Trabajo Fin de Grado

Desarrollo de un sistema plóter basado en un robot ABB IRB120

'Development of a plotter system based on an ABB IRB120 Robot'

Autor:

Eduardo Renta García

Director:

Gonzalo López Nicolás

2019

Agradecimientos

A mi Madre, sin tu apoyo constante y tu ejemplo de lucha no hubiera sido capaz de llegar hasta aquí.

A mi Padre, por la educación que me brindaste desde pequeño, allá donde estés espero que te sientas orgulloso de mí.

A mi hermano, por enseñarme a dar lo mejor de mí.

A mi familia, en especial a mis abuelos, por vuestras lecciones, consejos y ánimos.

A mis compañeros de Grado y amigos, por vuestra ayuda durante esta etapa.

Y a mí tutor, Gonzalo, gracias por tu paciencia y por dedicarme tu tiempo para poder desarrollar este proyecto.

Gracias a todos.

Resumen

En la actualidad, el uso de robots industriales se ha extendido a una gran variedad de aplicaciones en la industria gracias a su flexibilidad y fiabilidad. Este tipo de robots nos permite lograr buena calidad de producto con unos costes contenidos.

En general, el motivo de desarrollo de este proyecto es el uso de la tecnología robótica para lograr la reproducción de imágenes digitales en papel. Entre los objetivos principales está el aprendizaje de herramientas de visión por computador y de robótica industrial, junto con diferentes programas y lenguajes de programación comerciales. Estas herramientas nos permiten desarrollar una aplicación que será simulada e implantada en un robot, trabajando aspectos de visión y robótica. En particular, se aborda el control de un robot industrial mediante un programa que, a partir de una imagen proporcionada por el usuario, dibuja una copia de dicha imagen en papel mediante un rotulador incorporado como elemento terminal del robot.

Para cumplir los objetivos del trabajo se ha dividido la tarea propuesta en la resolución de varios problemas. El primero, conseguir que los datos contenidos en una imagen digital fueran comprensibles por un robot industrial, es decir, convertir los píxeles que componen la imagen en posiciones alcanzables por el Robot. Esta adaptación se ha logrado mediante funciones de tratamiento de imagen. Una vez codificada la imagen de forma apta para el robot surge el segundo problema, consistente en lograr transmitir la información del dibujo al robot. Para solventarlo se ha decidido realizar la transferencia de datos utilizando una estructura cliente-servidor del protocolo TCP/IP. Para comprobar el funcionamiento del sistema se ha realizado un gran número de simulaciones en el software RobotStudio, que permite emular de forma realista el comportamiento del Robot. Además de esto, para facilitar el uso de las aplicaciones desarrolladas se ha diseñado una interfaz gráfica que permite la integración de todas las funciones en un panel de control de fácil acceso y uso intuitivo.

Por último, se ha realizado una evaluación experimental en el laboratorio de la universidad utilizando el robot ABB IRB120 y hemos verificado la calidad de las simulaciones realizadas.

Palabras clave: Robot, IRB 120, MATLAB, Robot Studio, socket.

Índice

Lista de figuras....11

1. Introducción....15

1.1 Estado de la materia....15

1.2 Objetivos y motivación....16

1.3 Estructura de la memoria....16

1.4 Esquema de funcionamiento. Flujo de información....17

2. Herramientas....18

2.1 Matlab....18

2.2 ABB Robotics....19

2.2.1 Brazo robótico IRB120....19

2.2.2 Controlador IRC5 de ABB....21

2.2.3 Robot Studio....22

3. Procesamiento de imagen....25

3.1 Tratamiento de la imagen en Matlab....25

3.2 Binarización de la imagen....27

3.3 Filtrado de imagen....28

4. Protocolo de comunicación....32

4.1 Socket de comunicación. TCP/IP....32

4.1.1 Protocolo TCP/IP....32

4.2 Establecimiento de comunicación....33

4.3 Transmisión de datos....34

4.3.1 Emisión de datos desde Matlab....35

4.3.2 Recepción de datos en RobotStudio....37

4.3.3 Comprobación de posición. Interacción entre funciones 'Check'(Matlab) y 'SendPosition' (RobotStudio)....39

5. Interfaz gráfica de control en Matlab. GUI....41

5.1 Código de inicialización de la GUI....42

5.2 Conexión....43

5.3 Selección de imagen....45

5.4 Calibración....46

5.4.1 Mover robot a posición X,Y,Z....47

5.4.2 Configurar altura de dibujo Z....49

5.4.3 Configurar velocidades de dibujo y aproximación....50

5.5 Selección de filtro y previsualización de imagen....53

5.6 Visualización de imagen....54

5.7 Posición del robot 54

5.8 Comenzar a dibujar 55

5.8.1 Función Redimensión 57

5.8.2 Función Dibuja 58

5.8.3 Función NextPixel 60

5.9 Botón 'Stop' para parada de emergencia 61

6. Resultados experimentales 63

6.1 Comparativa para tipos de imágenes y filtros 63

6.2 Simulaciones en ordenador 64

6.2.1 Dibujo del logo de la universidad. Filtro Edge. 65

6.2.2 Dibujo de un Mandala. Filtro Thicken 67

6.3 Resultados en laboratorio 70

7. Conclusiones y trabajos futuros 75

8. Bibliografía 76

Anexo 1. Manual de usuario 78

I.1 Configuración previa de la estación en RobotStudio. 79

I.2 Transferencia de datos al controlador. 81

I.3 Puesta en marcha del programa 82

I.4 Funcionamiento de la interfaz 'Panel de Control' 83

I.4.1 Conexión 84

I.4.2 Selección de imagen 84

I.4.3 Selección de filtro y previsualización 84

I.4.4 Calibración 85

I.4.5 Comienzo del dibujo 86

I.4.6 Parada de emergencia 86

I.5 Tabla de funcionalidad de los elementos del panel de control 87

Anexo II. Código 91

II.1 Matlab 91

II.1.1 Conexión 91

II.1.2 Función 'Redimensión' 91

III.1.3 Función 'Dibuja' 92

III.1.4 Función 'Check' 93

III.1.5 Función 'Thicken' 94

III.1.6 Función 'Skeleton' 94

III.1.7 Función 'Edge' 95

III.1.8 Función 'Pixels' 96
II.1.9 Función 'NextPixel' 97
II.1.10 Interfaz gráfica (GUI) 'PanelControl' 98
II.2 RobotStudio 106
II.2.1 Modulo principal 'Mod1' 106
II.2.2 Módulo 'Communication' 107
III.2.3 Módulo 'SendPosition' 110
III.2.4 Módulo 'VelRobot' 110

Lista de figuras

Figura 1.1 E-David 15
Figura 1.2. Esquema de funcionamiento 17
Figura 2.1. Interfaz de Matlab........ 18
Figura 2.2. Robot ABB IRB120........ 19
Figura 2.3. Alcance del robot 20
Figura 2.4. Radio de acción 20
Figura 2.5. Posiciones de trabajo del robot 21
Figura 2.6 Controlador IRC5........ 21
Figura 2.7. Flex pendant........ 22
Figura 2.8. Interfaz de RobotStudio........ 22
Figura 2.9. Estructura de un programa en RAPID 23
Figura 3.1. Composición de una imagen........ 25
Figura 3.2. Composición de una imagen a color........ 26
Figura 3.3 Logo de la universidad de Zaragoza 26
Figura 3.4 Función imread 27
Figura 3.5 Tipo de datos de la imagen 27
Figura 3.6. Criterio de Umbralización 27
Figura 3.7. Proceso de binarización 28
Figura 3.8 Comparativa de umbrales........ 28
Figura 3.9 Obtención del valor Umbral........ 28
Figura 3.13. Imagen sometida a filtro ‘Edge’ 29
Figura 3.14. Imagen sometida a filtro ‘Thicken’ 29
Figura 3.15. Imagen sometida a filtro ‘Skeleton’........ 30
Figura 3.16. Imagen sometida a método ‘Pixels’........ 30
Figura 3.17. Comparación de filtros para ‘Stormtropper’ 30
Figura 3.18. Comparación de filtros para ‘Bender’ 31
Figura 4.1. Estructura Cliente/Servidor 32
Figura 4.2. Declaración de Cliente y Servidor en RobotStudio 33
Figura 4.3. Declaración de dirección IP y puerto 33
Figura 4.4. Establecimiento de comunicación en RobotStudio 34
Figura 4.5. Establecimiento de comunicación en Matlab........ 34
Figura 4.6. Bucle de la función ‘Dibuja’........ 35
Figura 4.7. Layout de la instalación........ 35
Figura 4.8. Criterio de identificación de puntos 36
Figura 4.9. Criterio de rotación de imagen 36
Figura 4.10. Forma de envío de los puntos de la línea 36
Figura 4.11. Calculo de la distancia entre puntos........ 36
Figura 4.12. Bucle de la función ‘NextPixel’........ 37
Figura 4.13. Comienzo del procedimiento de recepción de datos 37
Figura 4.14. Dibujado de puntos en RobotStudio........ 38
Figura 4.15. Función para saltos entre objetos del dibujo 38

Figura 4.16. Procedimiento 'SendPosition' ... 39
Figura 4.17. Procedimiento 'Check' ... 39
Figura 5.1. Interfaz Panel de Control ... 41
Figura 5.2. Código de inicialización ... 42
Figura 5.3. Funciones OpeningFcn y OutputFcn ... 43
Figura 5.4. Cuadro de conexión ... 43
Figura 5.5. Funciones para obtener la direccion IP ... 44
Figura 5.6. Funciones para obtener el Puerto de conexión ... 44
Figura 5.7. Función para establecer conexión ... 44
Figura 5.8. Procedimiento de conexión ... 45
Figura 5.9. Cuadro de selección de imagen ... 45
Figura 5.10. Función para seleccionar imagen ... 45
Figura 5.11. Ejemplo de imagen seleccionada en Panel de Control ... 46
Figura 5.12. Cuadro de calibración ... 46
Figura 5.13. Ejemplo para obtener coordenada X ... 47
Figura 5.14. Función 'IrPosición' ... 48
Figura 5.15. Procedimiento para ir a punto en RobotStudio ... 48
Figura 5.16. Instalación reproducida en RobotStudio ... 49
Figura 5.17. Función para configurar altura de dibujo Z ... 49
Figura 5.18. Procedimiento para calibrar altura de dibujo Z en RobotStudio ... 50
Figura 5.19. Selección de velocidades ... 50
Figura 5.20. Función para selección de velocidad de dibujo ... 51
Figura 5.21. Función para enviar velocidades al Robot ... 51
Figura 5.22. Procedimiento de calibración de velocidad ... 52
Figura 5.23. Procedimiento para transformar velocidad al tipo 'Speeddata' ... 52
Figura 5.24. Cuadro de selección de filtro ... 53
Figura 5.25. Función para selecci ... 53
Figura 5.26. Imagen mandala1 original ... 54
Figura 5.27. Imagen mandala1 filtrada ... 54
Figura 5.28. Cuadro de coordenadas de posición ... 55
Figura 5.29. Función para mostrar coordenadas en pantalla ... 55
Figura 5.30. Trayectoria del Robot trazada en Matlab ... 55
Figura 5.31. Botón Dibujar ... 56
Figura 5.32. Función para ejecutar el dibujo ... 57
Figura 5.33. Función redimensión ... 58
Figura 5.34. Layout de la isntalación ... 58
Figura 5.35. Función 'Dibuja' ... 59
Figura 5.36. Función NextPixel ... 61
Figura 5.37. Botón Stop ... 61
Figura 5.38. Función Stop ... 61
Figura 5.39. Procedimiento para frenar el Robot ... 62
Figura 6.1. Imágenes para probar ... 63

Figura 6.2. Comparativa de filtros usados 63
Figura 6.3. Ejemplo para el filtro 'Edge' 64
Figura 6.4. Ejemplo para el filtro 'Thicken' 64
Figura 6.5. Imagen de la Instalación 65
Figura 6.6. Logo de la universidad de Zaragoza 65
Figura 6.7. Logo filtrado por método 'Edge' 66
Figura 6.8. Resultado de la simulación en RobotStudio 66
Figura 6.9. Trazado del Robot representado en Matlab 67
Figura 6.10. Imagen Mandala 67
Figura 6.11. Imagen filtrada por método 'Thicken' 68
Figura 6.12. Resultado de la simulación 69
Figura 6.13. Trazado del Robot representado en Matlab 69
Figura 6.14. Montaje en el laboratorio 70
Figura 6.15. Logo de la universidad dibujado por el Robot real 71
Figura 6.16. Mandala dibujado por el Robot real 71
Figura 6.17. Bender dibujado por el Robot real 72
Figura 6.18. Figuras de Escher dibujadas por el Robot real 73
Figura 6.19. Tres en raya dibujado por el Robot real 73

Figuras de Anexos

Figura I.1. Inicializar aplicación en Matlab 79
Figura I.2 Administrador de la instalación 79
Figura I.4 Selección de la copia de seguridad 80
Figura I.5 Selección de PC Interface en opciones de comunicación 81
Figura I.6 Selección de los controladores 82
Figura I.7 Selección de la ventana de producción 82
Figura I.8 Selección del modo automático en el IRC5 83
Figura I.9 Puesta en marcha de la simulación 83
Figura I.9 Interfaz del panel de control 84
Figura I.10 Selección de filtro 85
Figura I.11 Opciones de calibración 85
Figura I.12 Botón dibujar 86
Figura I.13 Seta de emergencia del Flex Pendant 86
Figura I.14 Seta de emergencia del controlador IRC5 87
Figura I.15 Botón de parada del panel de control 87
Figura II.1. Interfaz de Control 106

1. Introducción

1.1 Estado de la materia

En los últimos años la robótica ha sufrido un importante desarrollo. Estos avances han venido impulsados por la necesidad de la industria de aumentar el volumen de producción reduciendo costes, mejorando la calidad del producto final, además de reducir los accidentes laborales y en muchos casos mejorar también la flexibilidad de fabricación de productos.

Fue a mitad del siglo pasado, cuando en el año 1956 se patentó el primer robot industrial (basado en los diseños de George Devol), usado para transferir objetos de un punto a otro de forma automática. Desde entonces se han desarrollado multitud de aplicaciones (soldadura, pintura, ensamblado, almacenaje…), teniendo gran importancia en el desarrollo de sectores industriales como el de la automoción.

Actualmente los principales retos son por una parte la integración de sistemas de visión artificial que permitan a la máquina localizar objetos en el espacio y manipularlos sin la necesidad de establecer trayectorias fijas. Y por otra parte, está cogiendo fuerza el conocido como '*machine learning'*, rama de la inteligencia artificial cuyo objetivo es desarrollar técnicas que permitan que las maquinas 'aprendan' con la experiencia.

Dentro de la temática y alcance de este proyecto se han desarrollado con anterioridad avances en el campo de la pintura y reproducción de imágenes mediante robots industriales. Un ejemplo de ello es el robot *e-David*, desarrollado por investigadores en la Universidad de Constanza:

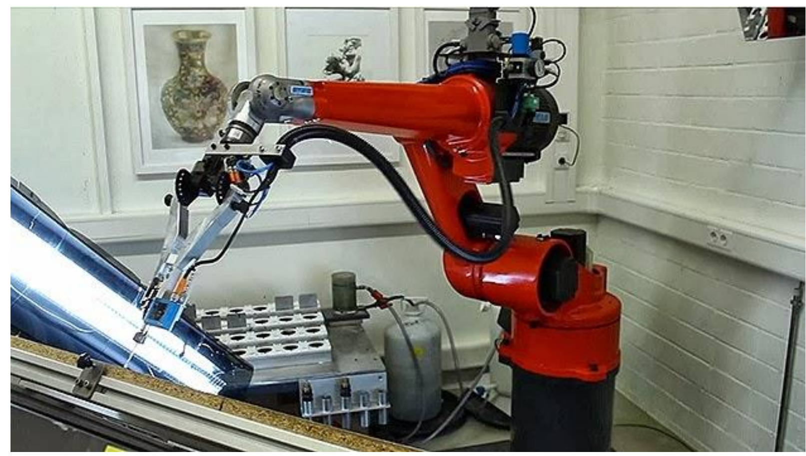


Figura 1.1 *E-David*

Este robot es capaz de intercambiar diferentes pinceles (tiene disponible una paleta de 24 colores) y bolígrafos, calculando las pinceladas necesarias para reproducir una imagen que previamente se le ha enviado como input. El gran avance de este robot frente a otros proyectos de parecida índole viene al introducir pasos iterativos en el pintado de una misma trayectoria, en vez de establecer solo una trayectoria perfecta de la pincelada, consiguiendo así mejores resultados. El sistema lleva integrado control por visión artificial, lo que le permite tener *feedback* sobre las pinceladas realizadas y calcular las siguientes.

El objetivo del proyecto será similar al del proyecto e-David, consiguiendo generar las trayectorias necesarias para reproducir la imagen pero con un menor nivel de complejidad ya que se planteará para trabajos futuros el intercambio de herramienta y el control en lazo cerrado por visión artificial.

1.2 Objetivos y motivación

En este proyecto se planteará el problema de la reproducción de una imagen digital seleccionada por el usuario. El objetivo será dibujarla en papel con la mayor calidad posible. Para lograrlo haremos uso de un robot industrial, que estará equipado con un rotulador en su extremo terminal.

Se abordarán maneras diferentes de reproducir la imagen, usando cuatro filtros de tratamiento de imagen distintos que nos permitan enviar al robot la información de manera comprensible, esto es, en forma de coordenadas que compondrán trayectorias por las que irse desplazando.

Se compararán los filtros en función del tiempo necesario para reproducir la imagen y la similitud entre la imagen original y la reproducción.

Por último se desarrollará una interfaz gráfica GUI con la que el usuario sea capaz de controlar el sistema de una manera simple e intuitiva.

1.3 Estructura de la memoria

En este apartado se explica cómo está distribuida la memoria de este trabajo de fin de grado.

Comenzando por este punto, el número 1 en el que se ha explicado el desarrollo de la tecnología de la que se hace uso, los objetivos de este proyecto y la estructura de la aplicación desarrollada.

En el punto número 2 se exponen los programas y herramientas con los que se ha trabajado, el software Robot Studio de ABB y su código de programación asociado RAPID, el Robot industrial IRB 120 de ABB y el software matemático MATLAB donde se desarrolla la interfaz de control del Robot.

