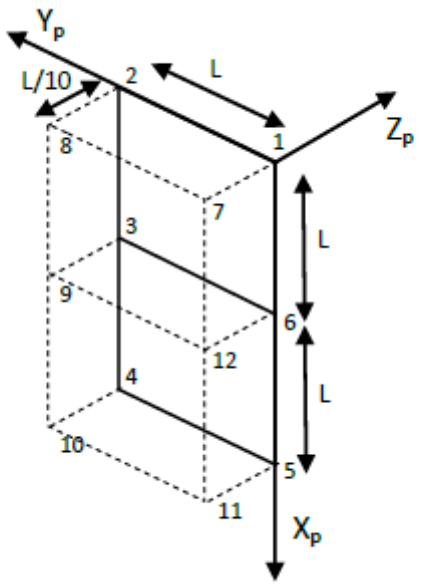


FIGURA 4.5 Panel numérico: sistema de referencia asociado y dimensiones.

A continuación, analizaremos el código necesario para realizar el panel numérico en Matlab. Los comentarios en letra cursiva y precedidos de % nos explican el significado de la línea de código que lo acompaña.

```
%Sistema de referencia del mundo, para poder dibujarlo

Origen[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
%Sistema de referencia del panel de números, que se puede modificar. Tenemos tres posibiliadades:
PosPanel=transl(0.3,0,0.7)*TRi;     %TRi es la rotación propia del efector final en la posición de reposo
PosPanel=transl(0.5,0,0.2)*TRi*troty(pi/2);  %TRi es la rotación propia del efector final en la posición de reposo
PosPanel=transl(0,0.5,0.6)*TRi*trotx(pi/2)*troty(pi/6)    %TRi es la rotación propia del efector final en la posición de reposo
```

```
% En nuestro caso el panel está colocado sobre el plano XY y paralelo al eje Z.
axis([-1 1 -1 1 -1 1]);hold on;grid on;
trplot(PosPanel,'color','r');
trplot(Origen);
%Escalado del panel
escala=0.1;
%Coordenadas homogéneas de los puntos en el sistema de referencia del panel.
%Normalizados
p1=escala*[0 0 0 1/escala]';
p2=escala*[0 1 0 1/escala]';
p3=escala*[1 1 0 1/escala]';
p4=escala*[2 1 0 1/escala]';
p5=escala*[2 0 0 1/escala]';
p6=escala*[1 0 0 1/escala]';
p7=escala*[0 0 -0.1 1/escala]';
p8=escala*[0 1 -0.1 1/escala]';
p9=escala*[1 1 -0.1 1/escala]';
p10=escala*[2 1 -0.1 1/escala]';
p11=escala*[2 0 -0.1 1/escala]';
p12=escala*[1 0 -0.1 1/escala]';

%Coordenadas de los puntos en el sistema de referencia total
p1T=PosPanel*p1;
p2T=PosPanel*p2;
p3T=PosPanel*p3;
p4T=PosPanel*p4;
p5T=PosPanel*p5;
p6T=PosPanel*p6;
p7T=PosPanel*p7;
p8T=PosPanel*p8;
p9T=PosPanel*p9;
p10T=PosPanel*p10;
p11T=PosPanel*p11;
```

```
p12T=PosPanel*p12;

%Dibujo de los puntos
plot3([p1T(1) p2T(1) p3T(1) p4T(1) p5T(1) p6T(1) p7T(1) p8T(1) p9T(1) p10T(1) p11T(1) p12T(1)],[p1T(2) p2T(2) p3T(2) p4T(2) p5T(2) p6T(2) p7T(2) p8T(2) p9T(2) p10T(2) p11T(2) p12T(2)],[p1T(3) p2T(3) p3T(3) p4T(3) p5T(3) p6T(3) p7T(3) p8T(3) p9T(3) p10T(3) p11T(3) p12T(3)],'go');
plot3([p1T(1) p2T(1) p3T(1) p4T(1) p5T(1) p6T(1) p1T(1)],[p1T(2) p2T(2) p3T(2) p4T(2) p5T(2) p6T(2) p1T(2)],[p1T(3) p2T(3) p3T(3) p4T(3) p5T(3) p6T(3) p1T(3)],'r--');
plot3([p3T(1) p6T(1)],[p3T(2) p6T(2)],[p3T(3) p6T(3)],'r--');

%Meto en un array las posiciones completas de cada vértice respecto al sistema de referencia global. Primero paso al sistema de referencia del panel y después aplico la traslación correspondiente a ese punto dentro de ese sistema de referencia.
T_Vertices(:,:,1)=PosPanel*transl(p1(1:3));
T_Vertices(:,:,2)=PosPanel*transl(p2(1:3));
T_Vertices(:,:,3)=PosPanel*transl(p3(1:3));
T_Vertices(:,:,4)=PosPanel*transl(p4(1:3));
T_Vertices(:,:,5)=PosPanel*transl(p5(1:3));
T_Vertices(:,:,6)=PosPanel*transl(p6(1:3));
T_Vertices(:,:,7)=PosPanel*transl(p7(1:3));
T_Vertices(:,:,8)=PosPanel*transl(p8(1:3));
T_Vertices(:,:,9)=PosPanel*transl(p9(1:3));
T_Vertices(:,:,10)=PosPanel*transl(p10(1:3));
T_Vertices(:,:,11)=PosPanel*transl(p11(1:3));
T_Vertices(:,:,12)=PosPanel*transl(p12(1:3));

%Ahora realizamos la DEFINICIÓN DE LOS NÚMEROS. Terminamos y empezamos siempre en la posición 7, que es la de reposo
cero=[1 2 3 4 5 6 1 7];
uno=[1 6 5 11 12 7];
dos=[8 2 1 6 3 4 5 11 12 7];
```

```
tres=[8 2 1 6 5 4 10 9 3 6 12 7];
cuatro=[8 2 3 6 12 7 1 6 5 11 12 7];
cinco=[1 2 3 6 5 4 10 9 8 7];
seis=[1 2 3 4 5 6 3 9 12 7];
siete=[8 2 1 6 5 11 12 7];
ocho=[1 2 3 4 5 6 1 7 12 6 3 9 12 7];
nueve=[12 11 5 6 1 2 3 6 12 7];

%Colocamos el robot en la posición 7, que será la de reposo
qreposo=irb120_ikine(T_Vertices(:,:,7),[0 0 0 0 0 0]);
irb120.plot(qreposo);
```

En la figura siguiente, podemos ver el resultado de ejecutar el script denominado *panel_números*, en el que podemos encontrar el código descrito anteriormente.

Para ejecutar este script debe ejecutarse antes mdl_irb120 (archivo creado en el punto anterior 4.2 “Modelado del robot irb120 en Matlab”), ya que utiliza la variable TRi (rotación intrínseca del sistema de referencia del efector final del robot respecto al sistema de referencia de la base (mundo).

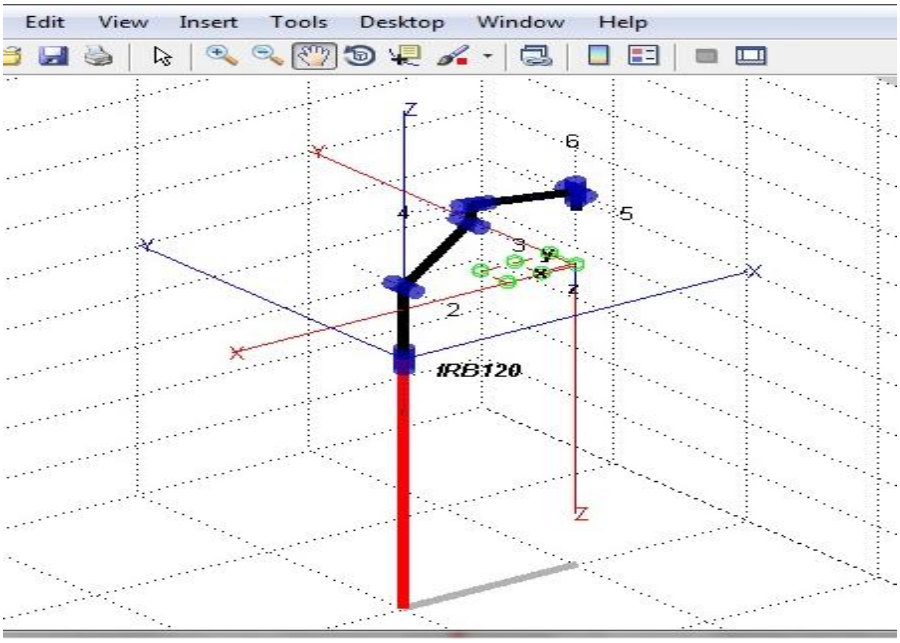


FIGURA 4.6 Representación del robot y del panel numérico

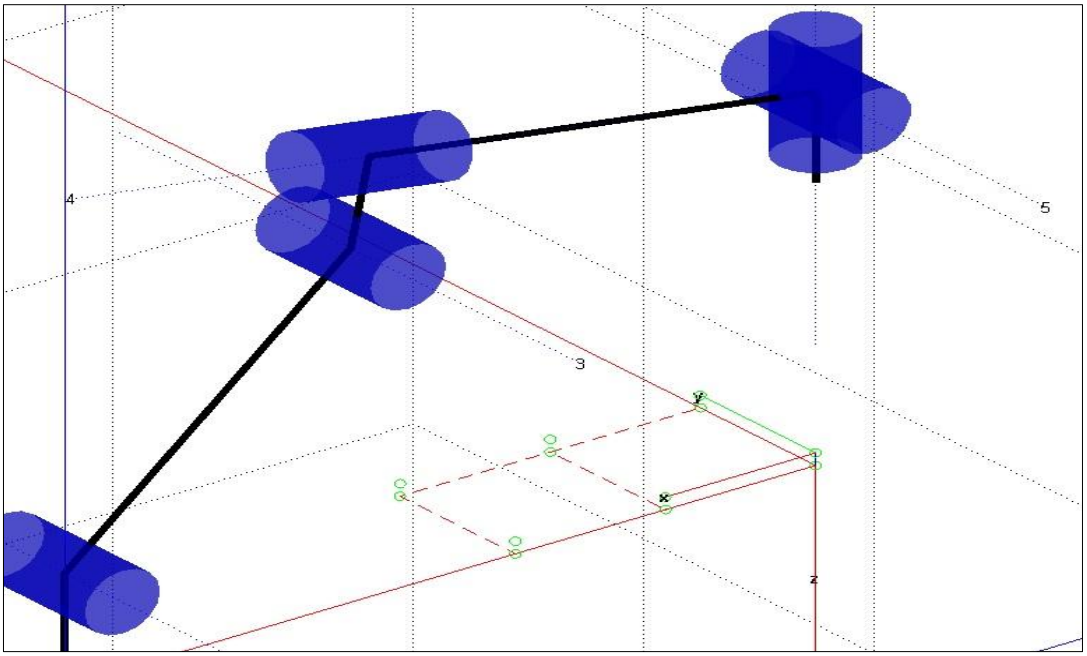


FIGURA 4.7 Representación del robot y del panel numérico + ZOOM

4.4 Trazado de números desde Matlab utilizando trayectorias cartesianas

Para el desarrollo de nuestra aplicación final de trazado de números, podríamos plantearnos dos posibles soluciones: realizarlo mediante trayectorias articulares o mediante trayectorias cartesianas.

La solución mediante trayectorias articulares implica tener que enviarle al robot por el socket, las configuraciones articulares de cada vértice y la configuración articular de los puntos situados entre los vértices. Debido a que no imponemos el tipo de trayectoria a realizar, para conseguir que el robot realice una trayectoria lo más recta posible deberíamos enviarle un número muy elevado de configuraciones articulares en un corto periodo de tiempo, por lo que esta solución no es viable para nuestra aplicación debido a la limitación temporal de la comunicación por el socket, que es aproximadamente tres segundos.