En el punto 3 se detalla el procedimiento de tratamiento de imagen utilizado en Matlab, cómo se binariza el valor de los pixeles que componen la imagen y se somete al filtro para obtener los contornos de las figuras que contiene la imagen.

En el apartado nº 4 se explica el protocolo de comunicación utilizado y el código desarrollado para que se produzca el intercambio de información entre ambos programas.

Para facilitar la labor del usuario en el punto número 5 se explica el código que hay tras la interfaz gráfica desarrollada.

Finalmente se documentan los resultados de las simulaciones tanto en ordenador como en el laboratorio con el robot real para acabar comentando las conclusiones y los posibles trabajos futuros en base a la aplicación desarrollada.

1.4 Esquema de funcionamiento. Flujo de información

Para el desarrollo de la aplicación indicada en el proyecto se ha hecho uso de los programas de simulación Matlab y RobotStudio y el robot industrial de ABB IRB120. Más adelante detallaremos sus características.

El propósito de nuestra aplicación es reproducir una imagen elegida por el usuario mediante nuestro robot. Para conseguirlo tendremos que atravesar varios pasos en los que ir tratando y transmitiendo la información necesaria.

En el esquema de la **Figura 1.2** podemos ver como es el flujo de información desde que tomamos la imagen hasta que llega al Robot, bien en su versión simulada o en la versión real. El primer punto de tratamiento será en Matlab donde transformaremos la imagen en datos comprensibles por el Robot, después mediante el socket TCP/IP le enviaremos esa información de manera ordenada.

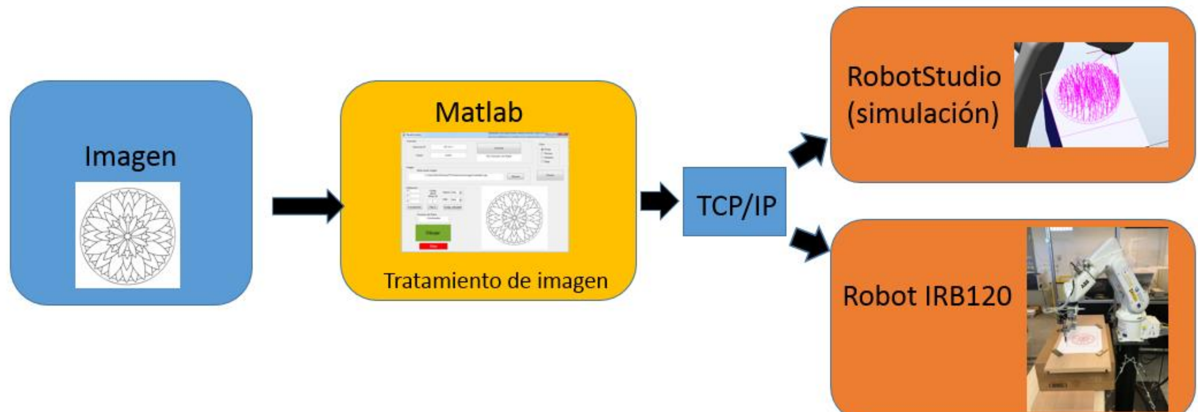


Figura 1.2. *Esquema de funcionamiento*

2. Herramientas

En este capítulo se van a presentar las herramientas utilizadas en el proyecto. Se introducirán las principales características tanto del software, Matlab y RobotStudio, como de la parte de hardware compuesta por el robot ABB IRB120 y su controladora.

2.1 Matlab

MATLAB (abreviatura de *MATrix LABoratory*, "laboratorio de matrices") es una herramienta de software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). En este proyecto usaremos la versión R2017b de Matlab.

Esta aplicación nos permite operar con vectores, matrices, funciones y además la programación orientada a objetos.

Integra también herramientas adicionales:

- Simulink, plataforma de simulación multidominio.
- GUIDE, editor de interfaces de usuario.
- Toolboxes, que son pequeñas aplicaciones dentro de Matlab que integran diversas funciones dependiendo de su cometido (por ejemplo la Toolbox Image Processing nos permite editar imágenes).

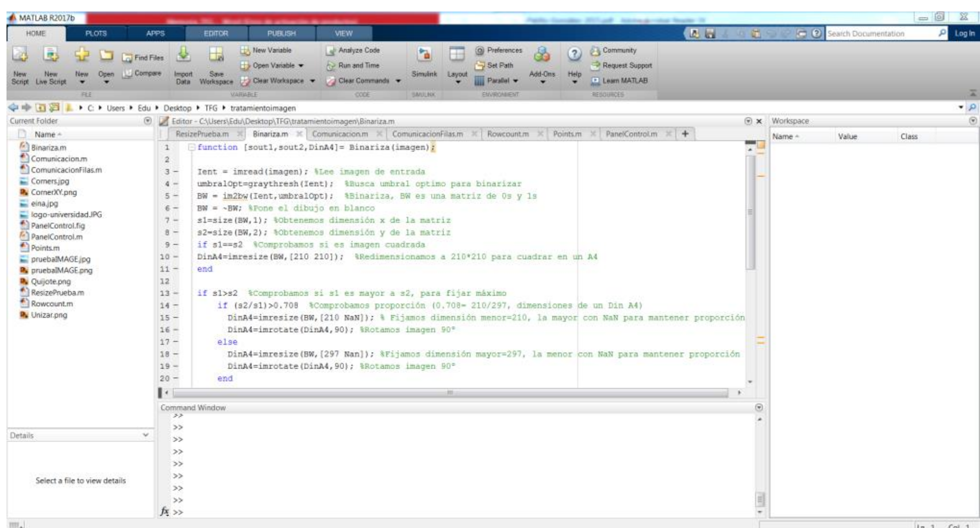


Figura 2.1. *Interfaz de Matlab*

En nuestro caso gracias a este software trataremos una imagen, seleccionada por el usuario, desarrollando varios programas en código M. El primer cometido de esta aplicación será poder descomponer la imagen en datos que puedan ser enviados a RobotStudio.

La siguiente parte será desarrollar otro programa que nos permita comunicar vía TCP/IP al software emisor de datos MATLAB y al Robot receptor.

Por último se desarrollara una interfaz gráfica GUI que permita al usuario controlar el proceso directamente desde una pantalla intuitiva sin tener que modificar el código del programa directamente.

2.2 ABB Robotics

ABB (Asea Brown Boveri) es una reconocida corporación multinacional especializada en ingeniería eléctrica y automatización industrial. Su sede se sitúa en Zúrich (Suiza) y fue fundada en 1988, tras la fusión de la empresa sueca ASEA y el fabricante suizo BBC (Brown Boveri & Cie). En España la sede se sitúa en Madrid y cuenta con fábricas en Zaragoza y Córdoba, entre otras.

Entre sus sectores de negocio en la actualidad (2018) encontramos:

- Redes eléctricas
- Productos eléctricos de media y baja tensión
- Robótica
- Automatización Industrial

  En este trabajo haremos uso de varios de sus productos de las ramas de robótica y automatización industrial.

2.2.1 Brazo robótico IRB120

ABB cuenta con 38 modelos de Robots distintos en su catálogo en la actualidad entre los diversos tipos (Articulados, Colaborativos, SCARA y Robots de Pintura). En nuestro caso el brazo robótico que usaremos será el IRB 120 (**Figura 2.2**), el más pequeño de la gama.

Figura 2.2. *Robot ABB IRB120*

2.2.1.1 Características técnicas

El robot IRB120 pesa 25 kg, y puede soportar pesos de hasta 3 kg con un alcance de 58 cm (**Figuras 2.3 y 2.4**):

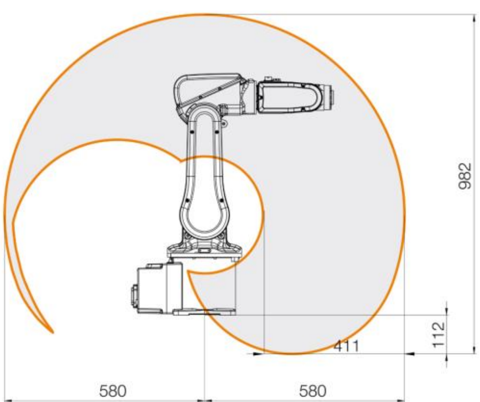


Figura 2.3. *Alcance del robot*

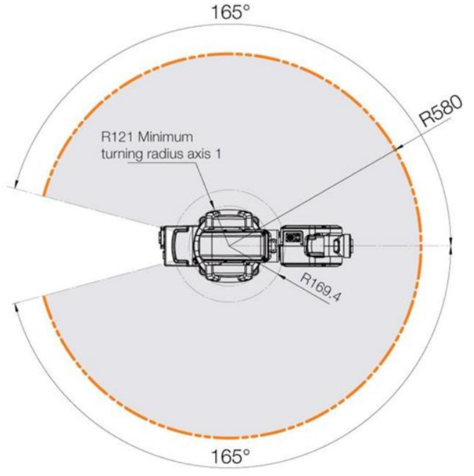


Figura 2.4. *Radio de acción*

Su máxima velocidad (referida al *Tool Center Point, TCP*) es de 6.2 m/s y su aceleración máxima de 28 m/s^2. Su precisión es +-0.01 mm.

A pesar de que está diseñado pensando en su trabajo en entornos industriales estas características hacen que sea un robot ideal para nuestra aplicación, además de poder integrarlo fácilmente en el laboratorio de la universidad.

Puede ser fijado en distintos planos (**Figura 2.5**), según la tarea que se vaya a desarrollar, en nuestro caso se colocará con la base apoyada sobre el suelo.

Figura 2.5. *Posiciones de trabajo del robot*

2.2.2 Controlador IRC5 de ABB

Para dirigir los movimientos del brazo robótico IRB120 necesitamos una centralita de control a través de la que podamos realizar la interacción máquina-humano. Dentro de la gama de controladores de ABB en nuestro caso usaremos el controlador IRC5 (**Figura 2.6**). Desde este equipo se envían y reciben las señales para controlar el brazo robótico y dentro de su memoria se almacenan los programas que rigen sus operaciones a realizar.

Figura 2.6 *Controlador IRC5*

El controlador IRC5 viene a su vez equipado junto a un mando de control que simplifica la programación del robot, con una interfaz intuitiva, llamado FlexPendant (**Figura 2.7**). Este equipo consta de un *Joystick*, botonera (incluyendo la seta de emergencia) y pantalla táctil y sirve para ejecutar los programas cargados en el controlador así como realizar ajustes. En su pantalla se puede seguir el estado de ejecución de las tareas del brazo robótico durante su funcionamiento.

Figura 2.7. *Flex Pendant*

Dentro del controlador encontramos dos módulos:

- Módulo de accionamiento, desde el cuál se controla la energía que se dirige a los motores del robot.
- Módulo de control, donde se alberga el ordenador principal, interruptor principal, panel del operador, conexión para el FlexPendant, puertos de servicio y equipos adicionales del cliente (como ejemplo tarjetas de Entradas/Salidas).
  Dentro de este módulo se guarda el software del sistema (*RobotWare*).

2.2.3 Robot Studio

A la hora de realizar simulaciones previas a la instalación física de una estación robótica ABB dispone de la herramienta Robot Studio. Este software nos permite construir y visualizar en 3D el funcionamiento de nuestra instalación (**Figura 2.8**).

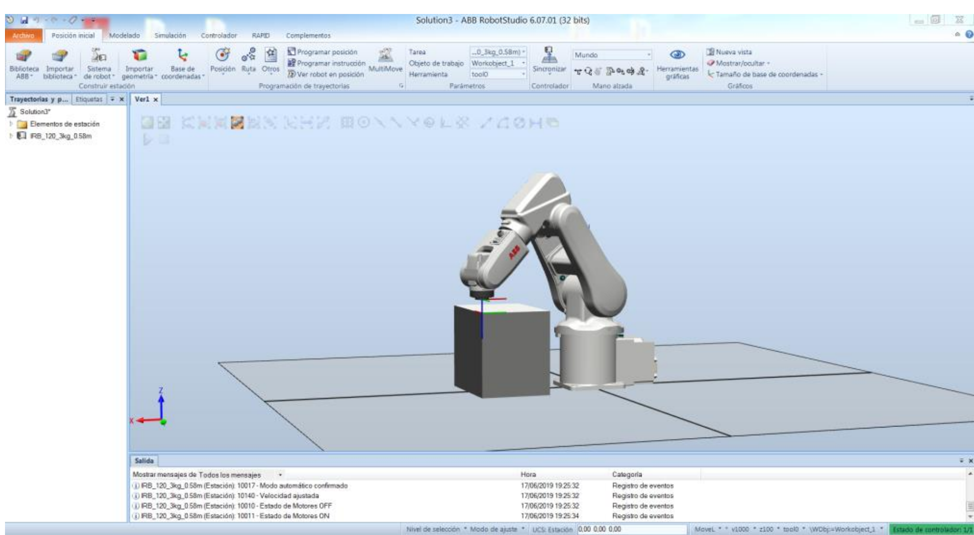


Figura 2.8. *Interfaz de RobotStudio*

Gracias a estas simulaciones podemos estudiar previamente el comportamiento y la interacción de los elementos del proceso (robots, *conveyors*, vallado de seguridad…) que queremos implantar. Para el análisis y optimización del proceso nos permite medir tiempos así como marcar trayectorias recorridas por el *end-effector* del robot.

Este programa está basado en el software real que usan los robots de ABB por lo que de una manera sencilla podemos crear el ciclo del robot que queremos programar (o robots si hubiera más de uno en nuestra instalación) para su posterior sincronización con la instalación real. Esto nos permite ahorrar costes, reducir riesgos de seguridad (posee un modo de detección de colisiones) y la posibilidad de ir implementando *loops* de mejora de la instalación mientras está ya en funcionamiento sin tener que detener la producción.

Para la construcción de nuestra estación en 3D disponemos de una amplia biblioteca de Robots, *end-effectors* y otros equipos tanto de la empresa ABB como de otras empresas especializadas como *Binzel*. Además si queremos podemos diseñar directamente en el programa nuevos elementos o importar CAD desde otros programas como *CATIA* o *Autocad*, ya que admite formatos *STEP, IGES, ACIS, VRML Y VDAFS*.

2.2.3.1 Lenguaje de programación. RAPID.

El lenguaje de programación que emplea RobotStudio es el lenguaje RAPID (*Robotics Application Programming Interactive Dialogue*). Es similar a otros lenguajes de programación de propósito general de alto nivel como *C o Pascal*.

Gracias a este código podemos construir estructuras lógicas comprensibles por el usuario y a la vez por el robot, para desempeñar las tareas que nosotros programemos.

El programa se compone de una serie de instrucciones o comandos que controlan el robot y se divide en tres partes:

- Rutina principal (*main*), módulo principal del programa.
- Subrutinas, pequeños programas a los que se va llamando en la rutina principal, esto nos da la ventaja de dividir el programa en módulos más pequeños.
- Datos del programa.

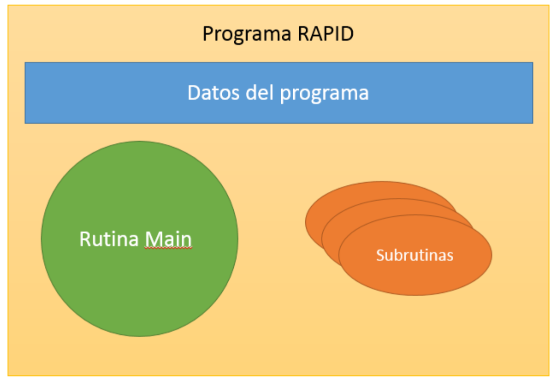


Figura 2.9. *Estructura de un programa en RAPID*

Dentro de los datos del programa podemos encontrarnos con 2 tipos de datos dependiendo de su alcance:

- Globales (GLOBAL), se puede acceder a ellos desde todos los módulos del programa
- Locales (LOCAL), sólo se accede en el módulo que estén definidos.

Dependiendo de la persistencia podemos encontrarnos con tres tipos distintos:

- Constantes (CONST), contienen valores fijos los cuáles se asignan en el momento de la declaración.
- Variables (VAR), contienen valores de datos. Si pausamos el programa el valor se conserva pero si lo paramos y reseteamos perdemos este valor.
- Variables persistentes (PERS), son un tipo de variables especiales que nos permiten almacenar el valor del dato incluso si se resetea el programa.

En este trabajo haremos uso del código Rapid para que el robot reciba datos enviados vía socket con protocolo de comunicación TCP/IP y realice los movimientos adecuados en consonancia a la información recibida. En el apartado 4 se explicaran los fundamentos del protocolo de comunicación elegido.

3. Procesamiento de imagen

El punto de partida de la aplicación es una imagen, seleccionada por usuario, que deberemos someter a diferentes procesos antes de poder reproducirla con el Robot. Este tratamiento nos descompondrá la imagen y permitirá la transferencia de datos al robot.

3.1 Tratamiento de la imagen en Matlab

Antes de encarar el proceso de tratamiento de la imagen vamos a explicar una serie de conceptos básicos acerca de cómo están compuestas las fotografías o dibujos digitales.

Una imagen digital consiste en una serie de elementos discretos llamados píxeles. Estos elementos son la unidad mínima de división como se muestra en la **figura 3.1.**

La imagen se puede descomponer en forma matricial y se podría entender como una función f(x, y) donde x e y son las coordenadas que ocupa cada pixel, compuesto por un valor f que es la intensidad de la imagen en dicho punto.

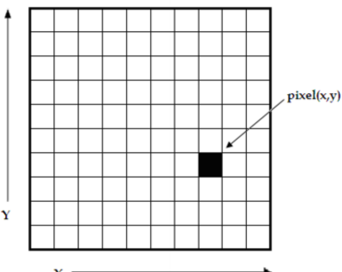


Figura 3.1. *Composición de una imagen*

La intensidad de los colores primarios de la luz se conoce como composición RGB (de sus siglas Red-Green-Blue en inglés). Estos tres colores son mezclados en diferentes proporciones para obtener el color del pixel. Cada color se suele codificar con un Byte (8 bits), por tanto la intensidad de cada color va desde 0 si no interviene a 255 como valor máximo. Se representan en una matriz de la forma (R, G, B):

- El rojo de obtiene con (255,0,0)
- El verde se obtiene con (0,255,0)
- El azul se obtiene con (0,0,255)

Por ejemplo si quisiéramos obtener el blanco tendríamos un pixel con el valor (255, 255, 255).

En Matlab esta composición se hace por 3 matrices, una para el valor del plano Rojo, otra para el plano Verde y la última para el plano Azul. Si quisiéramos obtener el valor de los pixeles correspondientes a estos planos se ejecutarían los comandos:

- Rojo=Imagen (:,:,1)
- Verde=Imagen(:,:,2)
- Azul=Imagen(:,:,3)

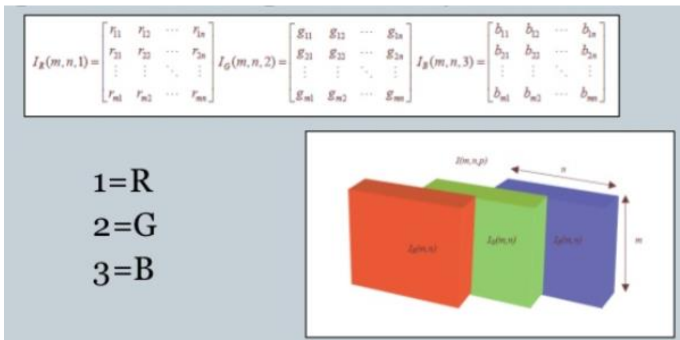


Figura 3.2. *Composición de una imagen a color*

Estas matrices serán de un tamaño MxN, siendo M el número de pixeles de la dimensión X y N el número de píxeles de la dimensión Y. El tipo de dato que Matlab almacena en cada píxel puede ser de distintos tipos:

- Matrices reales:
    1. *Single*, valores de precisión única con punto flotante, se almacenan en 4 bytes (32 bits)
    2. *Double,* valores de doble precisión con punto flotante, se almacenan en 8 bytes (64 bits).
- Matrices enteras:
    1. *Int*, pueden ser de 8, 16, 32 o 64 bits (int8, int16, int32 o int64), valores enteros. Ejemplo int8 cubre el rango [-128,127].
    2. *Uint*, pueden ser de 8, 16, 32 o 64 bits, enteros positivos (uint8, uint16, uint32 o uint64). Ejemplo uint8 cubre el rango [0,255].
- Otros tipos:
    1. *Logical*, los valores son 0 ó 1 (1 bit cada elemento).
    2. *Char*, información compuesta por caracteres.
    3. *Cell*, tipo de vectores que contienen datos en campos accesibles por índice numérico.
    4. *Struct*, tipo de vectores que contienen datos en campos accesibles por nombre.

El tipo de dato por defecto para trabajar con imágenes en Matlab es *uint8*.

Usaremos como la imagen Figura 3.3 (*''logo-universidad.jpg''*) para probar nuestra aplicación:

Figura 3.3 *Logo de la universidad de Zaragoza*

Para abrir esa imagen y crear una variable en forma de matriz que contenga sus valores se usa la función *imread.* Llamamos a esta variable *Ient* (imagen de entrada):

```
Ient = imread('logo-universidad.jpg');
```

Figura 3.4 *Función imread*

Al ejecutar el comando se genera una variable en nuestro "*WorkSpace*" (**Figura 3.5**), vemos que la imagen se traduce en una matriz de 591 x 1890 pixeles (*M x N*) y 3 planos, compuestas por datos *uint8*, números enteros positivos que toman valores de [0, 255].

Figura 3.5 *Tipo de datos de la imagen*

3.2 Binarización de la imagen

Nuestro Robot estará equipado con un rotulador en su extremo por lo que reproduciremos la imagen en Blanco y Negro. La imagen que tomamos de inicio está compuesta por varios colores por lo que el primer paso será obtener esa imagen en Blanco y Negro, es decir una matriz de 1s y 0s.

El proceso de umbralización consiste en convertir una escala de grises en una con solo dos niveles, para que se distinga el cuerpo y el fondo. A cada píxel se le asigna un segmento (**Figura 3.6**), dependiendo de su valor (comprendido entre 0, negro, y 255, blanco).

$$T_{global_n}(g) = \begin{cases} 0 & \text{si } g < t_1 \\ 1 & \text{si } t_1 \leq g < t_2 \\ \vdots & \vdots \\ n & \text{si } g \geq t_{n-1} \end{cases}$$

Figura 3.6. *Criterio de Umbralización*

Donde g es el valor de un píxel y $t_{1..n}$ el valor del umbral. En nuestro caso solo necesitaremos dos segmentos para el cuerpo de los objetos (negro) y el fondo de la imagen (blanco).

Esquemáticamente el proceso se dividiría en 2 pasos, la imagen atravesaría 3 estados (**Figura 3.7**).

Figura 3.7. *Proceso de binarización*

En Matlab no necesitaremos atravesar un paso intermedio para transformar la imagen de color a una imagen en escala de grises, podemos transformarla directamente a blanco y negro. Primero tomamos la imagen a color, obtenemos el valor umbral y mediante el comando *im2bw (image, umbral*) asignamos a los pixeles un valor de 0 si es negro o 1 si es blanco.

El umbral puede variar según necesidad del usuario, en un rango de valores de 0 a 1:

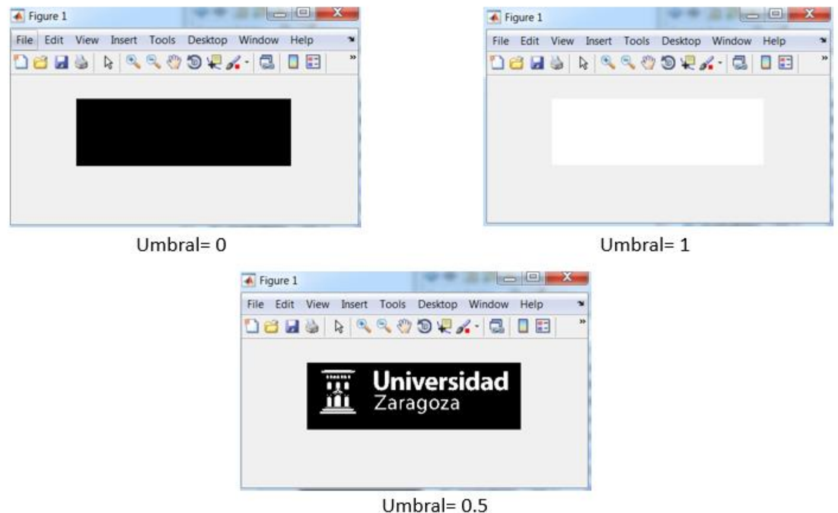


Figura 3.8 *Comparativa de umbrales*

Matlab es capaz de optimizar el valor del umbral automáticamente, mediante el comando *graythresh (imagen).* Obtenemos un valor de umbral óptimo de 0.6941 (**Figura 3.9**).

```
umbralOpt=graythresh(Ient);
```

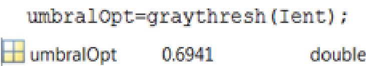


Figura 3.9 *Obtención del valor Umbral*

3.3 Filtrado de imagen

A partir de la imagen binarizada podemos seguir descomponiendo la imagen para obtener los contornos de las figuras que la conforman. En nuestro programa hemos

planteado 4 funciones distintas para que usuario elija la más adecuada dependiendo el tipo de imagen:

- Función *edge* (*BW* = *edge(I*)), a partir de la imagen previamente binarizada obtenemos una matriz de 1s donde el programa detecte contornos de figuras y 0s en el resto de posiciones.
  Este método se apoya en la función *bwboundaries (B* = *bwboundaries (BW*)), que nos devuelve un vector de celdas donde obtenemos las coordenadas de los pixeles que forman los contornos de los objetos de la imagen. Este comando nos va a permitir enviar la información al robot para trazar las figuras de la imagen de una manera ordenada. El resultado que obtenemos es el siguiente:

Figura 3.13. *Imagen sometida a filtro 'Edge'*

- Función *bwmorph*, que aplica operaciones morfológicas a la matriz de pixels binarizados para conseguir el contorno. Dentro de esta función usamos 2 opciones de filtros:

  1. Filtro '*Thicken*', *bwmorph(BW,'thicken',Inf)* . Este filtro espesa los objetos añadiendo pixeles al exterior hasta que los objetos desconectados se conecten a 8 pixeles. Se repite la operación *Inf* veces.

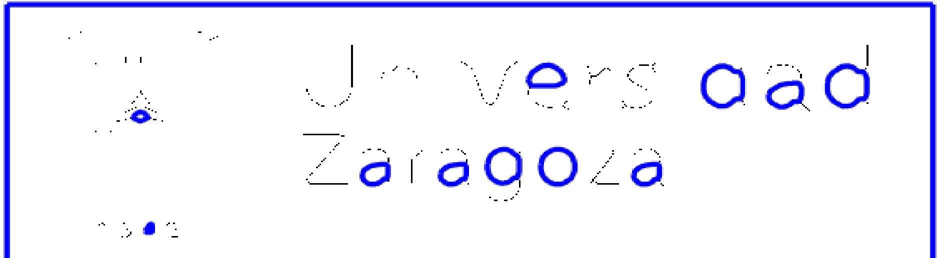


Figura 3.14. *Imagen sometida a filtro 'Thicken'*

  2. Filtro 'Skel', *bwmorph(~BW,'skel',Inf)*. Este filtro quita pixeles de los contornos sin permitir que se separen, se repite *Inf* veces.

Figura 3.15. *Imagen sometida a filtro 'Skeleton'*

- Función Pixels, se trata la imagen inversa a la original en blanco y negro con el filtro Skeleton y después busca los puntos de la matriz binarizada que sean 1 y para reordenarlos en una lista en la que se busca el pixel (n+1) más próximo al anterior (n), de tal forma que se van enviando al robot siguiendo la lista y por lo tanto en el orden de dibujo.

Figura 3.16. *Imagen sometida a método 'Pixels'*

Como podemos ver para la imagen a color el mejor resultado se obtiene con el filtro Edge, pero veamos otros ejemplos con imágenes en blanco y negro:

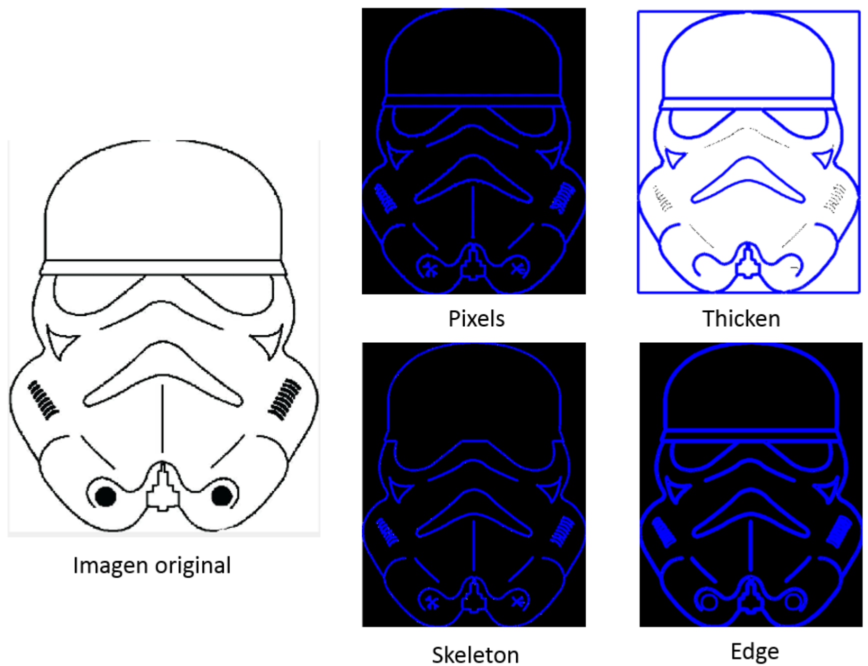


Figura 3.17. *Comparación de filtros para 'Stormtropper'*

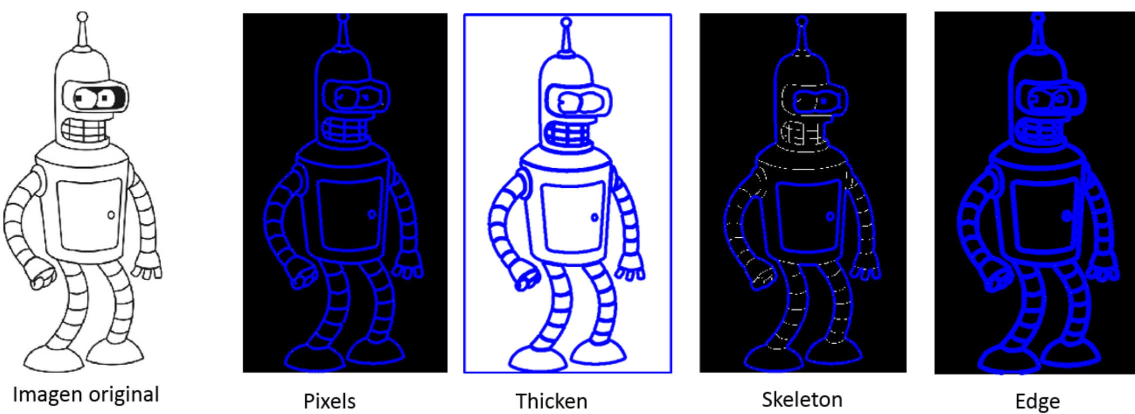


Figura 3.18. *Comparación de filtros para 'Bender'*

4. Protocolo de comunicación

Tras tener la imagen descompuesta, bien en una matriz binaria o en una lista de coordenadas de puntos, el siguiente paso es transmitir estos datos vía *socket* al programa RobotStudio.

4.1 Socket de comunicación. TCP/IP.

Cómo ya se ha nombrado anteriormente este proyecto se basará en la transmisión de datos establecida entre MATLAB y el Robot o el simulador RobotStudio.

Para establecer esta comunicación haremos uso de un socket de comunicación. Este concepto designa un modo por el cuál dos programas pueden intercambiar un flujo de datos, de manera fiable y ordenada. El socket se define mediante dirección IP, un protocolo de transporte y un número de puerto de conexión.

Existen multitud de protocolos de Internet (*ARP, HTTP, FTP…*), en nuestro caso utilizaremos dos de los protocolos más importantes y usados de esta familia, el protocolo TCP (protocolo de control de transmisión) y el protocolo IP (protocolo de internet). El esquema se muestra en la **figura 4.1.**

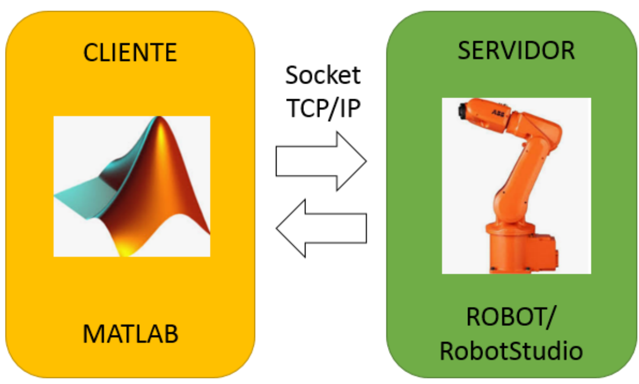


Figura 4.1. *Estructura Cliente/Servidor*

4.1.1 Protocolo TCP/IP

TCP (*Transmission Control Protocol*)/IP (*Internet Protocol*) son las siglas de un sistema (protocolo) que permite comunicar ordenadores que no se encuentran en la misma red.

El Protocolo de Control de Transmisión TCP permite a dos máquinas establecer una conexión para intercambiar información. Este protocolo se encarga de garantizar la entrega de datos, es decir, que lleguen correctamente y en el mismo orden que se enviaron del emisor al destinatario.

Por otra parte el Protocolo de Internet IP está compuesto por una serie de direcciones, que son una secuencia de números separados por puntos (cada número es un octeto

cuyo valor decimal está comprendido entre 0 y 255, por ejemplo 10.136.1.255) que identifican de manera lógica y ordenada un elemento de comunicación o conexión en red.

Este protocolo se utiliza si se requiere alta precisión en la transmisión de datos, ya que no permite que se pierda información por el camino. Lo usaremos para garantizar que la información que generamos desde MATLAB a la hora de descomponer la imagen se envía correctamente al robot y este la recibe, evitando la pérdida de trazos o la falta de definición de estos.

La comunicación está compuesta por un servidor y un cliente:

- *El cliente*, en nuestro caso programado en MATLAB, se encarga de procesar la información de la fotografía seleccionada y enviar los datos al servidor.
- *El servidor*, programado en RAPID. Nuestro Robot actúa como servidor y se ha programado para recibir los datos enviados por el cliente y realizar los movimientos adecuados para reproducir la fotografía.

4.2 Establecimiento de comunicación

Para poder comenzar a transmitir datos debemos habilitar una dirección IP y un puerto al que se conecten ambos sistemas. Esta dirección, la deberemos habilitar desde nuestro servidor, es decir RobotStudio o el Robot. Para Matlab es indistinto si actúa de servidor el software o el robot real, el comportamiento es el mismo ya que el código desarrollado es el mismo para ambos.

Primero establecemos las variables, tomando 2 del tipo *socketdev* (**Figura 4.2**) que nos indicarán si el afectado es el servidor o el cliente. Y además establecemos otras dos variables adicionales para indicar el número de la dirección IP y el puerto determinados (**Figura 4.3**).

```
VAR socketdev server;
VAR socketdev client;
```

Figura 4.2. *Declaración de Cliente y Servidor en RobotStudio*

```
VAR string ipadress:= "127.0.0.1";
VAR num port:=41595;
```

Figura 4.3. *Declaración de dirección IP y puerto*

Seguidamente tenemos el código que habilita la comunicación (**Figura 4.4**):

```
socketcreate server;
socketbind server,ipadress,port;
socketlisten server;
socketaccept server,client,\ClientAddress:=ipadress,\Time:=WAIT_MAX;
Socketsend  client,\str:="Ok";
```

Figura 4.4. *Establecimiento de comunicación en RobotStudio*

Tal y como se refleja en la **Figura 4.4**, primero usaremos el comando *socketcreate,* que nos permite crear el nuevo socket, para después gracias al comando *socketbind* enlazar la comunicación (indicándole el número de puerto y la dirección IP, sólo sirve cuando actúa de servidor).