Por otro lado disponemos de la solución mediante trayectorias cartesianas. Lo que le enviamos al robot por el socket en este caso, son las posiciones cartesianas del efector final correspondientes a cada vértice. Al imponer el tipo de trayectoria conseguimos que el robot vaya de vértice en vértice siguiendo una línea recta y precisa.

Por este motivo, elegimos la solución mediante trayectorias cartesianas para desarrollar nuestra aplicación.

4.4.1 Resultados obtenidos utilizando el modelo del robot en Matlab

Para realizar la solución de este apartado mediante trayectorias cartesianas, necesitamos hacer uso de tres archivos, dos de ellos mencionados anteriormente en el punto "Creación del panel numérico", como son el archivo *mdl_irb120* y el archivo *panel_numeros*. Nos centrarnos por tanto en el análisis del tercer archivo denominado *aplicación_ikine.*

En este archivo, vamos a realizar el trazado de números a través del recorrido de un vértice a otro. Iremos calculando el vértice siguiente y a su vez calcularemos 10 puntos intermedios entre cada vértice, para lograr que la trayectoria entre vértices sea lo más recta y lo más precisa posible.

Esto lo conseguimos gracias al código que podemos ver a continuación:

```
%Vamos recorriendo de vértice en vértice

n=length(num);

q0=qreposo;

%Primero vamos desde la posición de reposo hasta el primer vértice a pintar

T1=T_Vertices(:,:,7);

T2=T_Vertices(:,:,num(1));

Ts=ctraj(T1,T2,10);       % aquí seleccionamos 10 puntos en cada tramo de trayectoria

qs=irb120_ikine(Ts,q0);
```

```
q0=qs(end,:);

irb120.plot(qs);

for i=1:(n-1)

   T1=T_Vertices(:,:,num(i));

   T2=T_Vertices(:,:,num(i+1));

   Ts=ctraj(T1,T2,10);          % De nuevo 10 puntos en cada tramo de trayectoria

   qs=irb120_ikine(Ts,q0);

   q0=qs(end,:);

   irb120.plot(qs);

end
```

En las imágenes siguientes, observamos el funcionamiento de la aplicación en Matlab.

Veremos al robot pintar el número ocho. Es complicado ver el funcionamiento de la aplicación a través de imágenes, pero en archivos de video adjuntos a este proyecto, se podrá comprobar con mayor facilidad su correcto funcionamiento.

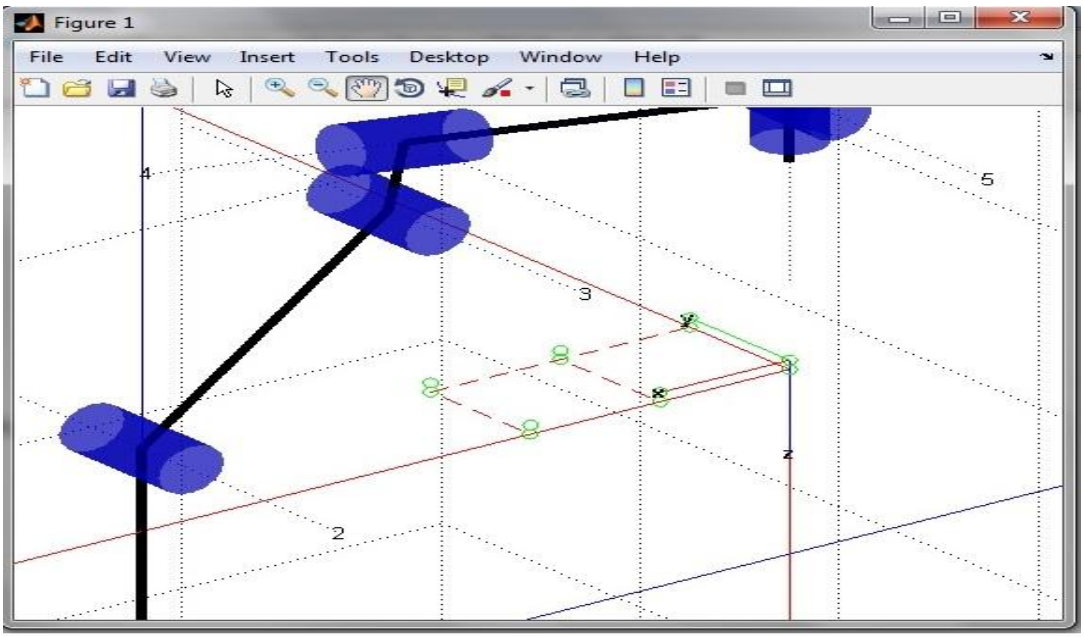


FIGURA 4.8 Robot situado en la posición de reposo

Primero, el robot se sitúa en la posición de reposo a la espera de recibir del usuario el número que desea pintar.

Cuando se da esta orden, el robot a través del código que hemos analizado antes, calcula el próximo vértice al que tiene que ir (calculando a su vez 10 puntos intermedios para hacer una trayectoria mucho más precisa).

Tras estos cálculos el robot se desplaza hasta el vértice siguiente y así sucesivamente. Observaremos que el robot se separa del panel (como ya comentamos en el punto 4.3 "Creación del panel numérico") para evitar pintar sobre trazo ya realizados.

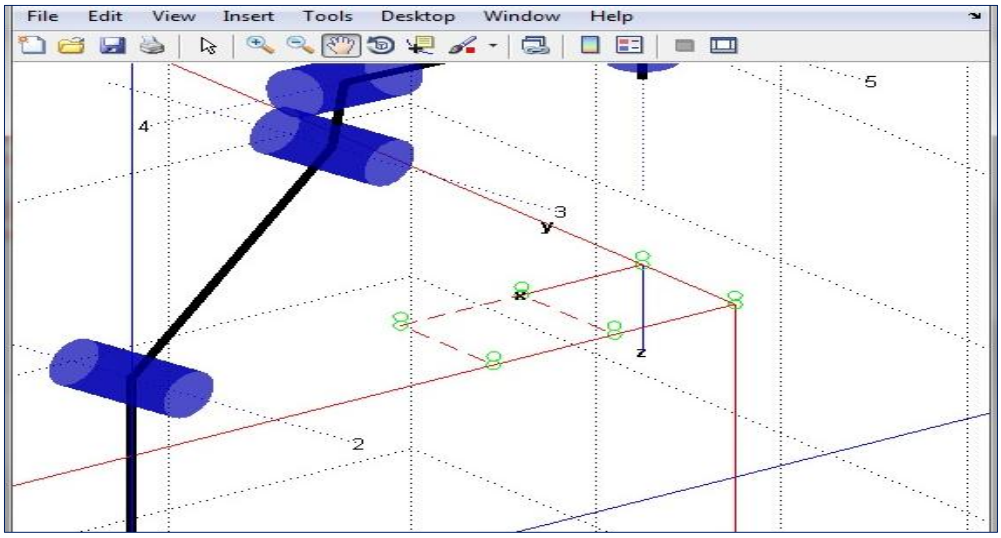


FIGURA 4.9 Robot llegando al segundo vértice del número a pintar

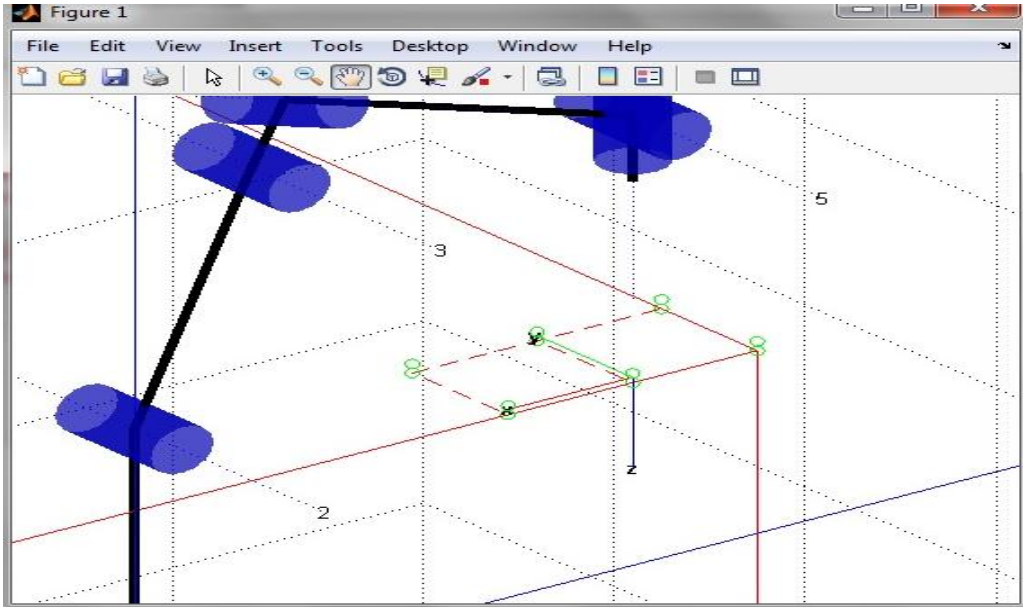


FIGURA 4.10 Robot llegando al sexto vértice del número a pintar

4.4.2 Resultados obtenidos con el robot utilizando la clase irb120

Estamos viendo simplemente la solución de la aplicación en Matlab, por tanto no es necesario el uso del socket ni de las interfaces creadas anteriormente.

Por tanto para demostrar el funcionamiento del socket y de las interfaces procedemos a comprobar el programa creado utilizando el simulador de RobotStudio en primer lugar y luego cargándolo sobre el controlador IRC5 para ver los resultados en el robot real.

La versión que se presenta a continuación posee una serie de variantes respecto a la que acabamos de analizar en Matlab.

Por ejemplo, necesitamos implementar la parte de conexión con el servidor a través de las funciones TCP/IP como ya hemos estudiado. También cambiará la forma de enviar los datos al robot.

Haremos uso de la clase "*irb120"* que creamos en apartados anteriores, para enviar al robot cada uno de los puntos y posiciones que se han ido calculando con el modelo del robot que habíamos desarrollado.

A continuación, veremos el código que varía respecto a la versión que acabamos de mencionar:

```
% Conectamos con el robot a través de la clase irb120

robot.connect;

%Primero vamos de la posición de reposo hasta el primer vértice

T2=T_Vertices(:,:,num(1));

pos=transl(T2);

% pasamos las posiciones a milímetros

pos=pos*1000;

rot=tr2rpy(T2);
```

```
% cambio de ángulos a grados
rot=rot*180/pi;
robot.TCProtandpos([rot(3) rot(2) rot(1) pos(1) pos(2) pos(3)]);
pause(2);
% Vamos recorriento de vértice en vértice
n=length(num);
q0=qreposo;
for i=1:(n-1)
   T2=T_Vertices(:,:,num(i+1));
   pos=transl(T2);
% pasamos las posiciones a milímetros
   pos=pos*1000;
   rot=tr2rpy(T2);
% cambio de ángulos a grados
   rot=rot*180/pi ;
   robot.TCProtandpos([rot(3) rot(2) rot(1) pos(1) pos(2) pos(3)]);
   pause(2);
end
```

Con este código, se consigue que nuestro brazo Robot en su software de simulación RobotStudio, realice el trazado de número de forma correcta y precisa.

Lo que podemos comprobar en las siguientes figuras, donde el robot trazará el número ocho.