El comando *socketlisten* permanece a la espera del cliente, actuando como servidor. Cuando el programa verifica que el cliente está conectado acepta la comunicación, gracias a *socketaccept* y para asegurarnos que se ha establecido correctamente el enlace enviamos un 'Ok' al cliente con *socketsend.*

Gracias a la *Toolbox* de control de instrumentos contenida en Matlab disponemos de varios comandos muy interesantes para trabajar con equipos externos, entre ellos el comando *tcpip.* Ahora veremos cuáles son las instrucciones que debemos procesar en Matlab para establecer el cliente (**Figura 4.5**):

```
%Comunicacion Socket con RobotStudio

t=tcpip(handles.direccionIP,handles.puerto); %establecemos datos de IP y puerto de conexión
handles.t= t;
fopen(t); % Abrimos comunicación TCP/IP
message= fread(t,2);% Leemos mensaje de RobotStudio
```

Figura 4.5. *Establecimiento de comunicación en Matlab*

El primer paso para configurar el cliente es crear un objeto *tcpip*, señalando la dirección IP y el puerto. Gracias a las variables *handles* podemos obtener y almacenar las variables obtenidas de la interfaz gráfica (GUIDE) que explicaremos más adelante. Gracias al comando *fopen,* establecemos el socket TCP/IP.

Cómo se explicaba en la parte de código para establecer la comunicación de RobotStudio, se enviará un Ok al cliente para verificar el enlace. Este mensaje se lee en Matlab con *fread.*

4.3 Transmisión de datos

Nuestro cliente Matlab debe transmitir los datos al servidor RobotStudio mediante un proceso de comunicación estructurado, que permita el dialogo entre ambos programas. Se han creado dos estructuras similares en ambos programas para que interactúen.

4.3.1 Emisión de datos desde Matlab

La transmisión de datos comenzara desde Matlab, mientras el servidor permanece a la espera de recepción. Dependiendo el filtro elegido para binarizar la imagen se han creado dos procedimientos distintos:

1. ‘*Dibuja*’, usado para los filtros *Edge, Skeleton y Thicken.* Estos filtros hacen uso del comando *bwboundaries* (**Figura 4.6**), que nos separa los distintos objetos en diferentes matrices dentro de una variable del tipo *cell.* Para indicar al robot cuando debe ‘saltar’ de un objeto a otro, y por lo tanto despegar el lápiz del papel, bastará con enviar una señal al robot y éste separará el lápiz del papel hasta el punto en el que comience a dibujar el siguiente objeto.

```
for k = 1:N

     if k==1 %Comprobamos para el primer punto que dibuje
    fwrite(t,'next'); %Enviamos señal 'next' al robot para que comience a dibujar el objeto
    Check(hObject); %Procedimiento de comprobación de recepción de datos
     end

   boundary = B{k}; %Creamos variable para una unica cell a partir de 'B' obtenida con bwboundaries
   [sz1,~]=size(boundary); %obtenemos tamaño para implementar el bucle for

    for i= 1:3:sz1
       if get(handles.stop,'UserData')
       fwrite(t,'stop');% stop condition
       break;
       end
   Line=[num2str(boundary(i,2)*factor1),',',num2str(boundary(i,1)*factor2)]; %Forma standard para enviar info del punto
   fwrite(t,Line); %Enviamos punto
   Check(hObject);
    end

  if k<N
   fwrite(t,'next'); %Si aun quedan objetos por dibujar enviaremos de nuevo el comando 'next' al robot
   Check(hObject);
  end
end
```

Figura 4.6. *Bucle de la función ‘Dibuja’*

Se comprueba mediante un condicionante *‘if’* si es el primer objeto que dibuja. Posteriormente se obtiene la dimensión de puntos del objeto para que vaya enviando puntos mediante un bucle *‘for’*.

Los puntos se envían en la forma standard *Line = [CoordenadaX, CoordenadaY].* La posición del robot frente a la hoja es la reflejada en la **Figura 4.7**:

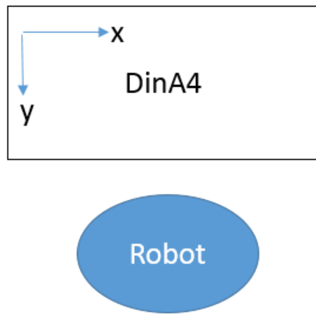


Figura 4.7. *Layout de la instalación*

Las dimensiones máximas son X=297mm y Y=210mm, las de un Din A4.

Por último si no es el último objeto sigue enviando en cada salto el comando 'next' al robot para que levante el bolígrafo y espere los siguientes puntos.

2. '*NextPixel*', usado para el método Pixel, de forma análoga al método anterior se irán enviando los puntos con la peculiaridad de que para este método los puntos a dibujar se han obtenido en el orden [x,y] (**Figura 4.8**).

```
[x,y]=find(BWP>0);
N=length(x);
puntos=[x,y, zeros(N,1)];
```

Figura 4.8. *Criterio de identificación de puntos*

Por lo que si se ha rotado la imagen al binarizarla para maximizarla en el DinA4 se debe rotar, esto se comprueba al principio:

```
if Rot==0
 j1=2;
 factorj1=factor2;
 j2=1;
 factorj2=factor1;
else
 j1=1;
 factorj1=factor1;
 j2=2;
 factorj2=factor2;
end
```

Figura 4.9. *Criterio de rotación de imagen*

Y el comando que se envía al robot es de la forma:

```
Line=[num2str(puntos(ii,j1)*factorj1),',',num2str(puntos(ii,j2)*factorj2)];
```

Figura 4.10. *Forma de envío de los puntos de la línea*

Por otra parte, para comprobar cuando levantar el lápiz del papel entre objetos se calcula la distancia entre el pixel n y el pixel n-1 de la lista (en nuestro caso n-2 ya que partimos del pixel nº 3 y aumentamos de 2 en 2):

```
for ii=3:2:N


    a=puntos(ii,1)-puntos(ii-2,1);
    b=puntos(ii,2)-puntos(ii-2,2);
    distancia=sqrt(a*a+b*b);
```

Figura 4.11. *Calculo de la distancia entre puntos*

El criterio es el siguiente, si hay una distancia >5 pixels entre ambos se detectan objetos distintos y se envía la orden '*next*' (análoga al método *boundary*) para que el robot levante el boli y dibuje el siguiente trazo:

```
if distancia>5

 fwrite(t,'next');
 Check(hObject);
 Line=[num2str(puntos(ii,j1)*factorj1),',',num2str(puntos(ii,j2)*factorj2)];
 fwrite(t,Line);
 Check(hObject);

else

Line=[num2str(puntos(ii,j1)*factorj1),',',num2str(puntos(ii,j2)*factorj2)];
fwrite(t,Line);
Check(hObject);
end
```

Figura 4.12. *Bucle de la función 'NextPixel'*

4.3.2 Recepción de datos en RobotStudio

Si el establecimiento de comunicación ha sido correcto se inicia el procedimiento '*Communication*' en RobotStudio, mostrado en la **Figura 4.13**:

```
PROC Communication()

   message1:="Inicio";

     while message1<>"stop" do


     SocketReceive client, \str:=message1 \Time:=40;

     draw:=1;
```

Figura 4.13. *Comienzo del procedimiento de recepción de datos*

Al comenzar se inicializa la variable '*draw*' con valor 1, para comprobar si debe entrar a dibujar u otras funcionalidades que explicaremos más adelante. Se ejecutará el bucle '*while*' hasta que el robot reciba el mensaje '*stop*'. El robot puede recibir varios tipos de mensajes, lo que comprobaremos mediante '*ifs*' anidados:

- '*calibVEL*', procedimiento integrado en la GUIDE para calibrar las velocidades de dibujo y de salto entre objetos. Se explicará más adelante.
- '*Zcalib*', procedimiento para calibrar la altura Z de dibujo.
- '*calib*' procedimiento para calibrar posición.
- '*next*', ejecutará salto entre objetos para que el robot levante el bolígrafo.
- 'fin', el robot finalizará la rutina y volverá al punto de reposo.

- Recibe una posición [x,y]. En este caso no habrá entrado en los anteriores ‘*ifs*’ y la variable *draw* seguirá valiendo 1.Lo explicamos a continuación:
    - Si Matlab envía un punto buscaremos en el string recibido (*message1*) la ‘,’ que divide la posición x (*Pos1*) y la y (*Pos2*) para obtener ambos strings(**Figura 4.14**). Transformaremos estos ‘*strings’* a valor numérico mediante el comando *StrToVal* y mediante ‘*MoveL*’ iremos al punto recibido [*Xp, Yp, Zbase*] (*Zbase* es la altura de dibujo en el eje Z).

```
IF draw=1 THEN

Pos1:=StrFind(message1,1,",");
Pos2:=StrLen(message1);
Ok:= StrToVal (StrPart(message1,1,Pos1-1),Xp);
Ok:= StrToVal (StrPart(message1,Pos1+1,Pos2-Pos1),Yp);
MoveL [[Xp,Yp,Zbase],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vdib,Z10,tool0\WObj:=Workobject_1;
Socketsend  client,\str:="Ok";
SendPosition;


ENDIF
```

Figura 4.14. *Dibujado de puntos en RobotStudio*

Cuando el robot haya alcanzado el punto, se enviará ‘*Ok*’ a Matlab para que envíe el siguiente punto. La comprobación entre ambos programas la explicaremos en el **apartado 4.3.3.**

En el caso que haya un salto entre objetos RobotStudio recibirá el mensaje ‘*next*’:

```
IF message1= "next" THEN

Movej [[Xp,Yp,Zbase-40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vpos,Z5,tool0\WObj:=Workobject_1;

Socketsend  client,\str:="Ok";
SendPosition;

SocketReceive client, \str:=message1 \Time:=20;

Pos1:=StrFind(message1,1,",");
Pos2:=StrLen(message1);
Ok:= StrToVal (StrPart(message1,1,Pos1-1),Xp);
Ok:= StrToVal (StrPart(message1,Pos1+1,Pos2-Pos1),Yp);

Movej [[Xp,Yp,Zbase-40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vpos,Z5,tool0\WObj:=Workobject_1;
MoveL [[Xp,Yp,Zbase-5],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vpos,Z5,tool0\WObj:=Workobject_1;
MoveL [[Xp,Yp,Zbase],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vdib,fine,tool0\WObj:=Workobject_1;
Socketsend  client,\str:="Ok";
SendPosition;
draw:=0;
```

Figura 4.15. *Función para saltos entre objetos del dibujo*

Cuando se reciba el mensaje, el Robot levantará el bolígrafo automáticamente 40mm en vertical sobre el último punto recibido *Movej [X, Y, Zbase-40]*. Tras esto enviará el OK a Matlab para que le envíe el primer punto del siguiente objeto.

Cuando reciba el punto encadenará 3 movimientos. El primero a una altura de 40 mm sobre el papel para posicionarse en la vertical z del punto, luego comenzará la aproximación hasta los 5mm, donde reducirá la velocidad. Así conseguimos evitar choques al entrar en contacto con el papel y una aproximación más suave sin perder velocidad de dibujo.

Por último se comprobará la posición y se enviará a Matlab. Se cambia la variable '*Draw*' a 0 para que no entre en el siguiente '*if* 'de dibujo y vuelva a entrar en el bucle.

4.3.3 Comprobación de posición. Interacción entre funciones 'Check'(Matlab) y 'SendPosition' (RobotStudio).

Primero vamos a echarles un vistazo a los procedimientos que interaccionan:

- Procedimiento '*SendPosition*' (RobotStudio), tal y como vemos en la **Figura 4.16**:

```
PROC SendPosition()

    Position := CPos(\Tool:=tool0 \WObj:=WObj0); !Obtenemos posición de la herramienta del robot respecto a centro de coordenadas
    Position.x:=round(Position.x\Dec:=0); !Redondeamos a un decimal la coordenada x
    Position.y:=round(Position.y\Dec:=0); !Redondeamos a un decimal la coordenada y
    Position.z:=round(Position.z\Dec:=0); !Redondeamos a un decimal la coordenada z
    posstring := ValToStr(Position);  !Cambiamos posición a string
    poslen:= StrLen(posstring); !obtenemos la longitud del string
    poslen2:= ValToStr(poslen); !Pasamos la longitud a string
    Socketsend  client,\str:=poslen2; !Enviamos el string de longitud
    Socketsend  client,\str:=posstring; !Enviamos las coordenadas de posición
```

Figura 4.16. *Procedimiento 'SendPosition'*

- Procedimiento '*Check*' (Matlab), mostrado en la **Figura 4.17:**

```
message=fread(t,2); %Leemos el mensaje de 2 digitos maximo (Ok)
waitfor(message,'Ok'); %Esperamos el Ok
poslen = char(fread(t,[1,2])); %Leemos la longitud del mensaje de la posición
poslen = str2double(poslen);   %Lo transformamos a Double
message = char(fread(t,[1,poslen])); %Leemos la posición de longitud poslen
handles.pos(z,:) = str2double(message); %Almacenamos posición recibida
z=z+1;
guidata(hObject,handles); %Actualizamos valores para guardar variable pos antes de drawnow
set(handles.coordenadas,'string',message); %Mostramos coordenadas por pantalla
drawnow; %Actualizamos dibujos de la interfaz para mostrar coordenadas
```

Figura 4.17. *Procedimiento 'Check'*

Como podemos observar ambos siguen una secuencia similar, mantienen una conversación. El procedimiento '*Check*' comienza esperando el '*Ok*' del Robot, tras lo cual se produce la interacción:

- RobotStudio: Obtiene la posición de la herramienta (*tool0*) respecto al centro de coordenadas de la base (*WObj0*) con el comando '*Cpos'.* Estas posiciones las redondeamos para sacar la posición sin decimales y se transforman a *String* para

enviarlas. El código continúa con la medición de la longitud del *String* (*StrLen*) y el cambio de esta cifra también a S*tring*. Primero se envía la longitud del *String* y después la posición.

- Matlab: Lee mediante *fread* la longitud de los caracteres que espera recibir y mediante '*str2double'* la transforma en variable numérica tipo *double.* Tras esto lee la posición y espera el tiempo justo para recibir la longitud de caracteres antes recibida (*poslen*). Este paso es fundamental para acelerar la comunicación, dado que si no le indicáramos la longitud de caracteres que debe esperar recibir por defecto esperaría recibir una secuencia de longitud máxima 24 caracteres, lo cual ralentiza la comunicación. La posición que recibimos la almacenamos en la variable *pos* (para mostrar en una gráfica las posiciones recorridas al terminar el dibujo) y la mostraremos en la interfaz gráfica con los comandos *set+drawnow.*

5. Interfaz gráfica de control en Matlab. GUI.

Para facilitar el control del robot desde el cliente, Matlab, se ha diseñado y programado una interfaz gráfica *GUI* (por sus siglas en inglés, Graphical User Interface). En este 'cuadro de mandos' se ha dado cabida a todas las funcionalidades programadas mediante la implementación de diversos botones y cuadros (tanto de texto como de imagen).

A continuación mostramos la apariencia de la interfaz (**Figura 5.1**):

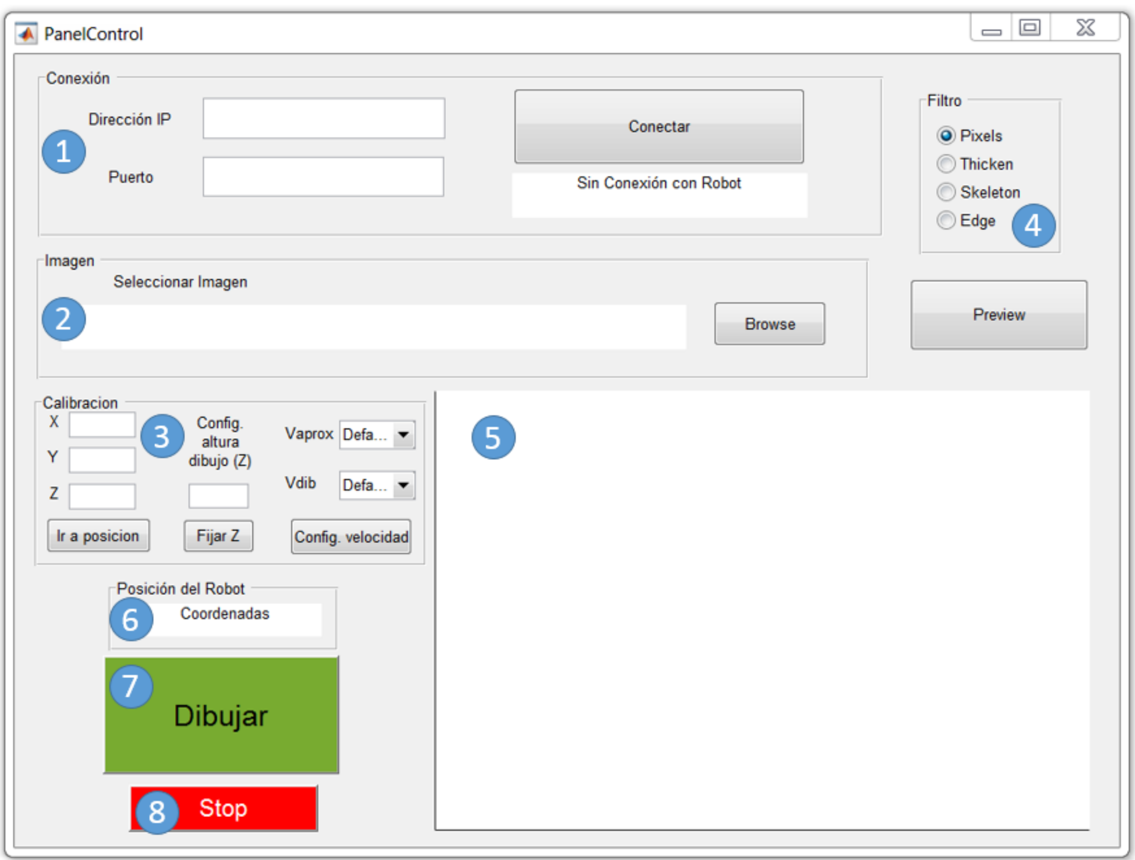


Figura 5.1. *Interfaz Panel de Control*

Para el diseño de esta interfaz se ha buscado la sencillez y claridad de los diversos campos.

En los próximos apartados, vamos a ir desarrollando 1 por 1 las diferentes funciones indicadas con números del 1 al 8 en el dibujo.

5.1 Código de inicialización de la GUI.

La interfaz es programada cómo un conjunto de funciones. Esta función comienza con un código estandarizado que ejecuta el arranque de la aplicación diseñada y que no debe ser editado (**Figura 5.2**).

```
function varargout = PanelControl(varargin)

% Código de inicialización. No editar.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PanelControl_OpeningFcn, ...
                   'gui_OutputFcn',  @PanelControl_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
```

Figura 5.2. *Código de inicialización*

GUI utiliza la función *guidata* para almacenar una estructura denominada *handles*, que contiene todos los componentes de la interfaz de usuario. MATLAB pasa la matriz *handles* a cada función de devolución de llamada, así podemos compartir variables entre funciones.

Tras la primera función de inicialización, se sitúan la función de apertura (*OpeningFcn*) y la de salida (*OutputFcn*), ambas también estandarizadas y obligatorias, aunque estas sí que son editables (**Figura 5.3**).

En nuestro caso para la función de apertura hemos introducido 3 líneas de código. Primero, mediante la función *'guidata'* actualizamos la estructura de datos *handles*.

Las siguiente línea nos permite la apariencia inicial de la interfaz, ya que del cuadro nº 5 de la interfaz, que es un cuadrante para mostrar gráficos (*axes*) eliminamos la numeración de los ejes *x* e *y* por motivo estético. Este cuadro lo usaremos posteriormente como pantalla para mostrar las imágenes a dibujar y podemos prescindir de los ejes.

Y en la 3ª línea inicializamos la variable *stop,* la cual nos servirá para activar la parada de emergencia programada en el botón indicado con el nº8 que explicaremos más adelante.

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

% Choose default command line output for PanelControl
handles.output = hObject;
% Update handles structure

guidata(hObject, handles);
set(handles.axes1,'xtick',[], 'ytick',[]);
set(handles.stop,'UserData',0);

% UIWAIT makes PanelControl wait for user response (see UIRESUME)
% uiwait(handles.figure1);
 % --- Outputs from this function are returned to the command line.
function varargout = PanelControl_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

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

Figura 5.3. *Funciones OpeningFcn y OutputFcn*

5.2 Conexión

Respecto a los diferentes cuadros de la interfaz, comenzamos por el cuadro indicado por el número 1, se muestra en la **Figura 5.4**. En este apartado vemos 3 cuadros de texto y un botón. Se ha nombrado como 'Conexión' ya que mediante este apartado conseguiremos establecer el socket TCP-IP con RobotStudio.

Desde aquí debemos indicar la dirección IP y el puerto de la red en el que está ubicado el Robot IRB 120 (o su versión simulada en RobotStudio) y pulsar el botón conectar. En el cuadro inferior al botón se nos mostrará el estado del robot.

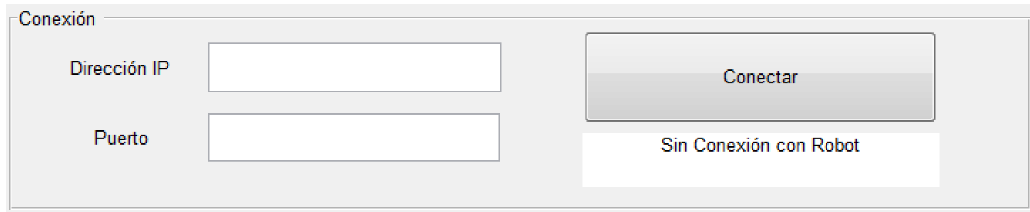


Figura 5.4. *Cuadro de conexión*

Esta función está compuesta por 2 cuadros de texto editables, de estructura similar. En la función *dirIP_Callback* se obtiene la dirección IP introducida por usuario, mediante el comando *get,* y la almacenamos en la estructura *handles*. La función *dirIP_CreateFcn* sirve para inicializar el cuadro de texto con las propiedades configuradas por usuario en la interfaz (**Figura 5.5**).

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

% Función que nos permite obtener la IP que introduce el usuario

direccionIP = get(handles.dirIP,'string'); %Lee y almacena la dirección IP
handles.direccionIP = direccionIP; %Guardamos en estructura handles
guidata(hObject, handles); %actualizamos guidata

% --- Executes during object creation, after setting all properties.

function dirIP_CreateFcn(hObject, eventdata, handles)

% Muestra la celda para recoger la IP con fondo blanco

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

Figura 5.5. *Funciones para obtener la direccion IP*

La estructura para obtener el nº de puerto es similar a la anterior, ya que ambos son cuadros de texto editables, compuestos por funciones *Callback+CreateFcn*. La única diferencia es que para el puerto se transforma el *String* obtenido en variable numérica (*str2double*), lo cual nos permitirá un mejor tratamiento (**Figura 5.6**). Esto no lo podíamos hacer con la dirección ya que esta es de la forma (*xxx.xxx.xxx.xxx*).

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

% Obtenemos el puerto indicado por el usuario

puerto = get(handles.Port,'string');
puerto=str2num(puerto);
handles.puerto = puerto;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function Port_CreateFcn(hObject, eventdata, handles)

% Muestra la celda para recoger el n° de puerto con fondo blanco

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

Figura 5.6. *Funciones para obtener el Puerto de conexión*

Para conectar el robot sólo nos falta configurar el botón que ejecute los comandos necesarios para establecer el socket. Este botón lleva asociada la función *Conectar_Callback*, mostrada en la **Figura 5.7**:

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

%Establece conexión con RobotStudio/Robot

Conexion;
set(handles.statusrobot,'string','Ok');
guidata(hObject, handles);
```

Figura 5.7. *Función para establecer conexión*

Esta función, a su vez, hace referencia a la función Conexión, programada fuera de la estructura principal de la GUI, que hace uso de las funciones para establecer la comunicación:

```
handles=guidata(hObject);


t=tcpip(handles.direccionIP,handles.puerto); %establecemos datos de IP y puerto de conexión
handles.t= t;
fopen(t); % Abrimos comunicación TCP/IP
message= fread(t,2);% Leemos mensaje de RobotStudio
waitfor(message,'Ok'); %Esperamos el Ok

guidata(hObject, handles);
```

Figura 5.8. *Procedimiento de conexión*

5.3 Selección de imagen

En este apartado se seleccionará la imagen que se vaya a dibujar (**Figura 5.9**). Esta imagen deberá encontrarse dentro de la memoria del ordenador o situada en alguna ruta accesible por el explorador de Windows.

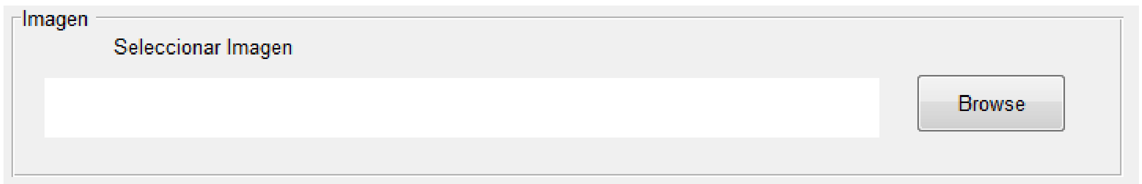


Figura 5.9. *Cuadro de selección de imagen*

Como se puede observar está compuesta por un cuadro de texto fijo y un botón. Este botón '*Browse*' está programado de tal forma que al pulsarlo se abre el explorador de Windows y nos permite ir buscando por las diversas carpetas y rutas la imagen (**Figura 5.10**).

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

%Función que nos permite seleccionar una imagen de nuestro ordenador para dibujarla

cla reset;
[filename,pathname]=uigetfile({'.jpg'},'File selector');  %Consigue el nombre del archivo y la ruta
handles.filename=filename;  %Guardamos el nombre del archivo en la estructura handles
fullpath=strcat(pathname,filename); %Concatenamos la ruta y el nombre del archivo
set(handles.text2,'string',fullpath);  %Lo mostramos en el cuadro de texto
axes(handles.axes1);
A=imread(filename,'jpg'); %Leemos la imagen, formato JPG
imshow(A); %Mostramos la imagen en el panel de control
set(handles.axes1,'xtick',[], 'ytick',[]);  %Mantenemos los ejes sin numeración
handles.output = hObject;
guidata(hObject, handles);   %Actualizamos la estructura handles
```

Figura 5.10. *Función para seleccionar imagen*

Matlab nos permite seleccionar una imagen mediante el comando *uigetfile*, en el que debemos indicar la extensión que queremos (en nuestro caso *.jpg*). Las siguientes líneas de código se utilizan para mostrar en la ventana de texto la ruta completa de la imagen que estamos tratando y mostrarla mediante el comando *imshow* en la ventana nº5. Ejemplo (**Figura 5.11**), con la imagen '*mandala1*', situada en la ruta que se indica en la imagen:

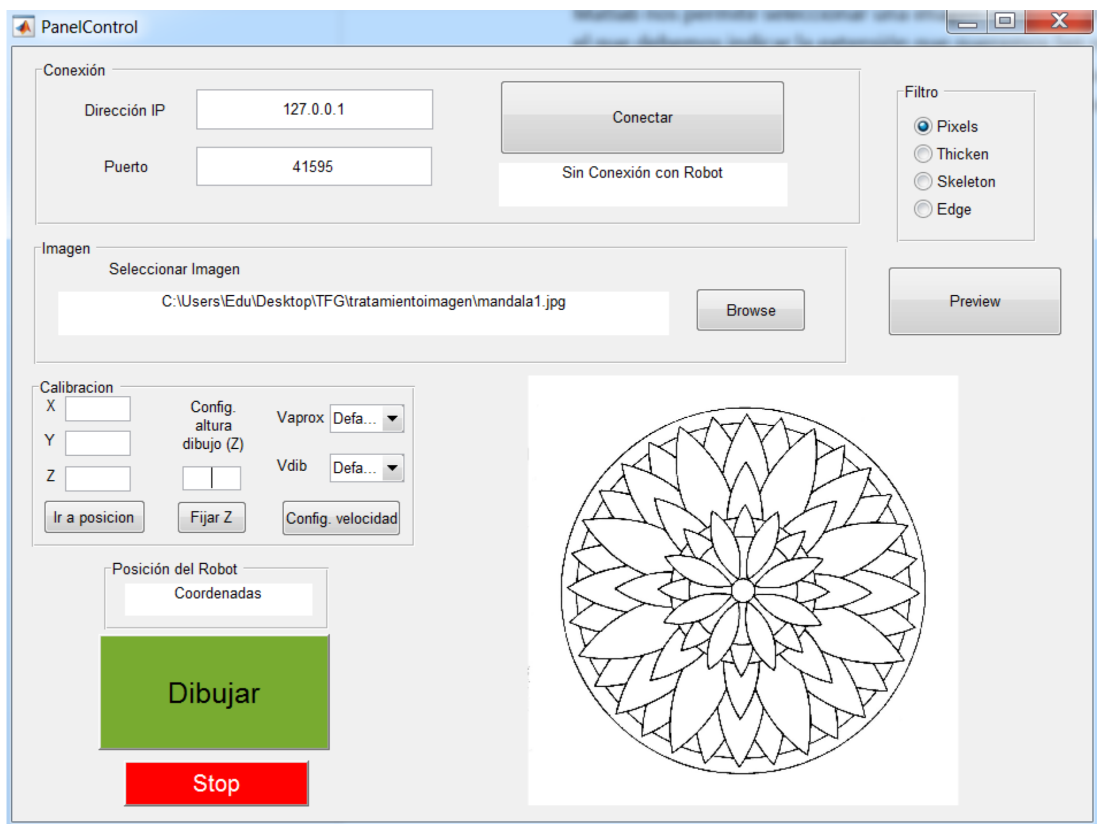


Figura 5.11. *Ejemplo de imagen seleccionada en Panel de Control*

5.4 Calibración

En este apartado se detallan 3 funciones (**Figura 5.12**):

- Mover robot a posición X,Y,Z
- Configurar altura de dibujo Z
- Configurar velocidades de dibujo y aproximación

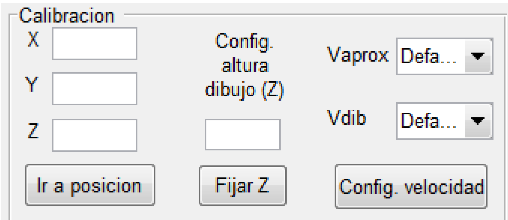


Figura 5.12. *Cuadro de calibración*

Con estas 3 funciones podemos configurar el robot para los ensayos en laboratorio. Vamos a detallarlas una por una.

5.4.1 Mover robot a posición X,Y,Z

Gracias a esta función se va a poder mover el robot a la posición que deseemos. Este apartado está compuesto por 3 cuadros de texto editables uno para cada valor de las coordenadas (X, Y, Z), ambos de idéntica estructura, formados por una función '*Callback*' más una función '*CreateFcn*' :

```
function xcalib_Callback(hObject, eventdata, handles)
%xcalib = get(handles.xcalib,'string');
%handles.xcalib = xcalib;
%guidata(hObject, handles);

% Hints: get(hObject,'String') returns contents of xcalib as text
%        str2double(get(hObject,'String')) returns contents of xcalib as a double


% --- Executes during object creation, after setting all properties.

function xcalib_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xcalib (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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

Figura 5.13. *Ejemplo para obtener coordenada X*

Se puede observar que no se introduce ninguna particularidad en estas dos funciones, el código es el estándar que se genera automáticamente al crearlas con la GUI.

Y por otra parte programamos el botón '*irposición*' (**Figura 5.14**).

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

handles=guidata(hObject);
t = handles.t;

xcalib = char(get(handles.xcalib,'string'));
ycalib = char(get(handles.ycalib,'string'));
zcalib = char(get(handles.zcalib,'string'));


fwrite(t,'calib');
   message=fread(t,2);
   waitfor(message,'Ok');
   Line=[xcalib,',',ycalib,',',zcalib];
   fwrite(t,Line);
   poslen= char(fread(t,[1,2]));
   poslen = str2double(poslen);
   message= char(fread(t,[1,poslen]));
   set(handles.coordenadas,'string',message);
   handles.message;
   drawnow;
   guidata(hObject, handles);
```

Figura 5.14. *Función 'IrPosición'*

Este botón comienza obteniendo los valores introducidos por usuario de *xcalib, ycalib y zcalib* de los cuadros de texto, tras esto envía el mensaje '*calib*' a RobotStudio y cuando recibe el Ok de vuelta envía la posición. El código análogo con el que conseguimos que el Robot alcance la posición deseada es el siguiente, en RobotStudio:

```
IF message1="calib" THEN

Socketsend  client,\str:="Ok";
SocketReceive client, \str:=message1 \Time:=20;

Pos1:=StrFind(message1,1,",");
Pos2:=StrFind(message1,Pos1+1,",");
Pos3:=StrLen(message1);
Ok:= StrToVal (StrPart(message1,1,Pos1-1),Xp);
Ok:= StrToVal (StrPart(message1,Pos1+1,Pos2-Pos1),Yp);
Ok:= StrToVal (StrPart(message1,Pos2+1,Pos3-Pos2),Zp);

Movej [[Xp,Yp,Zp],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vpos,Z5,tool0\WObj:=Workobject_1;

SendPosition;

draw:=0;


ELSE
```

Figura 5.15. *Procedimiento para ir a punto en RobotStudio*

Se puede observar que se lee el mensaje y si es '*calib*' entra en el '*if*' correspondinte. Dentro de este comando el mensaje se divide en coordenadas *Xp,Yp y Zp* y mediante *Movej* se mueve el cabezal del robot al punto recibido.

5.4.2 Configurar altura de dibujo Z

Con este botón se variará la altura de dibujo predeterminada en la simulación, en la que se modela una mesa situada a 250 mm de la base del robot. Veamos las coordenadas de la referencia situada en la esquina superior izquierda de la mesa en la **Figura 5.16**:

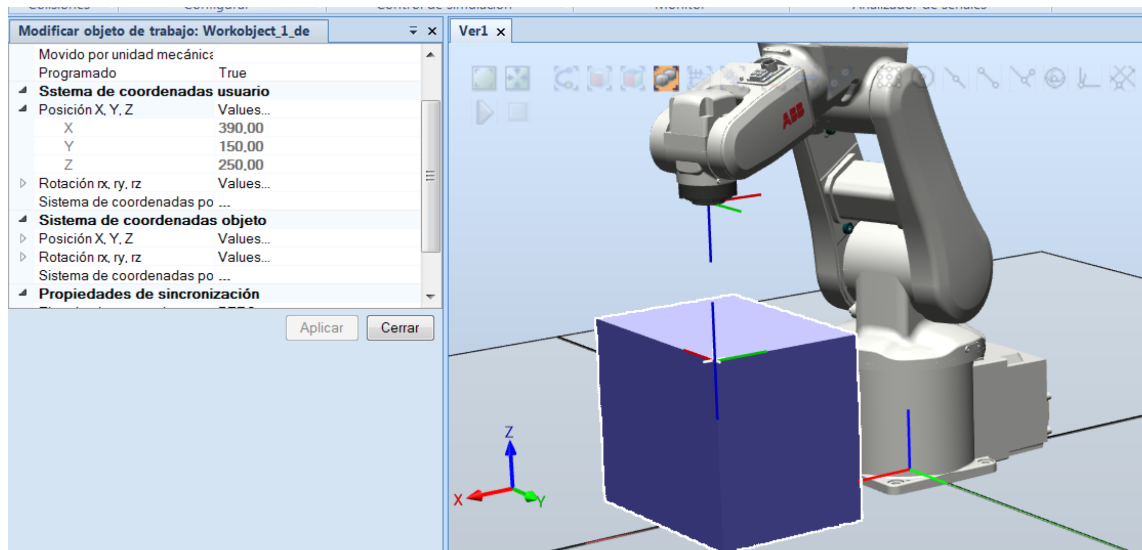


Figura 5.16. *Instalación reproducida en RobotStudio*

En laboratorio se debe medir la diferencia de altura del montaje real frente a este parámetro prefijado. La diferencia es la que deberá indicar en el cuadro de texto para que el robot modifique la altura de las coordenadas.