RESULTADOS EN ROBOTSTUDIO

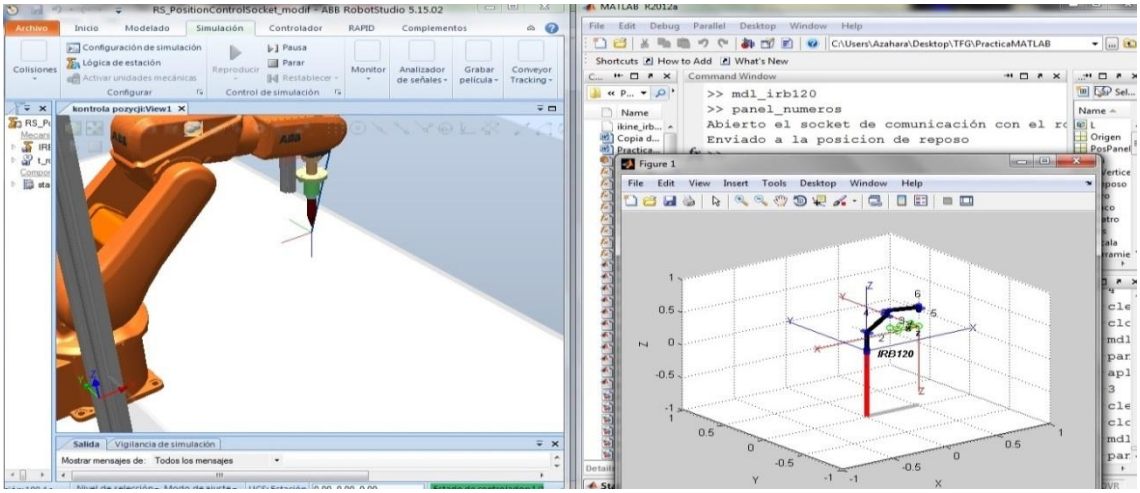


FIGURA 4.11 Robot situado en posición de reposo, esperando recibir del usuario el número a pintar

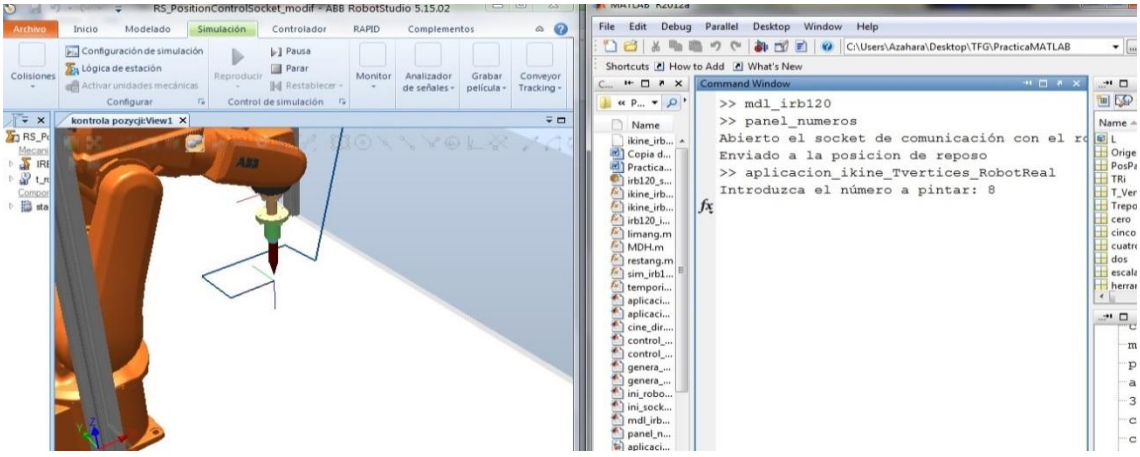


FIGURA 4.12 Robot realizando el trazado del número solicitado

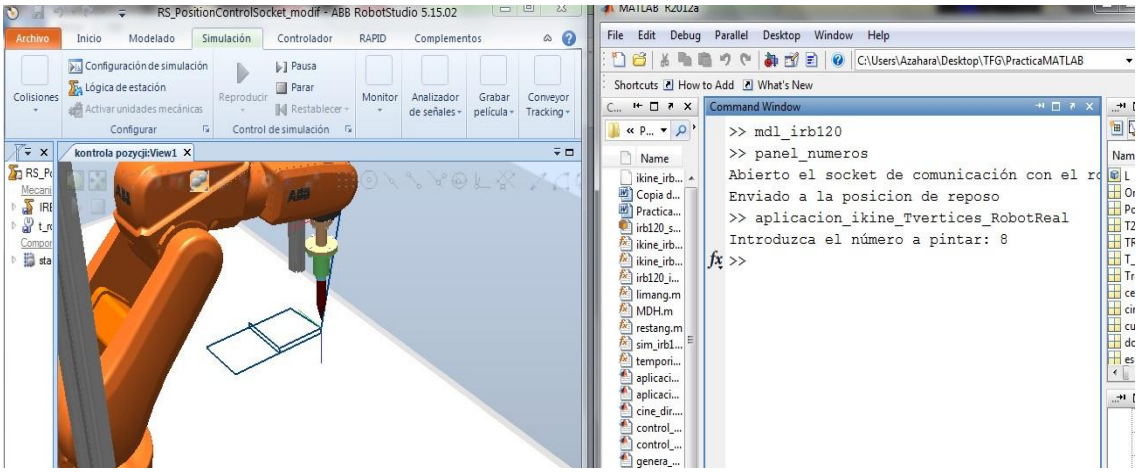


FIGURA 4.13 Robot en posición de reposo, tras finalizar el trazado del número

RESULTADOS EN EL ROBOT IRB120

A continuación podremos ver los resultados de la aplicación volcada sobre el robot IRB120. Para el documento, se han realizado una serie de fotografías, pero se podrá comprobar mejor su funcionamiento visualizando los videos que aparecen en el CD adjunto a este proyecto.

En las siguientes figuras observaremos en varios pasos cómo el robot va de la posición de reposo a los diferentes vértices, realizando el trazado del número cinco, en este caso.

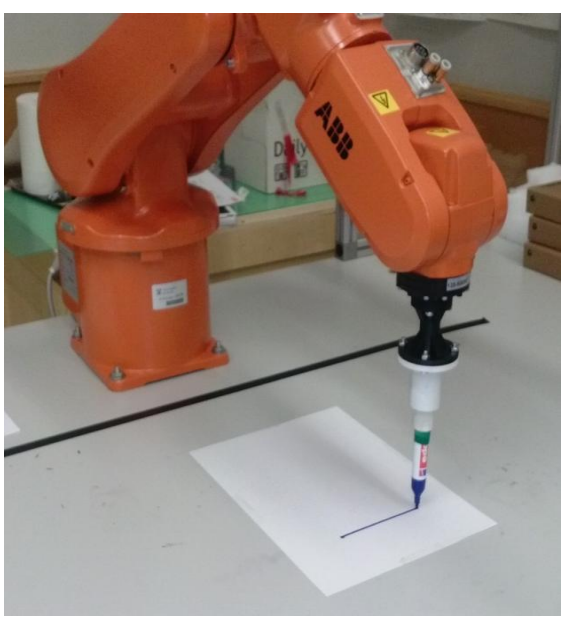


FIGURA 4.14 y 4.15 Robot Real dibujando el número cinco

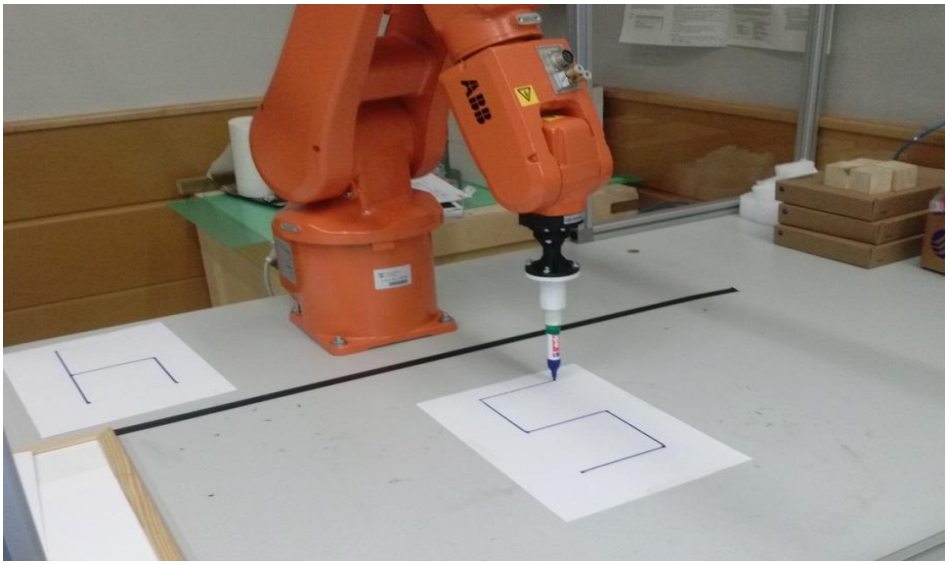


FIGURA 4.16 Robot Real dibujando el número cinco

En las siguientes imágenes podemos observar los números del cero al nueve que ha trazado el robot:

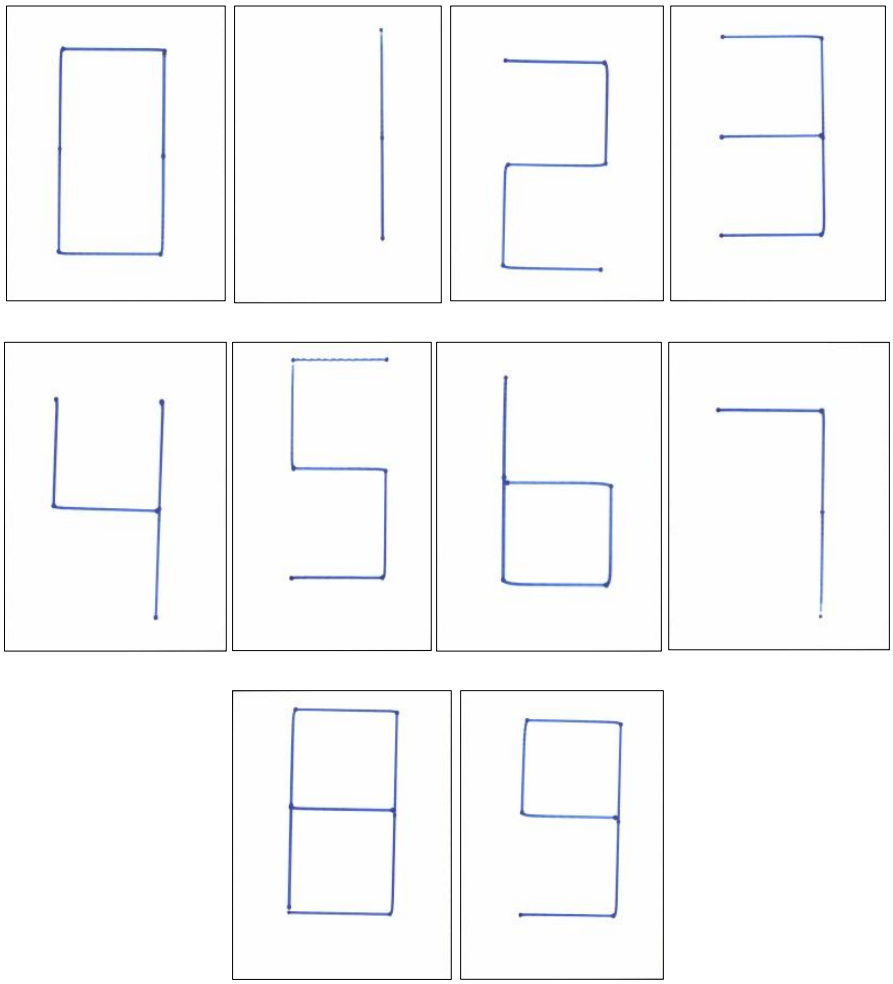


FIGURA 4.17 Números trazados por el robot IRB120

4.5 Trazado de números mediante control diferencial

Ahora abordaremos la segunda manera de realizar el trazado de números. En vez de utilizar la clase irb120, utilizaremos el bloque de Simulink que creamos, denominado "irb120".