El código será similar al explicado en el apartado anterior, un cuadro de texto que al establecerlo en la interfaz genera un código estándar cómo el de la **figura 5.17** (ver apartado anterior) y un botón, que denominamos '*Zcero*':

```
% --- Executes on button press in Zcero.
function Zcero_Callback(hObject, eventdata, handles)
% hObject    handle to Zcero (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles=guidata(hObject);
t = handles.t;

Zbase = char(get(handles.Zbase,'string'));


fwrite(t,'Zcalib');
   message=fread(t,2);
   waitfor(message,'Ok');
   fwrite(t,Zbase);
   poslen = fread(t,[1,2]);
   poslen= char(poslen);
   poslen = str2num(poslen);
   message = fread(t,[1,poslen]);
   message= char(message);
   set(handles.coordenadas,'string',message);
   handles.message;
   drawnow;
   guidata(hObject, handles);
```

Figura 5.17. *Función para configurar altura de dibujo Z*

La parte interesante de esta función se encuentra en RobotStudio, observemos el código análogo en la **Figura 5.18**:

```
IF message1="Zcalib" THEN

Socketsend  client,\str:="Ok";
SocketReceive client, \str:=message1 \Time:=20;

Ok:= StrToVal (message1,Zbase);


Movej [[0,0,Zbase],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vpos,Z5,tool0\WObj:=Workobject_1;
Movej [[0,0,Zbase-40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],Vdib,Z5,tool0\WObj:=Workobject_1;

SendPosition;

draw:=0;
```

Figura 5.18. *Procedimiento para calibrar altura de dibujo Z en RobotStudio*

Como se puede observar al recibir el mensaje '*message1*', guardamos ese valor en la variable *Zbase* y hacemos un punto en el papel para verificar la altura, programado mediante dos comandos *Movej,* uno de subida y otro de bajada.

5.4.3 Configurar velocidades de dibujo y aproximación

Por último dentro de este cuadro de comandos se verá como calibrar las 2 velocidades a las que se desplazará el robot, la velocidad de dibujo *Vdib* y la velocidad de transición entre objetos *Vaprox*.

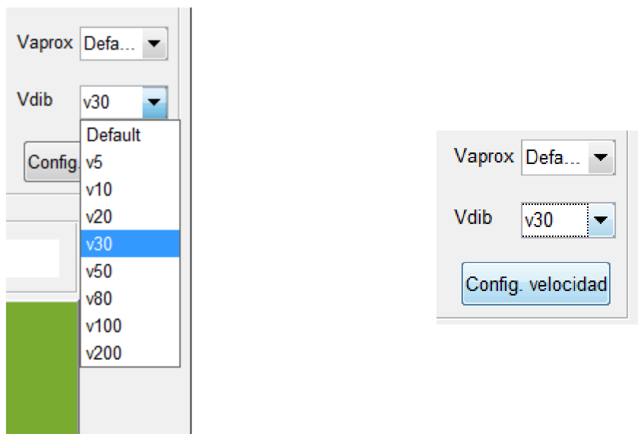


Figura 5.19. *Selección de velocidades*

Como observamos en la **figura 5.19**, al seleccionar las velocidades obtenemos un desplegable con las velocidades posibles preseleccionadas que nos permite elegir velocidades desde 5mm/s a 200mm/s. Este desplegable se programa de la siguiente forma, como ejemplo para *Vdib* (*Vaprox* se programa igual, **Figura 5.20**):

```
function Vdib_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
content=cellstr(get(hObject,'String'));
pop_choice=content(get(hObject,'Value'));
if (strcmp(pop_choice,'Default'))
    handles.Vdib='100';
elseif (strcmp(pop_choice,'v5'))
    handles.Vdib='5';
elseif (strcmp(pop_choice,'v10'))
    handles.Vdib='10';
elseif (strcmp(pop_choice,'v20'))
    handles.Vdib='20';
elseif (strcmp(pop_choice,'v30'))
    handles.Vdib='30';
elseif (strcmp(pop_choice,'v50'))
    handles.Vdib='50';
elseif (strcmp(pop_choice,'v80'))
    handles.Vdib='80';
elseif (strcmp(pop_choice,'v100'))
    handles.Vdib='100';
elseif (strcmp(pop_choice,'v200'))
    handles.Vdib='200';
end
guidata(hObject, handles);
```

Figura 5.20. *Función para selección de velocidad de dibujo*

La función está compuesta por una serie de comandos *‘if’* anidados, dependiendo del valor seleccionado en el desplegable *‘pop_choice’* que averiguaremos mediante el comando *‘strcmp’* se obtiene un valor de velocidad *Vdib* u otro.

Cuando se haya realizado la selección se enviarán ambos valores al robot mediante el botón *Velbutton* (**Figura 5.21**):

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

handles=guidata(hObject);
t = handles.t;

Vpos = handles.Vpos; %Tomamos el valor almacenado en handles, obtenido del desplegable
Vdib = handles.Vdib;

fwrite(t,'calibVEL'); %Enviamos el mensaje de calibración de velocidad
message=fread(t,2);   %Recibimos el OK

vel=[Vpos,',',Vdib]; %Enviamos ambas velocidades
fwrite(t,vel);
guidata(hObject, handles);
```

Figura 5.21. *Función para enviar velocidades al Robot*

En RobotStudio el procedimiento que se activará será el siguiente:

```
 IF message1="calibVEL" THEN

Socketsend  client,\str:="Ok";
SocketReceive client, \str:=message1 \Time:=20;

Pos1:=StrFind(message1,1,",");
Pos2:=StrLen(message1);
Ok:= StrtoVal(StrPart(message1,1,Pos1-1),Vpos1);
Ok:= StrtoVal(StrPart(message1,Pos1+1,Pos2-Pos1),Vdib1);
VelRobot;
draw:=0;

 ELSE
```

Figura 5.22. *Procedimiento de calibración de velocidad*

Como podemos observar el algoritmo es similar a los anteriores, con la particularidad que incluimos la función 'VelRobot' que nos permite traducir el mensaje recibido tipo *'string'* (variables Vpos1 y Vdib1) a dos variables del tipo '*speeddata*' (Vpos y Vdib), mostrado en la **Figura 5.23**. Las variables tipo *speeddata* para que sean comprensibles por el robot están compuestas por 4 números:

- *v_tcp*, la velocidad del punto central de la herramienta (TCP) en mm/s.
- *v_ori*, la velocidad de reorientación alrededor del TCP, expresada en grados/s. (por defecto 500º/s)
- *v_leax*, la velocidad de los ejes externos lineales, en mm/s. (por defecto 5000mm/s)
- *v_reax*, la velocidad de los ejes externos de rotación, en grados/s. (por defecto 1000º/s)

```
PROC VelRobot ()
    IF Vpos1=5 THEN
         Vpos:=v5;
    ELSEIF Vpos1=10 THEN
         Vpos:=v10;
    ELSEIF Vpos1=20 THEN
         Vpos:=v20;
    ELSEIF Vpos1=30 THEN
         Vpos:=v30;
    ELSEIF Vpos1=50 THEN
         Vpos:=v50;
    ELSEIF Vpos1=80 THEN
         Vpos:=v80;
    ELSEIF Vpos1=100 THEN
         Vpos:=v100;
    ELSEIF Vpos1=200 THEN
         Vpos:=v200;
    ENDIF

    IF Vdib1=5 THEN
         Vdib:=v5;
    ELSEIF Vdib1=10 THEN
         Vdib:=v10;
    ELSEIF Vdib1=20 THEN
         Vdib:=v20;
    ELSEIF Vdib1=30 THEN
         Vdib:=v30;
    ELSEIF Vdib1=50 THEN
         Vdib:=v50;
    ELSEIF Vdib1=80 THEN
         Vdib:=v80;
    ELSEIF Vdib1=100 THEN
         Vdib:=v100;
    ELSEIF Vdib1=200 THEN
         Vdib:=v200;
    ENDIF
```

Figura 5.23. *Procedimiento para transformar velocidad al tipo 'Speeddata'*

5.5 Selección de filtro y previsualización de imagen

Anteriormente planteábamos el problema de que no todos los filtros funcionan bien para todas las imágenes. El usuario debe decidir que filtro se ajusta mejor a la imagen que quiere reproducir en papel (**Figura 5.24**), por ello se ha programado una función de selección y previsualización de imagen.

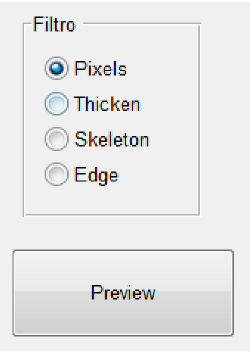


Figura 5.24. *Cuadro de selección de filtro*

Veamos cómo funciona en la **Figura 5.25**:

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

h=get(handles.Filtro,'SelectedObject');
type=get(h,'Tag'); %Comprobamos con que filtro trazar
set(handles.stop,'UserData',0);

switch type    %Ejecutamos la función correspondiente al filtro seleccionado

    case 'Pixels'
        Pixels(hObject)

    case 'Thicken'
        Thicken(hObject)

    case 'Skeleton'
        Skeleton(hObject)

    case 'Edge'
        Edge(hObject)

end

guidata(hObject, handles);
```

Figura 5.25. *Función para selecci*

La función comienza con un comando *get* que nos permite obtener el valor del filtro seleccionado, almacenado en la variable *handles.Filtro.* Con este valor mediante una estructura *switch…case* ejecutamos el código correspondiente al filtro asociado.

Dentro de cada función se incluye el comando *imshow* que nos mostrará en el cuadro nº 5 el resultado del filtro.

5.6 Visualización de imagen

Gracias a los ejes denominados ‘axes1’ podremos observar en nuestra interfaz la imagen a dibujar (**Figura 5.26**). Mediante los comandos *imshow o plot,* por defecto se mostrará en la ventana la imagen original (al seleccionarla de su carpeta) o filtrada (**Figura 5.27**, al previsualizar y mientras se dibuja):

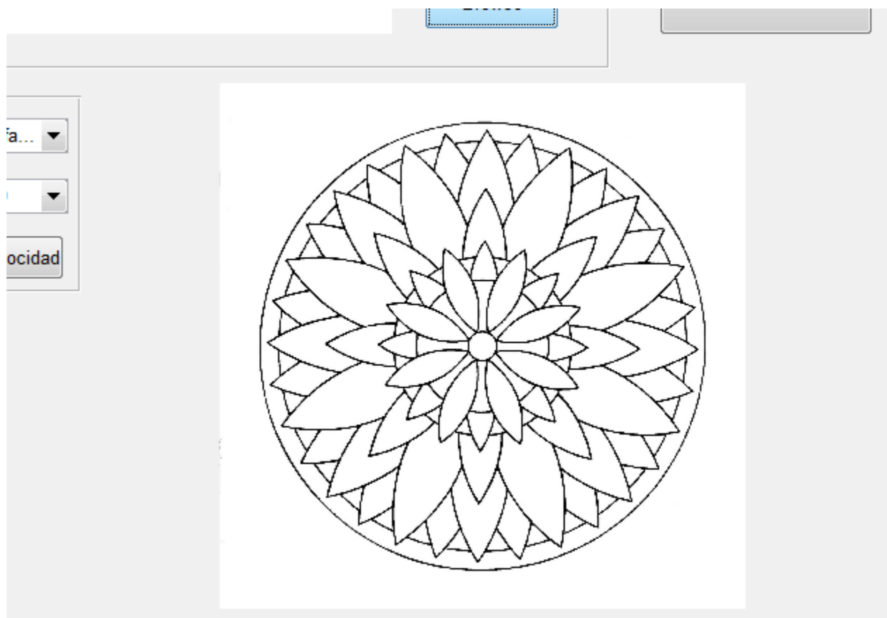


Figura 5.26. *Imagen mandala1 original*

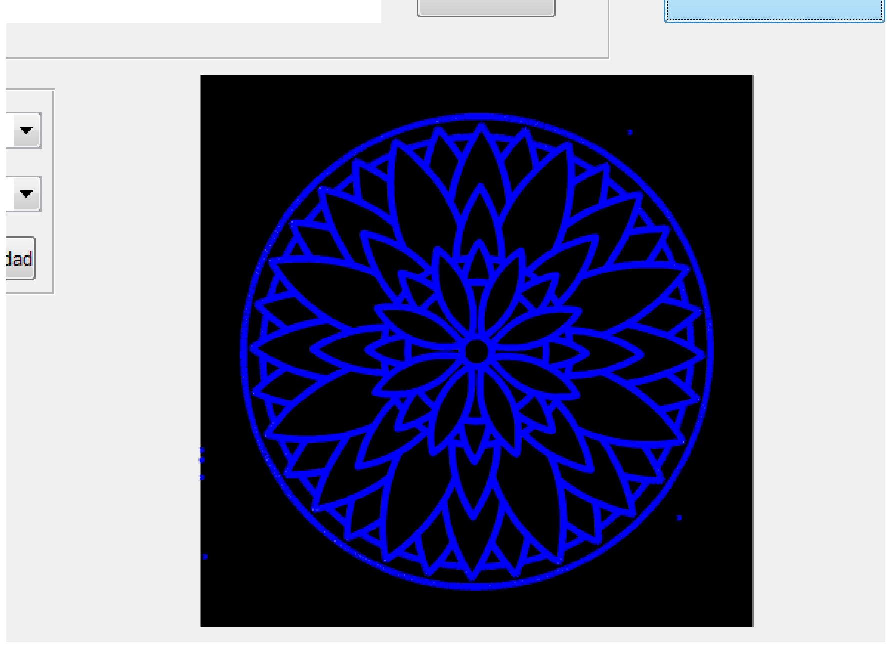


Figura 5.27. *Imagen mandala1 filtrada*

5.7 Posición del robot

En la ventana nº 6 se podrán ver las coordenadas respecto a la base del robot cuando esté dibujando (**Figura 5.28**):

Posición del Robot

[345,-21,400]

Figura 5.28. *Cuadro de coordenadas de posición*

Cada vez que el Robot recibe un punto nos devuelve las coordenadas de su cabezal. Estas coordenadas las mostramos en el cuadro de texto denominado cómo *handles.coordenadas* mediante el comando *set* (subrayado en la **Figura 5.29**):

```
message=fread(t,2); %Leemos el mensaje de 2 digitos maximo (Ok)
waitfor(message,'Ok'); %Esperamos el Ok
poslen = char(fread(t,[1,2])); %Leemos la longitud del mensaje de la posición
poslen = str2double(poslen);   %Lo transformamos a Double
message = char(fread(t,[1,poslen])); %Leemos la posición de longitud poslen
handles.pos(z,:) = str2double(message); %Almacenamos posición recibida
z=z+1;
guidata(hObject,handles); %Actualizamos valores para guardar variable pos antes de drawnow
set(handles.coordenadas,'string',message); %Mostramos coordenadas por pantalla
drawnow; %Actualizamos dibujos de la interfaz para mostrar coordenadas
```

Figura 5.29. *Función para mostrar coordenadas en pantalla*

Además en la variable *handle.pos* vamos almacenando todas las posiciones mostradas para, una vez que el dibujo ha finalizado, graficarlas en una ventana emergente:

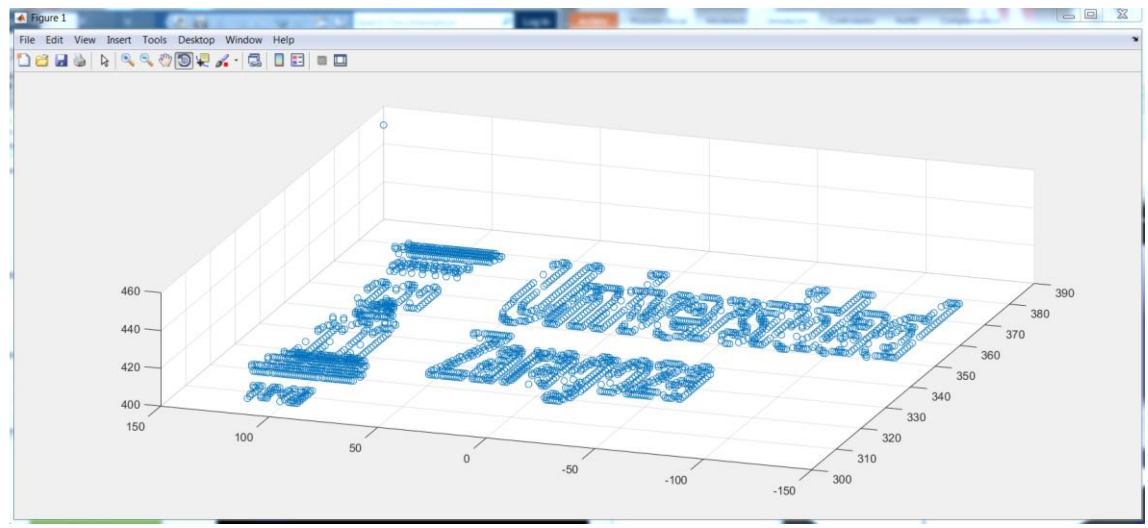


Figura 5.30. *Trayectoria del Robot trazada en Matlab*

5.8 Comenzar a dibujar

Cuando ya se ha elegido la imagen a dibujar y se ha elegido el tratamiento que queremos para dibujarla, además de configurar los parametros de calibración y se

ha establecido correctamente la comunicación con el robot el siguiente paso es comenzar a dibujar. Para comenzar el proceso debemos pulsar el botón 'Dibujar' (**Figura 5.31**):

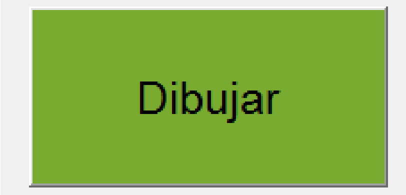


Figura 5.31. *Botón Dibujar*

Este botón comienza a enviar instrucciones al robot según el algoritmo establecido. Veamos el código que se ejecuta tras pulsarlo en la **Figura 5.32**, programado en la GUI:

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

handles=guidata(hObject);

% Función que envía los datos a RobotStudio/Robot para que dibuje
handles.pix=0;
h=get(handles.Filtro,'SelectedObject');
guidata(hObject,handles);
filter=get(h,'Tag'); %Comprobamos que filtro usar
set(handles.stop,'UserData',0);
%handles.pos=[0,0,0];

switch filter

    case 'Pixels'
        Pixels(hObject);
        Redimension(hObject);
        NextPixel(hObject);

    case 'Thicken'
        Thicken(hObject);
        Redimension(hObject); %Binarizamos imagen
        Dibuja(hObject);

    case 'Skeleton'
        Skeleton(hObject)
        Redimension(hObject); %Binarizamos imagen
        Dibuja(hObject);
```

```
    case 'Edge'
        Edge(hObject)
        Redimension(hObject); %Binarizamos imagen
        Dibuja(hObject);
end

set(handles.statusrobot,'string','Dibujo finalizado');

handles=guidata(hObject);

pos=handles.pos;
x = pos(:,1);
y = pos(:,2);
z = pos(:,3);

figure(1);
scatter3(x,y,z)

handles.output = hObject;
guidata(hObject, handles);
fclose(handles.t);
```

Figura 5.32. *Función para ejecutar el dibujo*

Esta función comienza obteniendo el filtro que usuario ha seleccionado mediante el comando *get,* tras esto también se inicializa la variable *stop* que más tarde usaremos para configurar la parada de emergencia.

Una vez que tenemos el filtro deseado mediante un comando *switch…case* dirigimos al robot a la combinación de funciones deseadas, compuesto por el filtro deseado más las funciones '*Redimensión*' y '*Dibuja*' y para el caso del filtro '*Pixels*' los puntos se enviarán con la función *'NextPixel'*. Vamos a verlas en detalle.

5.8.1 Función Redimensión

Mediante esta función logramos que la imagen a tratar tenga unas dimensiones máximas de 297x210 pixeles, consiguiendo así que quepa en un DinA4. El criterio utilizado para la redimensión es maximizar el tamaño del dibujo a las dimensiones marcadas.

```
function [handles]=Redimension(hObject)

handles=guidata(hObject);

if handles.pix==1
   BW=handles.BWP;
else
   BW=handles.BW;
end

s1=handles.s1;
s2=handles.s2;

Rot=0; %Variable para controlar rotación de la imagen durante binarizacion

if s1==s2  %Comprobamos si es imagen cuadrada
DinA4=imresize(BW,[210 210]); %Redimensionamos a 210*210 para cuadrar en un A4
end

if s1>s2  %Comprobamos si s1 es mayor a s2, para fijar máximo
    if (s2/s1)>0.708  %Comprobamos proporción (0.708= 210/297, dimensiones de un Din A4)
      DinA4=imresize(BW,[NaN 210]); % Fijamos dimensión menor=210, la mayor con NaN para mantener proporción
    else
      DinA4=imresize(BW,[297 NaN]); %Fijamos dimensión mayor=297, la menor con NaN para mantener proporción
    end
end
```

```
if s2>s1 %Comprobamos si s2 es mayor a s1, para fijar máximo
    if (s1/s2)>0.708 %Comprobamos proporción (0.708= 210/297, dimensiones de un Din A4)
      Rot=1; %activamos el control de rotación
      DinA4=imresize(BW,[210 NaN]); % Fijamos dimensión menor=210, la mayor con NaN para mantener proporción
    else
      Rot=1; %activamos el control de rotación
      DinA4=imresize(BW,[NaN 297]); % Fijamos dimensión menor=210, la mayor con NaN para mantener proporción
    end
end


sout1=size(DinA4,1); %Dimensión 1 de la imagen de salida
sout2=size(DinA4,2); %Dimensión 2 de la imagen de salida


factor=sout1/s1;
```

Figura 5.33. *Función redimensión*

Con ese criterio, partimos de 3 casos diferentes, que la dimensión 1 (s1) del dibujo sea mayor que la dimensión 2 (s2), que sea al contrario o que ambas sean iguales. Mediante el comando *imresize* fijamos la dimensión que nos limita a 297 o 210 y calculamos proporcionalmente la otra dimensión (*NaN*).

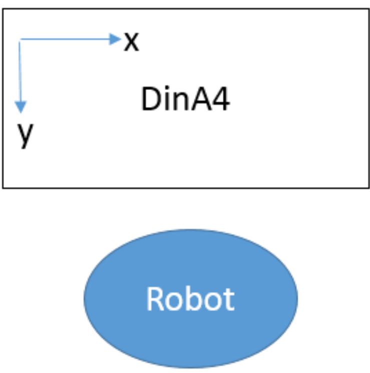


Figura 5.34. *Layout de la isntalación*

Dado el diseño de la estación, mostrado en **figura 5.34,** la dimensión X máxima será 297 y la y máxima será 210. En el caso de que la dimensión 2 (s2) sea mayor que la dimensión 1 (s1) se activa la variable *Rot*=1, esta variable se usará en la función que explicaremos en el siguiente apartado.

Finalmente calculamos el factor de redimensionamiento, denominado '*factor*' que será igual a la dimensión de salida partida por la dimensión de entrada. Este factor se usará en la función *Dibuja* para no perder calidad en la imagen, lo veremos en el siguiente apartado.

5.8.2 Función Dibuja

Esta función posiblemente sea la mas importante del proceso de comunicación, ya que es la que mediante dos bucles '*for*' anidados le va enviando continuamente los puntos al robot o si hay transición entre objetos, envía el mensaje '*next*' para que el robot se levante del papel.

```
function [handles]=Dibuja(hObject)

handles = guidata(hObject);

BW=handles.BW;

[B,~,N,~]=bwboundaries(BW,8,'noholes');

s1=handles.s1;
s2=handles.s2;
factor=handles.factor;
t=handles.t;
handles.z=1;
Rot=handles.Rot;
guidata(hObject, handles);

set(handles.statusrobot,'string','Dibujando'); %Mostramos status 'Dibujando' en pantalla

for k = 1:N

     if k==1 %Comprobamos para el primer punto que dibuje
    fwrite(t,'next'); %Enviamos señal 'next' al robot para que comience a dibujar el objeto
    Check(hObject); %Procedimiento de comprobación de recepción de datos
     end

   boundary = B{k}; %Creamos variable para una unica cell a partir de 'B' obtenida con bwboundaries
   [sz1,~]=size(boundary); %obtenemos tamaño para implementar el bucle for

    for i= 1:3:sz1

        if Rot==1  %Comprobamos si estamos en el caso s2>s1
            Line=[num2str(boundary(i,2)*factor),',',num2str(boundary(i,1)*factor)];
        else
            Line=[num2str(boundary(i,1)*factor),',',num2str(boundary(i,2)*factor)]; %Forma standard para enviar info del punto
        end
   fwrite(t,Line); %Enviamos punto
   Check(hObject);

    end

  if k<N
   fwrite(t,'next'); %Si aun quedan objetos por dibujar enviaremos de nuevo el comando 'next' al robot
   Check(hObject);
  end
end

fwrite(t,'fin');
```

Figura 5.35. *Función 'Dibuja'*

Como se muestra en el código de la **Figura 5.35** el primer paso es obtener los contornos de los diferentes objetos de la imagen, ya transformada a blanco y negro y tratada por el filtro mediante el comando *bwboundaries,* que nos va a dar una variable *'B'* en la que están contenidos los puntos de los contornos, y una variable '*N'* que indica el número de objetos del dibujo.

Con estos datos implementamos el primer bucle, que irá pintando los objetos de 1 hasta N y dentro de este bucle se obtiene la dimensión *sz1* para implementar el segundo bucle, que irá enviando los puntos desde 1 hasta *sz1* al robot.

Como ya se había mencionado en el apartado anterior, los puntos [*x, y*] se multiplican por un *factor* para no perder la nitidez de imagen (nº de pixeles) y el resultado de esta multiplicación es el que se envía al robot.

Si para la imagen la dimensión s2 es mayor que s1, al redimensionarla habremos activado la variable *Rot*=1, por lo que pintaremos las coordenadas inversamente, maximizando s2 a x=297 mm.

5.8.3 Función NextPixel

Para el filtro Pixels el método de dibujo es diverso, ya que de él se obtiene un listado de todos los pixeles que componen el dibujo ordenados por distancia partiendo desde el origen [0,0].
La particularidad es el método en el que se “salta” entre diversos objetos. Para detectarlos se ha de calcular la distancia entre pixeles. Se ha establecido el criterio por el cual si la distancia entre pixeles seguidos en el listado es superior a 5, se envía el mensaje ‘*next*’ al Robot para que levante el bolígrafo del dibujo y vaya al siguiente objeto.

Para enviar los puntos se ha implementado un bucle *for* similar a las funciones anteriores, mostrado en la **Figura 5.36**:

```
function [handles]= NextPixel(hObject)

handles=guidata(hObject);
puntos=handles.puntos;
N=handles.N;
factor=handles.factor;
Rot=handles.Rot;
t=handles.t;
handles.z=1;
guidata(hObject, handles);


if Rot==0  %Comprobamos si se ha rotado la imagen al redimensionar
 j1=1;
 j2=2;
else
 j1=2;
 j2=1;
end

set(handles.statusrobot,'string','Dibujando'); %Mostramos status 'Dibujando'

fwrite(t,'next');
Check(hObject);
```

```
for ii=3:2:N    %Recorremos el listado de puntos


    a=puntos(ii,1)-puntos(ii-2,1);
    b=puntos(ii,2)-puntos(ii-2,2);
    distancia=sqrt(a*a+b*b);        %Calculamos la distancia entre pixeles

   if distancia>5  %Si la distancia es superior a 5 pixeles, enviamos 'next'

    fwrite(t,'next');
    Check(hObject);
    Line=[num2str(puntos(ii,j1)*factor),',',num2str(puntos(ii,j2)*factor)];
    fwrite(t,Line);
    Check(hObject);

   else

   Line=[num2str(puntos(ii,j1)*factor),',',num2str(puntos(ii,j2)*factor)];
   fwrite(t,Line);
   Check(hObject);
   end
end

fwrite(t,'fin');   %Fin del dibujo

handles = guidata(hObject);

guidata(hObject, handles);
```

Figura 5.36. *Función NextPixel*

5.9 Botón 'Stop' para parada de emergencia

En la interfaz se ha incluido un botón que frena automáticamente el envío de información al robot para detener un dibujo (**Figura 5.37**).

Figura 5.37. *Botón Stop*

El modo de funcionamiento es simple, el botón cuenta con un *callback* en la GUI:

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

% Función para la parada del programa
fwrite(handles.t,'stop'); %envía señal de stop al robot
set(handles.stop,'UserData',1);
set(handles.statusrobot,'string','Stop');
guidata(hObject, handles);
```

Figura 5.38. *Función Stop*

Este *callback* al ser pulsado el botón envía la señal '*stop'* al robot y muestra por pantalla el mensaje 'Stop'.

En cuanto al código en RobotStudio el bucle principal de dibujo está programado usando una estructura *while* (**Figura 5.39**):

```
PROC Communication()
   message1:="Inicio";
     while message1<>"stop" do

     SocketReceive client, \str:=message1 \Time:=40;
```

Figura 5.39. *Procedimiento para frenar el Robot*

Si el mensaje recibido es igual a 'stop' el bucle se detiene y el robot frena su movimiento.

6. Resultados experimentales

Una vez explicadas las distintas aplicaciones desarrolladas se va a analizar los resultados obtenidos en simulaciones en ordenador y la comparativa con los resultados en laboratorio.

6.1 Comparativa para tipos de imágenes y filtros

Se ha preparado una comparativa para los 4 filtros que hemos implementado en Matlab, para ello hemos tomado 8 imágenes y las hemos previsualizado en la interfaz gráfica desarrollada. Dentro de estos tipos de imágenes hemos seleccionado formas de distinta dificultad y se han probado también figuras a color:

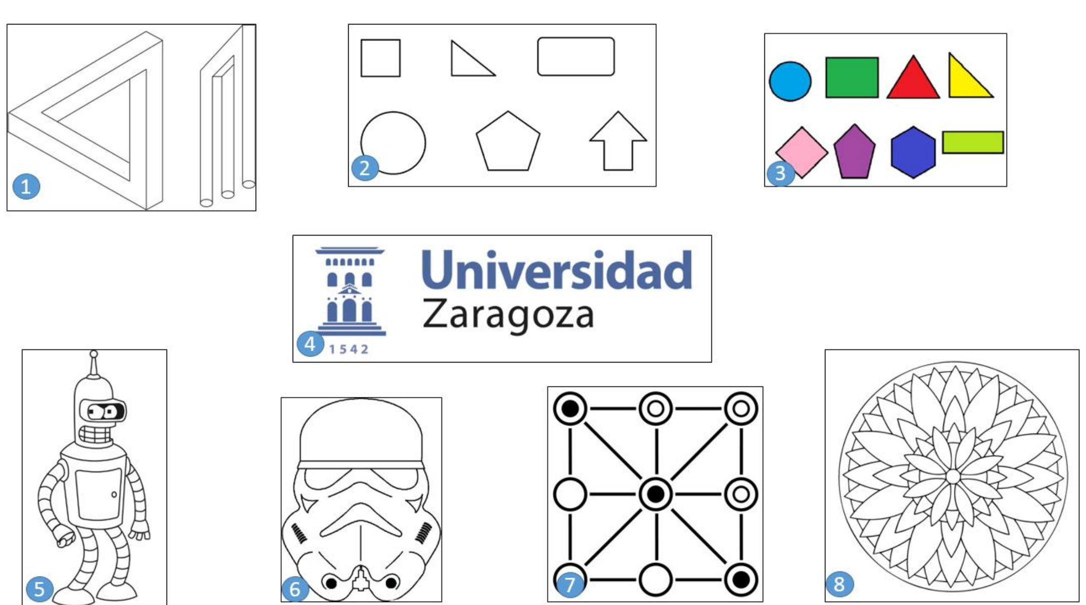


Figura 6.1. *Imágenes para probar*

Veamos los resultados obtenidos en una tabla resumen:

Nº	Imagen	Pixels	Thicken	Skeleton	Edge
1	Escher		Con borde ext		Doble linea
2	Figuras B&W				Doble linea
3	Figuras Color				
4	Logo Univ. Zaragoza				
5	Bender		Con borde ext		Doble linea
6	Stormtropper				Doble linea
7	Tres en Raya				Doble linea
8	Mandala		Con borde ext		Doble linea

Figura 6.2. *Comparativa de filtros usados*

En esta tabla (**Figura 6.2**) se indica con colores la similitud entre la imagen original y después de aplicar el filtro.

Vemos que el filtro que mejor se comporta es '*Edge*' aunque para las figuras que están en blanco y negro se genera una doble línea en cada línea del dibujo original. Esta doble línea es debida a que el método fija las líneas de la imagen filtrada cuando detecta un gradiente entre el valor de pixeles contiguos, por cada línea del dibujo original existen dos gradientes, uno a cada lado de la línea (imagen derecha de la **Figura 6.3**):

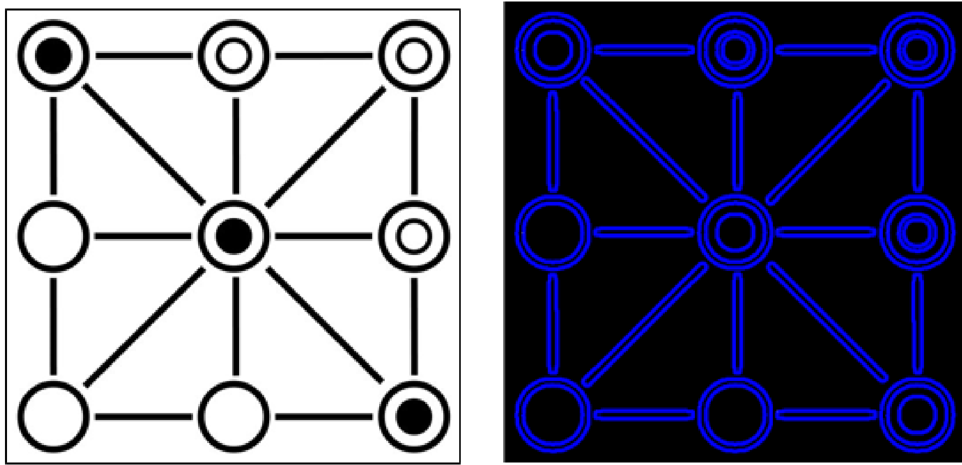


Figura 6.3. *Ejemplo para el filtro 'Edge'*

El filtro 'Pixels' por su parte no funciona correctamente para las imágenes a color pero si con las imágenes en blanco y negro y el filtro 'Thicken' nos devuelve algunas imágenes con una línea formando marco exterior.

Por último el filtro Skeleton observamos que solo nos sirve para figuras en blanco y negro con contornos cerrados. Para otro tipo de imágenes reduce las figuras a una línea que formaría su 'esqueleto', veamos un ejemplo:

Figura 6.4. *Ejemplo para el filtro 'Thicken'*

6.2 Simulaciones en ordenador

Ahora observaremos cómo se comporta el robot al pasarle las imágenes. Para las simulaciones se han utilizado por defecto las velocidades 20 mm/s para dibujar y 100 mm/s para el traslado entre puntos. Estas velocidades nos permiten obtener buena calidad sin dañar el rotulador, para velocidades mayores el robot no se comporta de manera fluida. Se ha diseñado una instalación muy simple en la que se ha insertado el robot IRB 120 y una pequeña mesa situada justo delante (**Figura 6.5**):

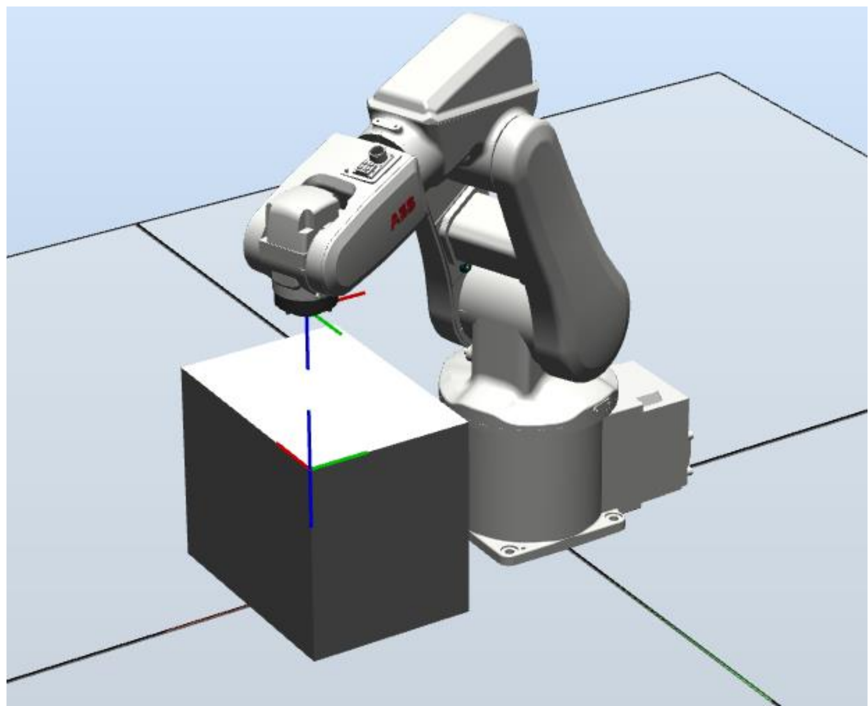


Figura 6.5. *Imagen de la Instalación*

Vamos a tratar dos ejemplos, el primero con el logo de la universidad usando el filtro '*Edge*' y el segundo con un *Mandala* usando el filtro '*Thicken*'.