En este apartado, haremos uso tanto de la herramienta de Simulink, como de archivo .m y los comandos introducidos desde la ventana de comandos de Matlab.

4.5.1 Resultados obtenidos en Matlab utilizando el bloque de Simulink "irb120"

Antes de entrar en la implementación de esta solución para nuestra aplicación final, debemos comentar ciertos conceptos necesarios, para la comprensión de lo que veremos a continuación.

El control diferencial, hace uso de la matriz Jacobiana inversa, para calcular las velocidades articulares necesarias para conseguir una determinada velocidad (o trayectoria) cartesiana (del efector final), según la conocida relación:

$$q = J(q)^{-1} \cdot \vartheta$$

La gran ventaja de este método de control es que no requiere aplicar la cinemática inversa del robot y que puede ser aplicado a robots, tanto redundantes (más grados de libertad que dimensiones en el espacio de la tarea), como infractuados (menos grados de libertad que dimensiones del espacio de la tarea).

El esquema de control diferencial responde a las siguientes ecuaciones discretas:

$$q\ (k) = J(q(k))^{-1} \cdot \vartheta$$

$$q\ \ k+1\ = q\ \ k\ \ + T \cdot q\ (k)$$

Donde la segunda ecuación es un integrador que permite calcular las variables articulares del siguiente periodo de muestreo a partir de las actuales y del vector de velocidad deseada para el efector final v .

El algoritmo puede implementarse tanto en lazo abierto como en lazo cerrado.

Nosotros hemos implementado el algoritmo en lazo cerrado. En ese archivo de Simulink que a continuación veremos, también hemos incluido nuestro bloque irb120, que se encargará de enviar al robot los puntos y posiciones que se vayan calculando a través de la ejecución del conjunto de bloques que forman nuestro modelo de control diferencial.

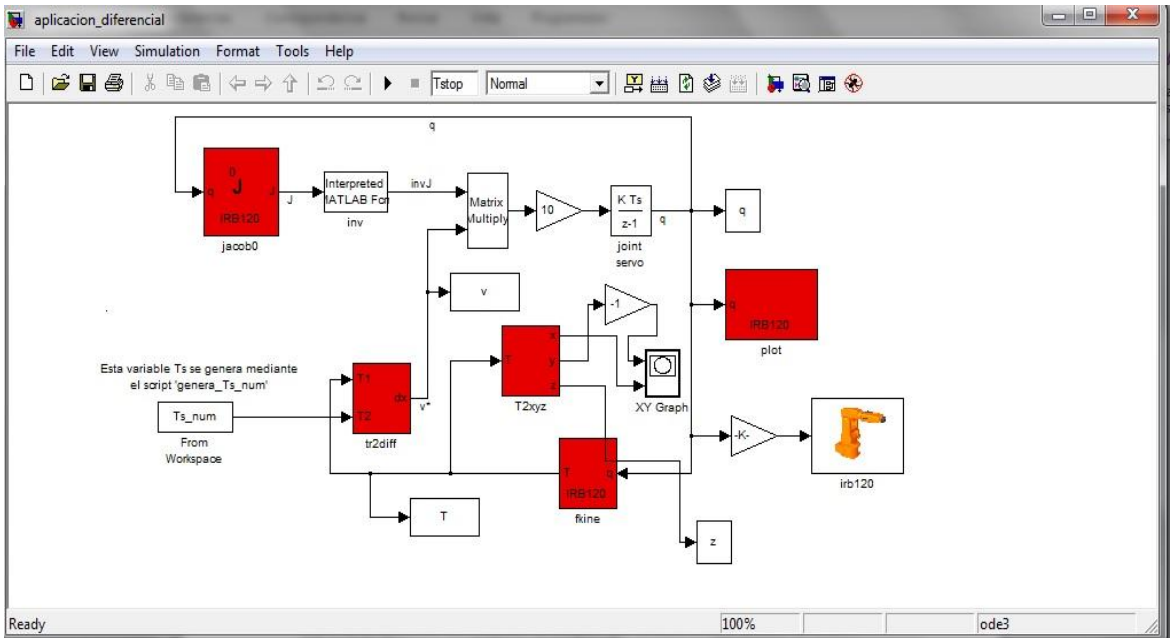


FIGURA 4.18 Modelo de Simulink del control diferencial con el bloque "irb120"

Al igual que en el punto anterior, la demostración del funcionamiento la veremos en Matlab y posteriormente con el Robot en simulación y con el Robot real.

Para implementar esta aplicación, necesitamos hacer uso de dos archivos más, respecto a las anteriores. El del modelo de Simulink que acabamos de ver en la imagen superior y un nuevo archivo denominado *genera_Ts_num.*

El código de este archivo .m lo podemos ver a continuación (es el código referente al archivo utilizado para realizar la simulación con RobotStudio, aunque los ejemplos de demostración que veamos más adelante sean sólo desde la plataforma Matlab):

```
%Abrimos la conexión para realiza la comunicación con el robot

global t

t=tcpip('127.0.0.1',1024);

fopen(t);

numero=input('Introduzca el número a pintar: ');

switch (numero)

    case 0,
```

```
        num=cero;
    case 1,
        num=uno;
    case 2,
        num=dos;
    case 3,
        num=tres;
    case 4,
        num=cuatro;
    case 5,
        num=cinco;
[…]
        num=ocho;
    case 9,
        num=nueve;
    otherwise,
      error('Entrada no válida');
end
%Inicializamos los campos de la estructura, vacíos
Ts_num.time=[];
Ts_num.signals.values=[];
Ts_num.signals.dimensions=[4 4];
```

```
% Vamos recorriento de vértice en vértice

n=length(num);

q0=qreposo;

for i=1:(n-1)

   T1=T_Vertices(:,:,num(i));

   T2=T_Vertices(:,:,num(i+1));

%10 puntos en cada tramo de trayectoria

   Ts=ctraj(T1,T2,20);
Ts_num.signals.values(:,:,[length(Ts_num.signals.values)+1:length(Ts_num.signals.val
ues)+length(Ts)])=Ts;

end

%Generamos el vector de tiempos, cada 100 ms

paso=2;   %paso de tiempo

Ts_num.time=[0:paso:(length(Ts_num.signals.values)-1)*paso];

%Finalización de la simulación

Tstop=Ts_num.time(end);
```

El modo de ejecutar esta aplicación, es similar a las demás, con la salvedad de la ejecución del archivo de Simulink.

Contamos entonces, con los archivos *mdl_irb120*, *panel_numeros*, (para ver cómo se trazan los números), *genera Ts_num* y el archivo .mdl *control diferencial*.

Para una mejor visualización de los resultados, hemos creado una figura en Matlab a modo de panel, para poder observar mejor el trazado de los números.

Vamos a ver en las figuras siguientes, el trazado del número 8 (el más complejo), a través del bloque de Simulink irb120 junto con el control diferencial.

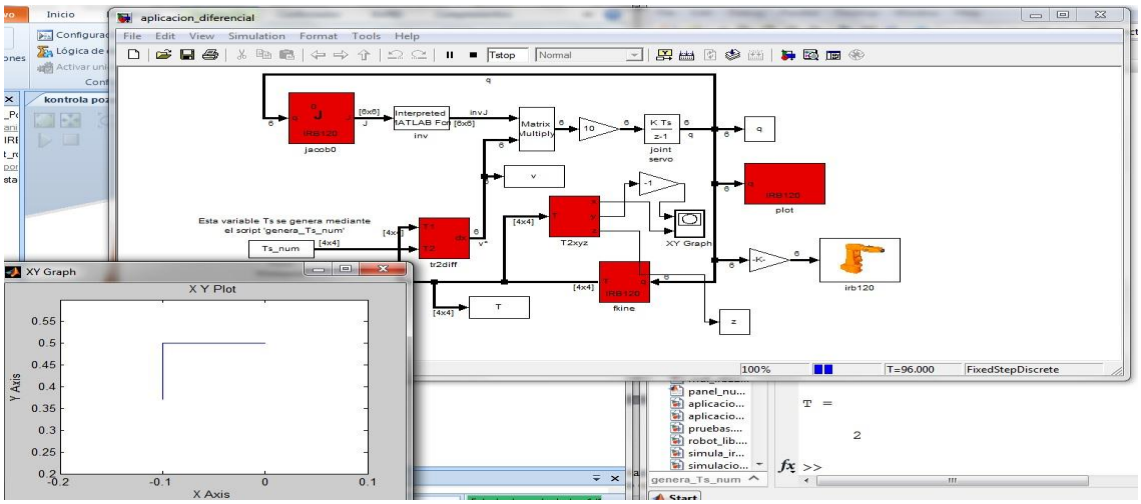


FIGURA 4.19 Trazado del número 8 mediante el control diferencial con el bloque "irb120" [1]

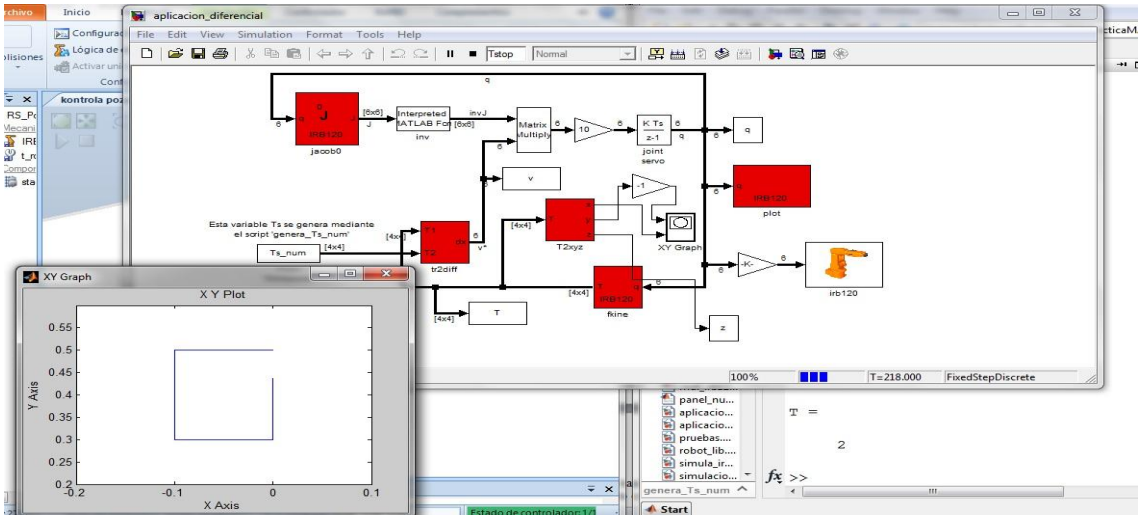


FIGURA 4.20 Trazado del número 8 mediante el control diferencial con el bloque "irb120" [2]

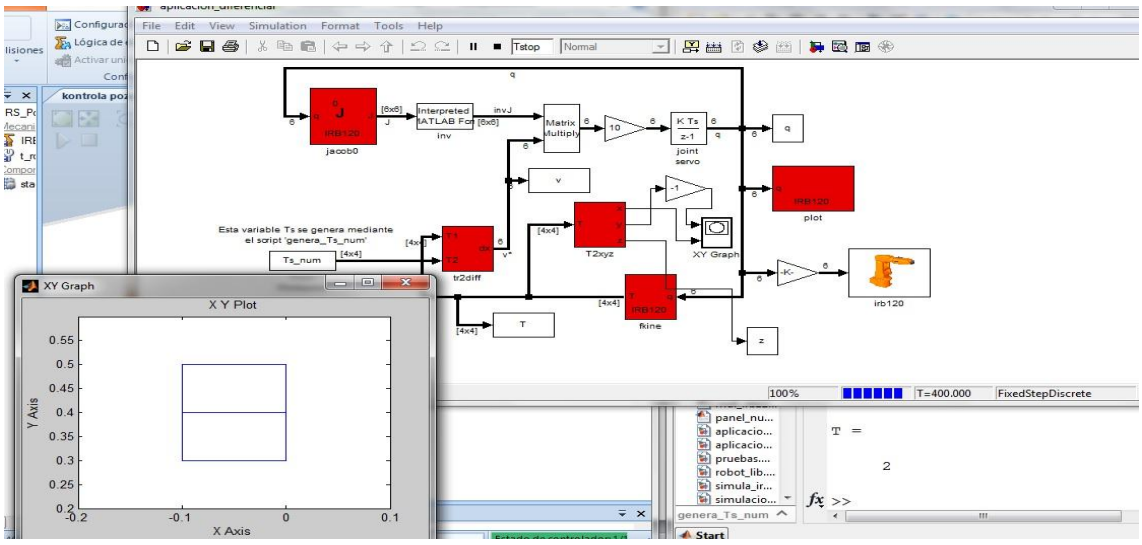


FIGURA 4.21 Trazado del número 8 mediante el control diferencial con el bloque "irb120" [3]

Nota: *En la simulación a través de Matlab no surge ningún problema a simple vista al realizar el trazado de los números, pero al mandarle al bloque irb120 los resultados y al procesarlos el robot, el pintado que realiza no es tan preciso como el que podemos ver en las imágenes anteriores.*

En el siguiente apartado mostraremos la solución mediante el uso del bloque "irb120" de Simulink y analizaremos el porqué del problema surgido.

4.5.2 Resultados obtenidos con el robot, utilizando el bloque de Simulink "irb120"

El código y los archivos utilizados en este apartado, son los mencionados anteriormente. La única diferencia que presenta respecto a la solución en Matlab, es la introducción del Simulador RobotStudio y la forma de ejecutar el programa.

En las siguientes imágenes observaremos el funcionamiento de la aplicación en el simulador del robot, a su vez se visualiza la ventana (plot) realizada en Matlab para comprobar que el trazado es simultáneo, salvando los errores que trataremos más adelante.

RESULTADOS EN ROBOTSTUDIO

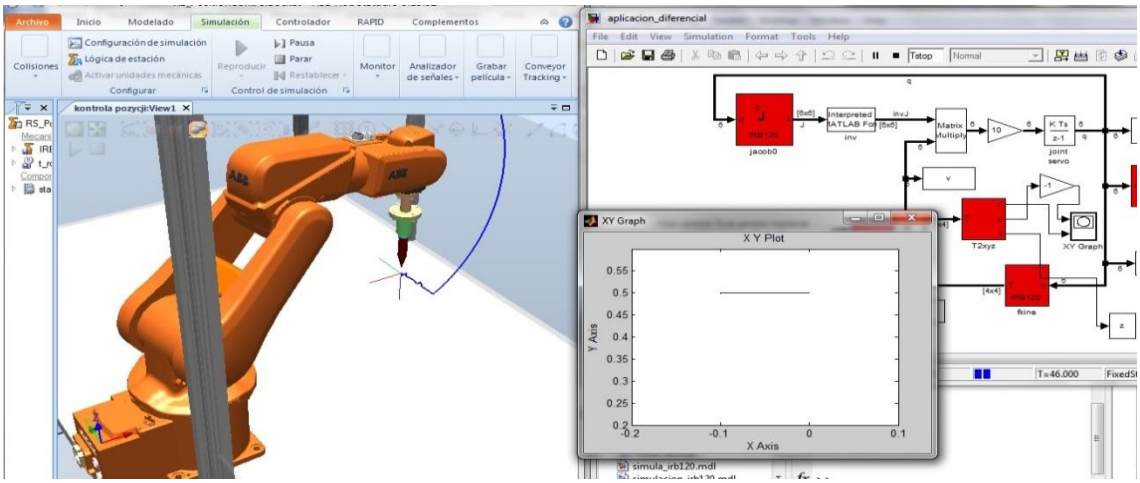


FIGURA 4.22 Trazado del número 0 en el simulador, mediante el control diferencial con el bloque "irb120" [1]

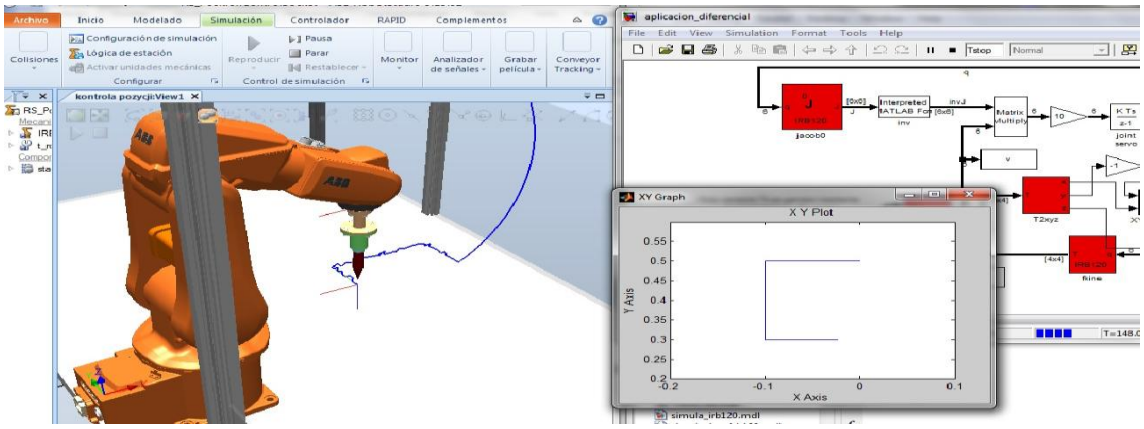


FIGURA 4.23 Trazado del número 0 en el simulador, mediante el control diferencial con el bloque "irb120" [2]

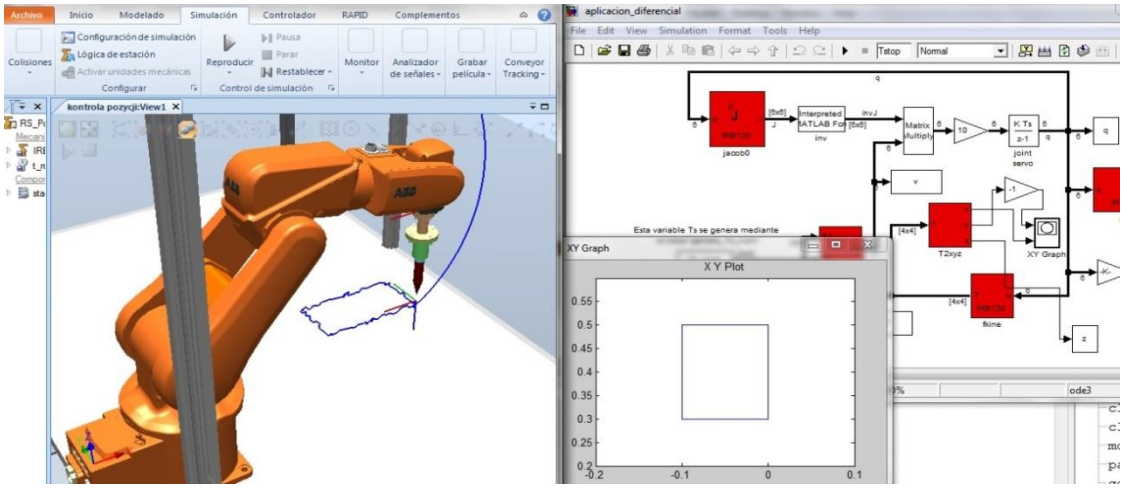


FIGURA 4.24 Trazado del número 0 en el simulador, mediante el control diferencial con el bloque "irb120" [3]

El problema mencionado anteriormente, deriva de los bloques que calculan la cinemática, es decir; el control diferencial conlleva el problema de que no exigimos una trayectoria recta. Si nos fijamos en las zetas, no son cero (suponiendo que colocamos el panel en un plano con Z=0) varían, por tanto las q's calculadas son enviadas al robot y el resultado es un trazado correcto del número, ya que se entiende que número es, pero no es preciso ni posee una trayectoria lineal.

Debido a la limitación de tiempo que presenta el socket (tres segundos aprox.) no podemos realizar la prueba en el robot real, ya que el robot recibe demasiadas ordenes en un corto periodo de tiempo y no le da tiempo a ir realizando cada una, por lo que el trazado no se realiza correctamente.

5 Conclusiones y trabajos futuros

En este proyecto se han desarrollado varias interfaces de comunicación que permiten controlar el brazo robot IRB120 de ABB desde Matlab, utilizando para ello un socket TCP/IP. La comunicación puede realizarse bien desde un interfaz gráfico, desde una clase de Matlab, o desde un bloque de Simulink. El disponer de varias opciones amplía el abanico de aplicaciones que se pueden implementar en Matlab para controlar el robot.

La principal limitación que se ha detectado en la comunicación es el tiempo que tarda el socket en enviar un datagrama al robot, que está entorno a los 3 segundos, y que condiciona en cierta medida el tipo de aplicaciones que se podrán desarrollar desde Matlab. En definitiva, serán apropiadas aquellas aplicaciones que desarrollen el control a nivel del espacio cartesiano, es decir, enviando puntos de destino de la herramienta, pero las aplicaciones que requieran un control articular más preciso no podrán ejecutarse de forma correcta. Esto no supone una limitación en la mayor parte de los casos, ya que los algoritmos de control de bajo nivel ya están implementados en el controlador del robot.

Una vez realizados los interfaces de comunicación, es posible plantear como trabajos futuros multitud de aplicaciones para el control del brazo robot desde Matlab, como pueden ser la evitación de obstáculos en la trayectoria detectados por una cámara, la reproducción del movimiento de un brazo humano detectado por una Kinect, la detección y apilamiento de determinados objetos dentro de la estación de trabajo del robot, etc.

6 Manual de usuario

En este capítulo se detallará un manual para el usuario, con el cual se podrá acceder a cualquier funcionalidad de la aplicación. Se explicarán detalladamente los paneles de cada una de las aplicaciones desarrolladas y su función dentro del programa. También se hará una guía de botones con los que se ayudará al usuario a entender mejor el funcionamiento del sistema.

6.1 Uso de la Interfaz de comunicación a través de GUI

La interfaz GUI está desarrollada para que al ser ejecutada la aplicación o simplemente escribiendo en la ventana de comandos de Matlab el nombre del archivo, nos aparezca la ventana de la Interfaz lista para ser usada.

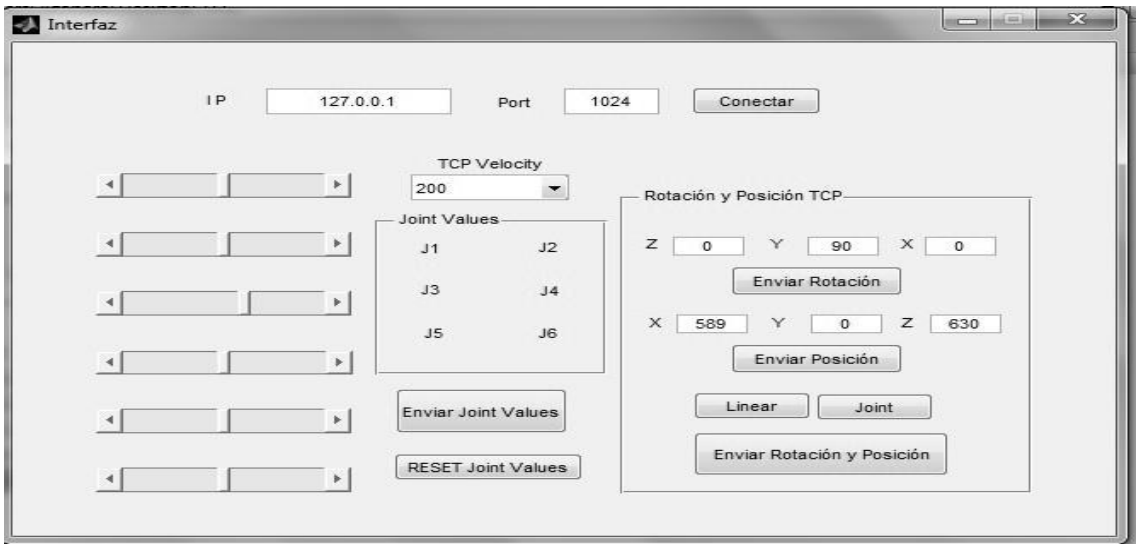


FIGURA 5.1 Vista de la Interfaz al ser ejecutada

En la figura anterior podemos ver lo que ocurre si desde la ventana de comandos de Matlab introducimos lo siguiente

```
>> Interfaz
```

Con la ventana que parece al ejecutar esta línea de código, es con la que trabajaremos para dar las instrucciones necesarias para que el robot realice los correspondientes movimientos.

El flujo de trabajo se inicia desde Matlab abriendo el archivo .fig haciendo clic con el botón derecho y seleccionando *"Open in GUIDE"*.

Después debemos tener ejecutada nuestra aplicación de RobotStudio para que el servidor se quede a la escucha. Esto lo realizamos pulsando el botón de *reproducir* en la pestaña de Simulación o bien utilizando el controlador IRC5 y el servidor para el Robot real.

A continuación ejecutaremos nuestra interfaz GUIDE de la manera que hemos indicado anteriormente y seguiremos los siguientes pasos para utilizar la interfaz gráfica:

1. Introducimos la dirección IP y el número de puerto en su cuadro de texto correspondiente. Hacemos clic en Conectar.

2. Para enviar un valor de Joint al robot, debemos seleccionar el valor en el slider, comprobando el número de grados elegidos en los valores de la tabla de la derecha de los sliders. Tras elegir los valores de los ángulos hacemos clic en el botón Enviar Joint values.

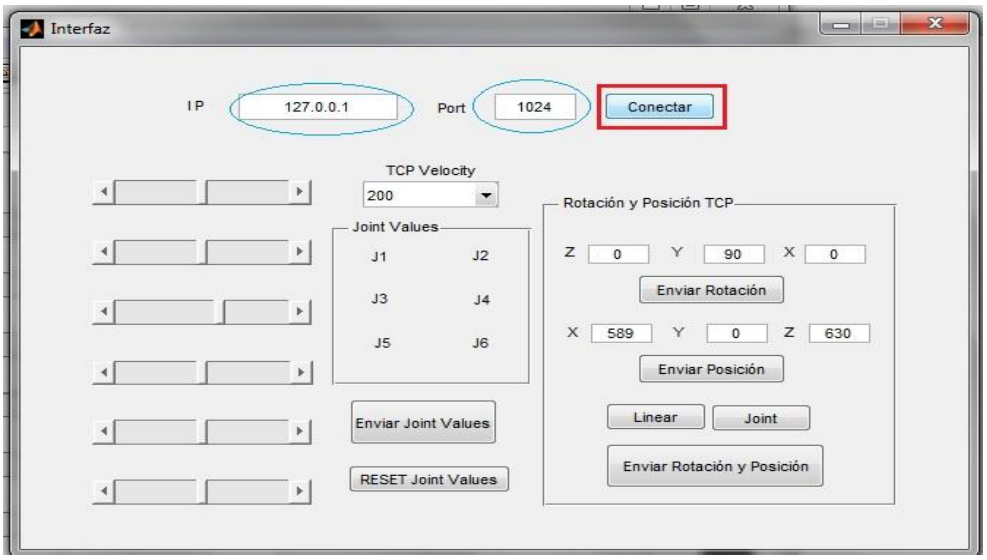


FIGURA 5.2 Paso Conectar con el servidor

3. Si queremos devolver al robot a la posición de reposo, haremos clic en el botón RESET Joint Values

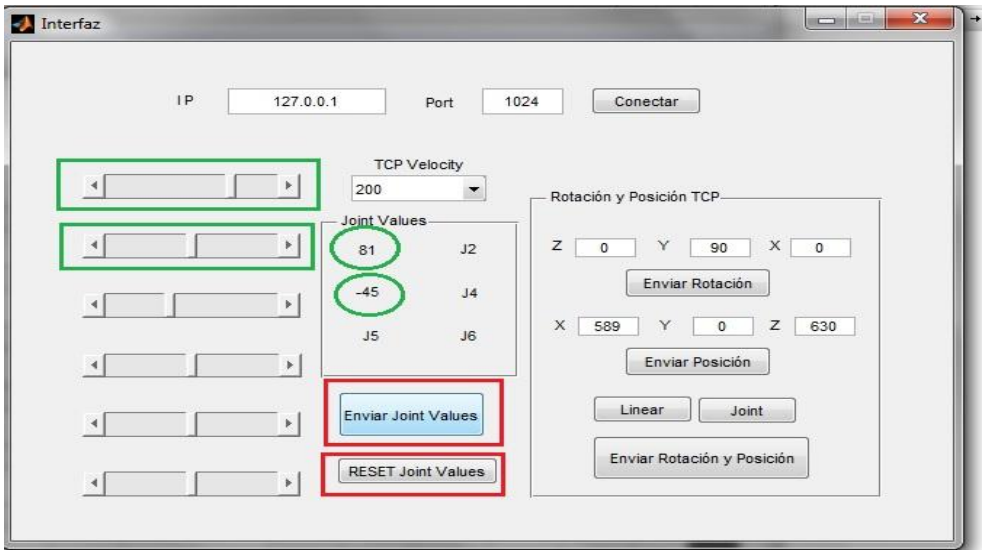


FIGURA 5.3 Enviar valores de Joint y resetear

4. Por último, podemos enviarle al robot la rotación del TCP, la posición o ambas al mismo tiempo. Esto se realiza escribiendo en el cuadro de texto correspondiente los valores deseados y haciendo clic en los botones: Enviar rotación, Enviar posición y Enviar Rotación y Posición, respectivamente.

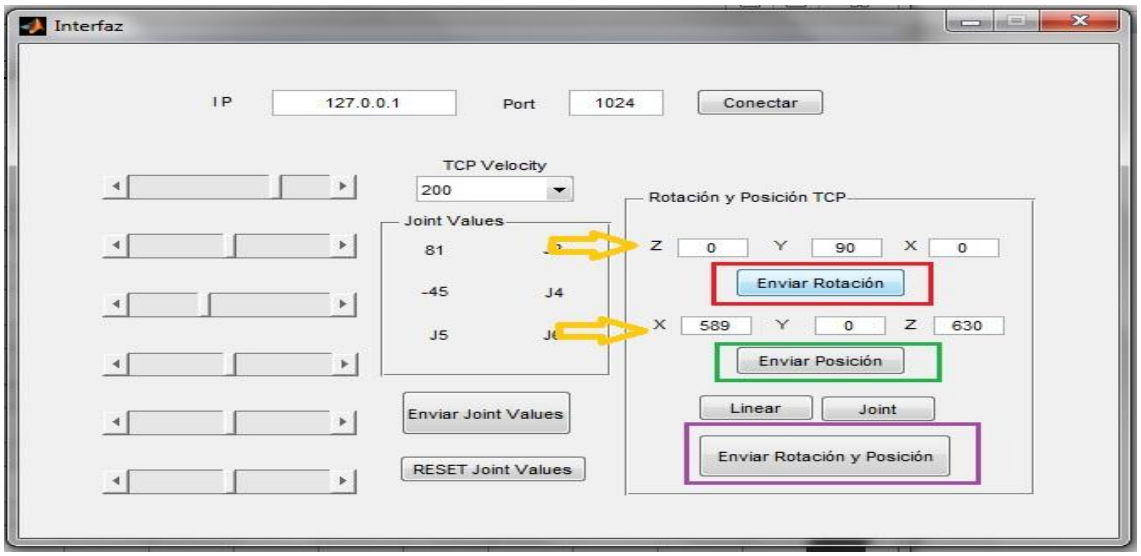


FIGURA 5.4 Enviar Rotación y Posición del TCP

Los resultados de ir ejecutando los pasos mencionados, se podrá comprobar en la ventana de RobotStudio. El Robot irá realizando los movimientos que le indiquemos a través de la interfaz.

6.2 Uso de la Interfaz de comunicación a través de clases

El uso de esta interfaz, no requiere mayores indicaciones que las que hemos dado en apartados anteriores.

Para comenzar debemos crear la clase y conectarnos con el robot.

Depués ejecutaremos el código correspondiente a cada una de las funciones asociadas a las órdenes, que debe recibir el robot para realizar cada uno de los movimientos que se le indiquen.

En las siguientes figuras, observamos los pasos que se deben seguir para trabajar de forma correcta con esta interfaz:

```
>> robot=irb120('127.0.0.1',1024)
```

```
>> robot.connect
```

FIGURA 5.5 Creación de la clase y conexión con el robot

```
>> robot.jointvalues([20 -40 0 0 0 0])

>> robot.TCProtation([45 0 90])
>> robot.TCPposition([400 0 630])

>> robot.TCProtandpos([45 0 90 400 0 630])
```

FIGURA 5.6 Uso de las respectivas funciones de la clase

6.3 Uso de la Interfaz de comunicación a través del bloque de Simulink "irb120"

El flujo de esta interfaz comienza desde Matlab. Debemos ejecutar el archivo .m denominado *ini.m*

Con la ejecución de este archivo conseguimos realizar la conexión con el servidor de RobotStudio (que cómo ya sabemos, deberá estar ejecutándose).

1. Ejecutamos el archivo .m de inicialización.

2. Abrimos el archivo .mdl en el que tenemos nuestro bloque de Simulink y le damos al botón de reproducir.

3. Para enviar unos valores de Joint al Robot hacemos doble clic sobre el vector de constantes y modificamos los valores que deseamos enviar al robot.

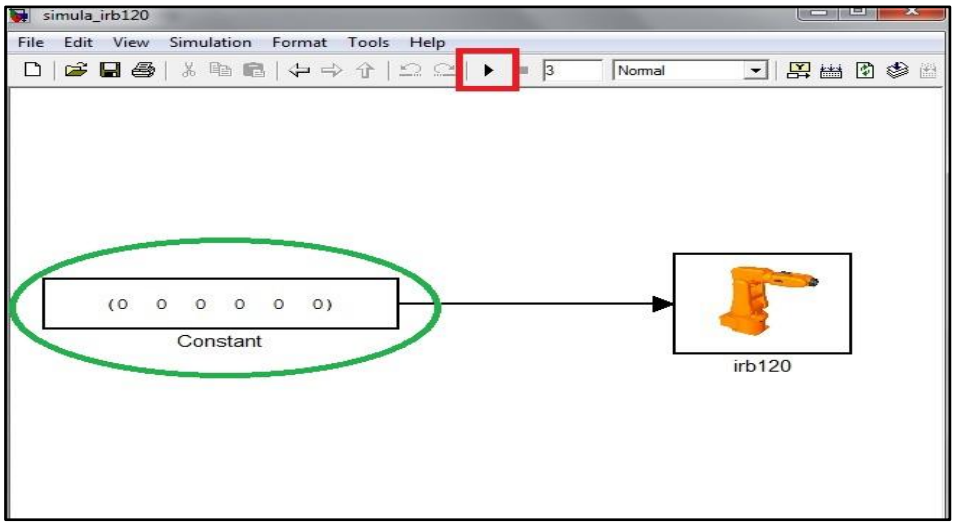


FIGURA 5.7 Ejecución del archivo .mdl

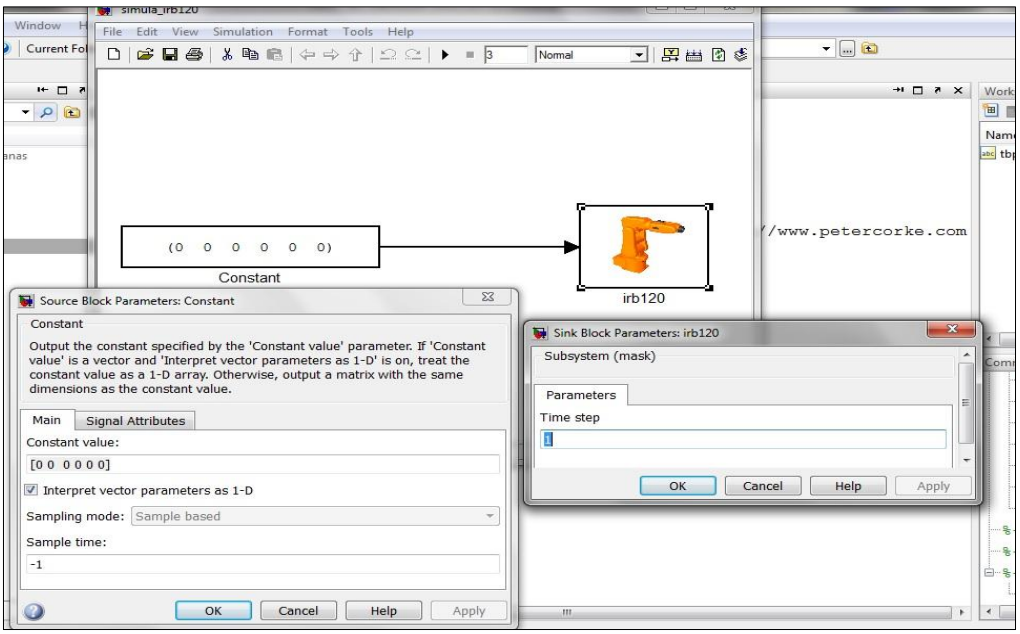


FIGURA 5.8 Ventanas para modificar los parámetros necesarios

7 Pliego de condiciones

En este capítulo se detallan las características de las herramientas utilizadas.

7.1 Hardware

1. **Portátil Sony VAIO**

- Procesador: Intel® Core™ i5
- RAM: 8 GB
- Tarjeta gráfica : NVIDIA GeForce 720M

2. **Robot Industrial ABB-IRB120**

- Peso: 25kg
- Altura: 580 mm
- Carga soportada: 3kg (hasta 4 Kg con muñeca vertical)
- Controlador: IRC5 Compact

7.2 Software

- Windows 7 © Microsoft Corporation.
- ABB RobotStudio.
- Matlab R212a (Con Toolbox de Robótica)
- Microsoft Office Home and Student 2010

8 Planos

En este capítulo podemos encontrar el código de los archivos realizados tanto para las interfaces como para la aplicación final.

De los archivos que sean demasiado extensos, sólo pondremos en este documento las partes más importantes; el resto podrá consultarse en el CD adjunto a este proyecto.

Interfaz GUI (archivo .m)

```
function edit1_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.

global IP
global t
global Port

IP = get(hObject,'String');  % guardamos en la variable IP el valor del edit text que introduzca el usuario

guidata(hObject, handles);   %Salvar datos de la aplicación

function pushbutton1_Callback(hObject, eventdata, handles)
%Definición de variables globales utilizadas para este control.

global t
global IP
global Port

% Establecemos la conexión mediante el protocolo TCP/IP utilizando las funciones de la toolbox de Matlab correspondiente.

t=tcpip(IP, Port); fopen(t);

msgbox('La Conexión se ha establecido'); % Abrimos un cuadro de diálogo al haberse establecido de forma correcta la conexión

guidata(hObject, handles); %Salvar datos de la aplicación

function slider1_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.

global J1
global sj1

% cogemos el valor del slider y nos quedamos con su valor absoluto y su signo para asignárselo a las variables J y sj

      ang =get(hObject,'Value');
      signo = sign(ang);
      J1 = uint8(round(abs(ang)));


      if signo == -1
         sj1 = 0;
      else
         sj1=1;
```

```
        end
       angulo =round(ang);

% actualizamos el valor del static text     correspondiente a este Joint Value

             set(handles.text2,'String',angulo);

    guidata(hObject, handles); %Salvar datos de la aplicación

function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Definición de variables globales utilizadas para este control.

global J1
global J2
global J3
global J4
global J5
global J6
global t
global sj1
global sj2
global sj3
global sj4
global sj5
global sj6


 % Construimos el datagrama

D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5 sj6 J6]);


%Enviamos el datagrama realizando una pausa de dos segundos, para dar tiempo al robot a realizar la instrucción ordenada.

fwrite(t, D1);
pause(2);

guidata(hObject, handles); %Salvar datos de la aplicación

function pushbutton3_Callback(hObject, eventdata, handles)%

%Definición de variables globales utilizadas para este control.

global J1
global J2
global J3
global J4
global J5
global J6
global sj1
global sj2
global sj3
global sj4
global sj5
global sj6
global t


% Enviamos el robot a la posición de Reposo

set(handles.slider1,'Value',0) ; set(handles.text2,'String',0);
```

```
set(handles.slider2,'Value',0);  set(handles.text3,'String',0);
[…]

%Y así sucesivamente con los demás valores Joint del robot.

sj1=1; J1=0; sj2=1; J2=0; sj3=1; J3=0; sj4=1; J4=0; sj5=1; J5=0; sj6=1;J6=0; % asignamos los varales  de reset y enviamos el datagrama

D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5 sj6 J6]);
fwrite(t, D1);
pause(2);

guidata(hObject, handles); % %Salvar datos de la aplicación

function edit3_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.

global sz
global Rz


% Obtenemos de los cuadros de texto las variables de Rotación para construir el datagrama

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

signo = sign (vble);

if signo == -1
   sz=0;
else
   sz=1;
end

Rz= uint8(round(abs(vble))); % Obtenemos la variable de rotación Rz.

guidata(hObject, handles); %Salvar datos de la aplicación

function edit6_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.

global spx
global X
global xr

%Obtenemos las variables en este caso correspondientes a la coordenada x para construir el datagrama

x=str2double(get(hObject,'String'));
xr=rem(x,100);
X=floor(x/100);
signo = sign (x);
if signo == -1
   spx=0;
else
   spx=1;
end

guidata(hObject, handles); %Salvar datos de la aplicación

function pushbutton4_Callback(hObject, eventdata, handles)

%Definición de variables globales utilizadas para este control.

global sz
global sx
global sy
```

```
global Rz
global Rx
global Ry
global t

D2=uint8([2 sz Rz sy Ry sx Rx]); % Construimos el datagrama


fwrite(t, D2); %Enviamos datagrama realizando una pausa de 2 segundos entre medias
pause(2);

guidata(hObject, handles); %Salvar datos de la aplicación

function popupmenu2_Callback(hObject, eventdata, handles)

%definimos las variables globales utilizadas en este control.

global t;
global Vt;
global Vtr;
global Vr;
global Vrr;
sxl=1; xl=0; xlr=0; szl=1; zl=0; zlr=0;


vel= get(hObject,'Value'); % Ajustamos valores de velocidad a partir del control del popup menu
Vtr=rem(vel,100);
Vt=floor(vel/100);
Vrr=rem(vel,100);
Vr=floor(vel/100);

D5=uint8([5 1 Vt Vtr Vr Vrr sxl xl xlr szl zl zlr]);

% construimos y enviamos en datagrama.

fwrite(t,D5);
pause(2);

guidata(hObject, handles); %Salvar datos de la aplicación
```

Interfaz a través de clases en Matlab

```
classdef irb120 < handle

   % La clase denominada irb120 constará de
propiedades( IP, port;
   % conexion) y métodos en los que definimos las
funciones que van a
   % relizar la conexión con el robot y el envio de los
datagramas.

   properties
      IP;
      port;
      conexion;
   end

   methods
      function r=irb120(ip,puerto)
         r.IP=ip;
         r.port=puerto;
      end
      function connect(r)
         r.conexion=tcpip(r.IP, r.port);
         fopen(r.conexion);
         D1=uint8([1 1 0 0 2 1 2 1 0 1 0 1 0]);
    %Enviamos el dato
         fwrite(r.conexion,D1);
      end

       function disconnect(r)
         fclose(r.conexion);
      end

      function jointvalues(r,q)
         J1=uint8(round(abs(q(1))));
         signo = sign(q(1));
           if signo == -1
                  sj1 = 0;
           else
                  sj1=1;
           end

J2=uint8(round(abs(q(2))));
```

```
        signo2 = sign(q(2));
         if signo2 == -1
                sj2 = 0;
         else
                sj2=1;
         end

        J3=uint8(round(abs(q(3))));
        signo3 = sign(q(3));
         if signo3 == -1
                sj3 = 0;
         else
                sj3=1;
         end

        J4=uint8(round(abs(q(4))));
        signo4 = sign(q(4));
         if signo4 == -1
                sj4 = 0;
         else
                sj4=1;
         end

        J5=uint8(round(abs(q(5))));
        signo5 = sign(q(5));
         if signo5 == -1
                sj5=0;
         else
                sj5=1;
         end

        J6=uint8(round(abs(q(6))));
        signo6 = sign(q(6));
         if signo6 == -1
                sj6 = 0;
         else
                sj6=1;
         end


        D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5
sj6 J6]);

        %Enviamos el dato
        fwrite(r.conexion,D1);
        pause(1);
       % fwrite(r.conexion,D1);
       % pause(1);

      end

      function TCProtation(r,zyx)
        Rz=uint8(round(abs(zyx(1))));
        signoz = sign (zyx(1));

           if signoz == -1
              sz=0;
           else
              sz=1;
           end
        Ry= uint8(round(abs(zyx(2))));
        signoy = sign (zyx(2));

           if signoy == -1
              sy=0;
           else
              sy=1;
           end
        Rx= uint8(round(abs(zyx(3))));
        signox = sign (zyx(3));

           if signox == -1
              sx=0;
           else
              sx=1;
           end
        D2=uint8([2 sz Rz sy Ry sx Rx]);

        fwrite(r.conexion, D2);
        pause(2);
        fwrite(r.conexion, D2);
        pause(2);

      end

      function TCPposition(r,xyz)
        xr=rem(xyz(1),100);
        X=floor(xyz(1)/100);
        signox = sign (xyz(1));
           if signox == -1
              spx=0;
           else
              spx=1;
           end
        yr=rem(xyz(2),100);
        Y=floor(xyz(2)/100);
        signoy = sign (xyz(2));
           if signoy == -1
              spy=0;
           else
              spy=1;
           end
        zr=rem(xyz(3),100);
        Z=floor(xyz(3)/100);
        signoz = sign (xyz(3));
           if signoz == -1
              spz=0;
           else
              spz=1;
           end
        D3=uint8([3 spx X xr spy Y yr spz Z zr 0]);
        fwrite(r.conexion,D3);
        pause(2);
        fwrite(r.conexion,D3);
        pause(2);
      end


      function TCProtandpos(r,tcp)
       Rz=uint8(round(abs(tcp(1))));
        signorz = sign (tcp(1));

           if signorz == -1
              sz=0;
           else
              sz=1;
           end
        Ry= uint8(round(abs(tcp(2))));

  signory = sign (tcp(2));

           if signory == -1
              sy=0;
           else
```

```
            sy=1;
        end
    Rx= uint8(round(abs(tcp(3))));
    signorx = sign (tcp(3));

        if signorx == -1
            sx=0;
        else
            sx=1;
        end
    xr=rem(tcp(4),100);
    X=floor(tcp(4)/100);
    signopx = sign (tcp(4));
        if signopx == -1
            spx=0;
        else
            spx=1;
        end
    yr=rem(tcp(5),100);

  Y=floor(tcp(5)/100);
```

```
  signopy = sign (tcp(5));
        if signopy == -1
            spy=0;
        else
            spy=1;
        end
    zr=rem(tcp(6),100);
    Z=floor(tcp(6)/100);
    signopz = sign (tcp(6));
        if signopz == -1
            spz=0;
        else
            spz=1;
        end
    D4=uint8([4 sz Rz sy Ry sx Rx spx X xr spy Y
yr spz Z zr]);
    fwrite(r.conexion,D4);
    pause(2);
    fwrite(r.conexion,D4);
    pause(2);
  end
end
end
```

Función del bloque de Simulink “irb120”

```
function sim_irb120(x)
%#codegen

u=x(1:6);
T=x(7);

global t
J1=uint8(round(abs(u(1))));
signo = sign(u(1));
 if signo == -1
        sj1 = 0;
 else
        sj1=1;
 end

J2=uint8(round(abs(u(2))));
signo2 = sign(u(2));
 if signo2 == -1
        sj2 = 0;
 else
        sj2=1;
 end

J3=uint8(round(abs(u(3))));
signo3 = sign(u(3));
 if signo3 == -1
        sj3 = 0;
 else
        sj3=1;
 end

J4=uint8(round(abs(u(4))));
```

```
signo4 = sign(u(4));
 if signo4 == -1
        sj4 = 0;
 else
        sj4=1;
 end

J5=uint8(round(abs(u(5))));
signo5 = sign(u(5));
 if signo5 == -1
        sj5 = 0;
 else
        sj5=1;
 end

J6=uint8(round(abs(u(6))));
signo6 = sign(u(6));
 if signo6 == -1
        sj6 = 0;
 else
        sj6=1;
 end


D1=uint8([1 sj1 J1 sj2 J2 sj3 J3 sj4 J4 sj5 J5 sj6 J6]);

%Enviamos el dato
fwrite(t,D1);

T
pause(T);
```

mdl_irb120

```
%INTRODUCCIÓN DEL MODELO DEL IRB120

global herramienta

herramienta=0.215;

%Distancias entre ejes, en metros
l1=0.29;
l2=0.27;
l3=0.07;
l4=0.302;
l5=0.072;

%Definición de la cadena cinemática D-H
L(1)=Link([0 l1 0 -pi/2]);
L(2)=Link([0 0 l2 0]);

L(3)=Link([0 0 l3 -pi/2]);
L(4)=Link([0 l4 0 pi/2]);
L(5)=Link([0 0 0 -pi/2]);
L(6)=Link([0 l5 0 0]);

irb120=SerialLink(L,'name','IRB120');
irb120.offset=[0 -pi/2 0 0 0 pi];
irb120.tool=transl([0 0 herramienta]);
%Descomentar si queremos que funcione ikine6s

%Posicion nominal. Es un punto singular.
qn=[0 0 0 0 0 0];

%Otra posición para hacer pruebas
qr=[0 pi/8 -pi/6 0 pi/5 0];

%Rotación del sistema S0 al S6 en la posición de
reposo
TRi=troty(pi/2);
```

Panel_numeros

```
%Sistema de referencia del mundo, para poder
dibujarlo
Origen=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];

%Sistema de referencia del panel de números, se
puede modificar
PosPanel=transl(0.3,0,0.7)*TRi;    %TRi es la
rotación propia del efector final en la posición de
reposo
PosPanel=transl(0.5,0,0.2)*TRi*troty(pi/2); %TRi es
la rotación propia del efector final en la posición de
reposo
PosPanel=transl(0,0.5,0.6)*TRi*trotx(-
pi/2)*troty(pi/6);    %TRi es la rotación propia del
efector final en la posición de reposo

axis([-1 1 -1 1 -1 1]);hold on;grid on;
trplot(PosPanel,'color','r');
trplot(Origen);

%Escalado del panel
escala=0.1;

p1=escala*[0 0 0 1/escala]';
p2=escala*[0 1 0 1/escala]';
[…]
p12=escala*[1 0 -0.1 1/escala]';

%Coordenadas de los puntos en el sistema de
referencia total
p1T=PosPanel*p1;
[…]
p12T=PosPanel*p12;

%Dibujo de los puntos
plot3([p1T(1) p2T(1) p3T(1) p4T(1) p5T(1) p6T(1)
[…] p6T(3)],'r--');

T_Vertices(:,:,1)=PosPanel*transl(p1(1:3));
[…]
T_Vertices(:,:,12)=PosPanel*transl(p12(1:3));
```

```
%DEFINICIÓN DE LOS NÚMEROS. Terminamos
y empezamos siempre en la posición 7, que es la
%de reposo
cero=[1 2 3 4 5 6 1 7];
uno=[1 6 5 11 12 7];
dos=[8 2 1 6 3 4 5 11 12 7];
tres=[8 2 1 6 5 4 10 9 3 6 12 7];
cuatro=[8 2 3 6 12 7 1 6 5 11 12 7];
cinco=[1 2 3 6 5 4 10 9 8 7];
seis=[1 2 3 4 5 6 3 9 12 7];
siete=[8 2 1 6 5 11 12 7];
ocho=[1 2 3 4 5 6 1 7 12 6 3 9 12 7];
nueve=[12 11 5 6 1 2 3 6 12 7];

%Colocamos el robot en la posición 7, que será la de
reposo
qreposo=irb120_ikine(T_Vertices(:,:,7),[0 0 0 0 0 0]);
irb120.plot(qreposo);

%Creamos la clase para controlar el robot real
robot=irb120_socket('127.0.0.1',1024);
robot.connect;
disp('Abierto el socket de comunicación con el robot
irb120');
Treposo=T_Vertices(:,:,7);

pos=transl(Treposo);
pos=pos*1000;  %posiciones a milímetros
rot=tr2rpy(Treposo);
rot=rot*180/pi;  %ángulos a grados
robot.TCProtandpos([rot(3) rot(2) rot(1) pos(1)
pos(2) pos(3)]);
pause(5);
robot.TCProtandpos([rot(3) rot(2) rot(1) pos(1)
pos(2) pos(3)]);
pause(5);
disp('Enviado a la posicion de reposo');

robot.disconnect;
```

Aplicación_ikine_Tvertices_Robot_real

```
%APLICACIÓN FINAL: PINTAR NÚMEROS
%Resolución mediante cinemática inversa

%Abirmos la comunicación con el robot de nuevo
robot.connect;

numero=input('Introduzca el número a pintar: ');

switch (numero)
   case 0,
      num=cero;
   case 1,
      num=uno;
   case 2,
      num=dos;
   case 3,
      num=tres;
   case 4,
      num=cuatro;
   case 5,
      num=cinco;
   case 6,
      num=seis;
   case 7,
      num=siete;
   case 8,
      num=ocho;
   case 9,
      num=nueve;
   otherwise,
      error('Entrada no válida');
end

%Primero vamos de la posición de reposo hasta el
primer vértice
T2=T_Vertices(:,:,num(1));
pos=transl(T2);
pos=pos*1000;  %posiciones a milímetros
rot=tr2rpy(T2);
rot=rot*180/pi;  %ángulos a grados
robot.TCProtandpos([rot(3) rot(2) rot(1) pos(1)
pos(2) pos(3)]);
pause(2);

% Vamos recorriento de vértice en vértice
n=length(num);
q0=qreposo;
for i=1:(n-1)
   %T1=T_Vertices(:,:,num(i)); Posición de origen.
No la necesitamos
   T2=T_Vertices(:,:,num(i+1));
   pos=transl(T2);
   pos=pos*1000;  %posiciones a milímetros
   rot=tr2rpy(T2);
   rot=rot*180/pi ; %ángulos a grados
   robot.TCProtandpos([rot(3) rot(2) rot(1) pos(1)
pos(2) pos(3)]);
   pause(2);
end
```

Genera_Ts_num

```
%APLICACIÓN FINAL: PINTAR NÚMEROS

%Resolución mediante control diferencial (modelo
simulink
%'aplicacion_diferencial')

%Volvemos a abrir la conexión
global t
t=tcpip('127.0.0.1',1024);
fopen(t);

numero=input('Introduzca el número a pintar: ');

switch (numero)
   case 0,
      num=cero;
   case 1,
      num=uno;
   case 2,
      num=dos;
   case 3,
      num=tres;
   case 4,
      num=cuatro;
   case 5,
      num=cinco;
   case 6,
      num=seis;
   case 7,
      num=siete;
   case 8,
      num=ocho;
   case 9,
      num=nueve;
   otherwise,
      error('Entrada no válida');
end

%Inicializo los campos de la estructura, vacíos
Ts_num.time=[];
Ts_num.signals.values=[];
Ts_num.signals.dimensions=[4 4];


% Vamos recorriento de vértice en vértice
n=length(num);

q0=qreposo;
for i=1:(n-1)
   T1=T_Vertices(:,:,num(i));
   T2=T_Vertices(:,:,num(i+1));
   Ts=ctraj(T1,T2,20);          %10 puntos en cada
tramo de trayectoria
```

```
Ts_num.signals.values(:,:,[length(Ts_num.signals.values)+1:length(Ts_num.signals.values)+length(Ts)])=Ts;

%Ts_num.signals.values(:,:,length(Ts_num.signals.values)+1)=Ts

end
```

```
%Generamos el vector de tiempos, cada 100 ms (como son 10 pasos, cada tramo
%tarda 1 segundo)
paso=2;   %paso de tiempo
Ts_num.time=[0:paso:(length(Ts_num.signals.values)-1)*paso];

%Finalización de la simulación
Tstop=Ts_num.time(end);
```

Ini_robotics

```
addpath('C:\Program Files\MATLAB\R2012a\toolbox\rvctools');

startup_rvc
```

Algunos de los programas anteriores hacen uso de otros para calcular ciertas funciones o dar pasos intermedios. Esos programas se adjuntan también en el CD, a continuación se nombran:

Ikine_irb120

Irb120_ikine

MDH.m

9 Presupuesto

Este capítulo proporciona información detallada acerca de los costes teóricos del desarrollo del proyecto, incluyendo los costes de materiales y las tasas profesionales.

9.1 Costes materiales

Objeto		*Cantidad*	*Precio unidad (€)*	*Precio total (€)*
Hardware	Portátil Sony	1	700€	700€
	ABB-IRB120	1	10954,00	10954,00
Software	Microsoft Windows 7	1	0,00	0,00
	RobotStudio	1	0,00	0,00
	Matlab R2012a	1	0,00	0,00
	Microsoft Office Home and Student 2010	1	100,00	100,00
TOTAL				11754,00

Tabla 7: Costes materiales (hardware y software) sin IVA.

9.2 Costes totales

Los costes totales del Proyecto han sido obtenidos sumando los costes materiales y profesionales aplicando el IVA. Además hay que tener en cuenta los costes de impresión y encuadernación.

Objeto		***Costes totales***
Costes materiales		11754,00 €
Costes profesionales		6000,00 €
Proyecto	Impresión	60 €
	Encuadernación	40 €
Subtotal		17854 €
IVA (21%)		3749 €
TOTAL		21603 €

Tabla 8: Costes totales con IVA.

10 Bibliografía

[1] Trabajo Fin de Grado: "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station", Marek Jerzy Frydrysiak

[2] Fundamentos de Robótica, Antonio Barrientos; McGraw Hill

[3] Manual de referencia técnica. Instrucciones, funciones y tipos de datos de RAPID

[4] Robotics Toolbox. Peter Corke. User's Guide

[5] Instrument Control Toolbox 2. User's Guide

[6] ManualOperador-Robostudio ,

[7] http://www.mathworks.es/

[8] http://digital.ni.com/

[9]http://www.abb.es/product/seitp327/26a64e26204118e5c12576a4004a8497.aspx