6.2.1 Dibujo del logo de la universidad. Filtro Edge.

Partimos de la imagen original (**Figura 6.6**):

Figura 6.6. *Logo de la universidad de Zaragoza*

Se ha dibujado la imagen 'logo-universidad' utilizando el método Thicken, veamos la previsualización en Matlab, mostrada en la **Figura 6.7**:

Figura 6.7. *Logo filtrado por método 'Edge'*

Y ahora comparemos la previsualización con el resultado de la simulación en RobotStudio en la **Figura 6.8:**

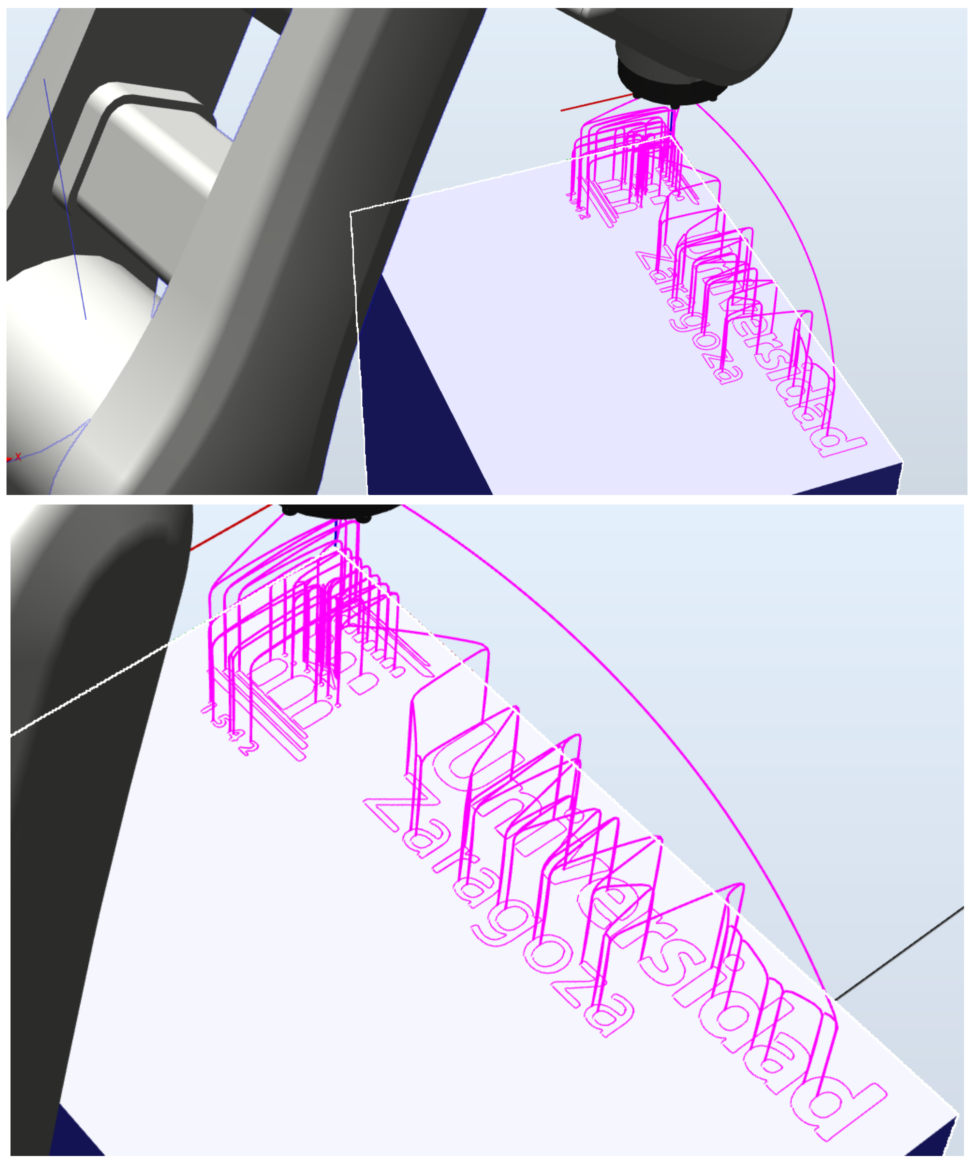


Figura 6.8. *Resultado de la simulación en RobotStudio*

El resultado del trazado de los puntos mostrados en pantalla, graficado en Matlab es el siguiente (**Figura 6.9**):

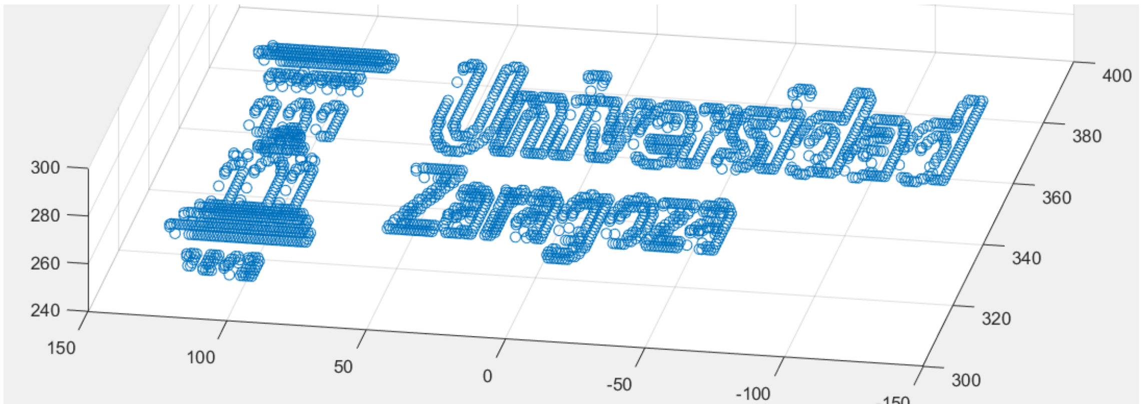


Figura 6.9. *Trazado del Robot representado en Matlab*

Analizando los resultados vemos que para imágenes con colores y letras el filtro 'Edge' funciona correctamente y se obtiene un trazado optimizado que nos permite dibujar la imagen con buena resolución.

El tiempo de dibujo del logo de la Universidad (con *Vdib*=20mm/s y *Vaprox*=100mm/s) es de 4 minutos y 10 segundo.

6.2.2 Dibujo de un Mandala. Filtro Thicken

Veamos ahora cómo se comporta la simulación con el siguiente dibujo, un tipo de dibujo conocido como Mandala (**Figura 6.10**), un tipo de dibujo proveniente del budismo.

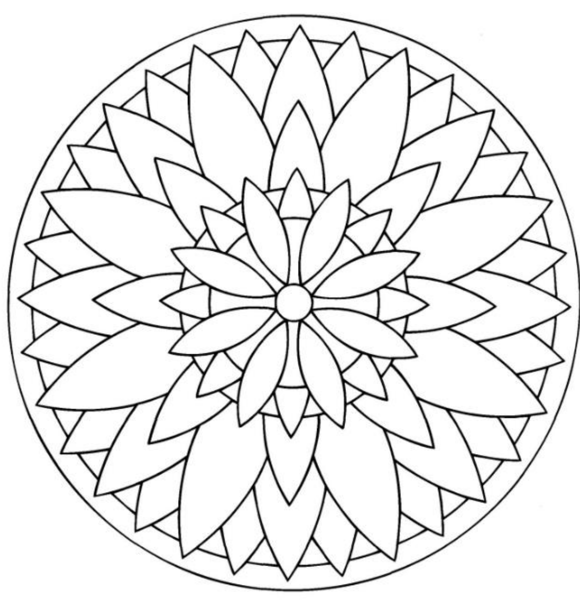


Figura 6.10. *Imagen Mandala*

Usando el filtro Thicken, el resultado de la previsualización en Matlab es el siguiente (**Figura 6.11**):

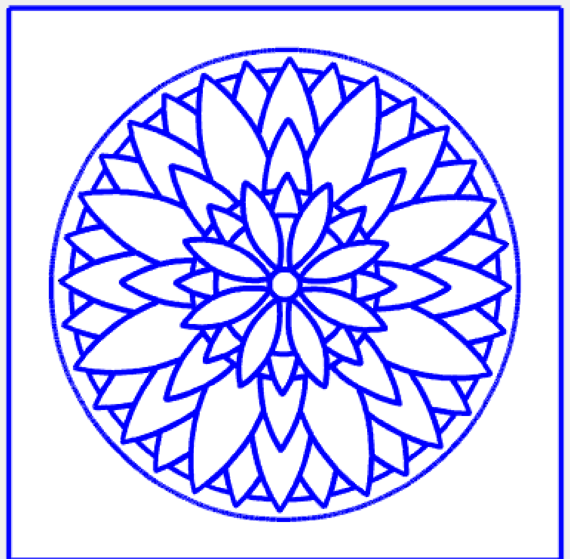
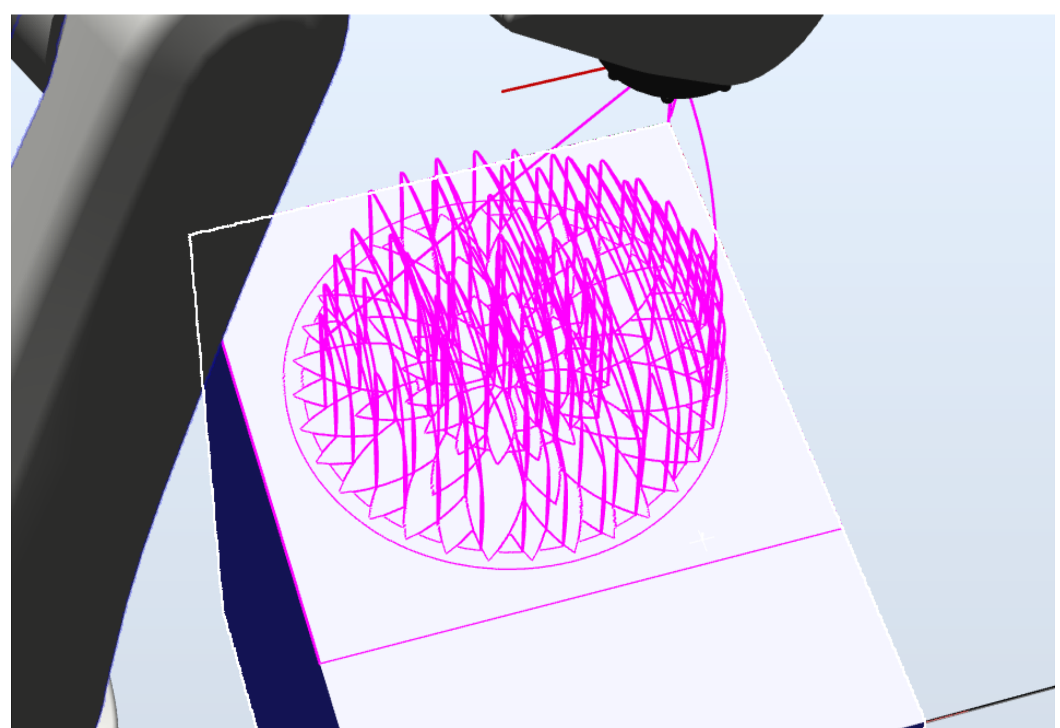


Figura 6.11. *Imagen filtrada por método 'Thicken'*

A priori podemos observar que la definición es buena y no se alteran las líneas, el único 'defecto' introducido en la imagen es el marco exterior que la bordea. Veamos cómo se comporta RobotStudio (**Figura 6.12**).



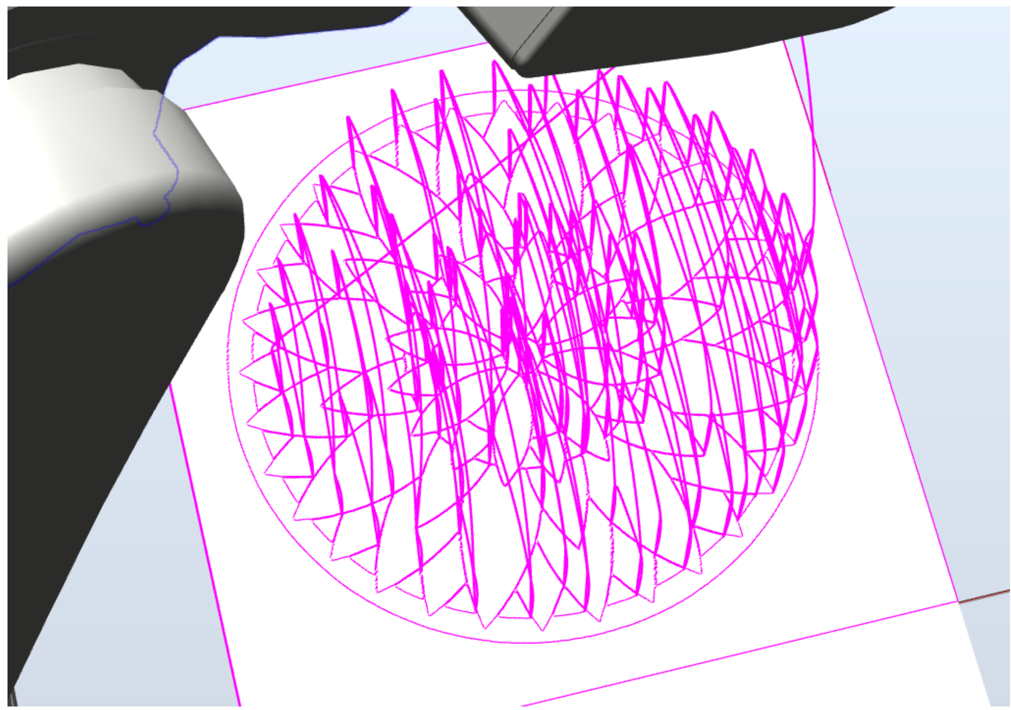


Figura 6.12. *Resultado de la simulación*

El resultado del trazado de los puntos mostrados en pantalla, graficado en Matlab es el siguiente (**Figura 6.13**):

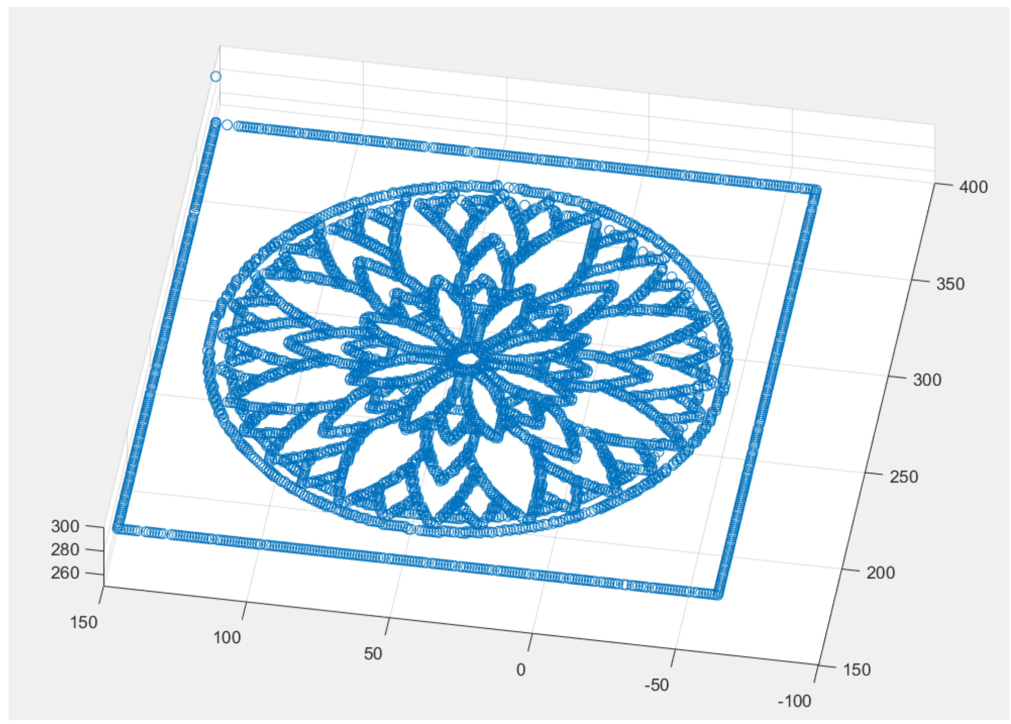


Figura 6.13. *Trazado del Robot representado en Matlab*

El tiempo de dibujo del mandala (con Vdib=20mm/s y Vaprox=100mm/s) es de 10 minutos y 5 segundos.

6.3 Resultados en laboratorio

En el laboratorio de la universidad hemos intentado reproducir las características de la instalación simulada con los recursos disponibles. Al ser un montaje sencillo se ha colocado una mesa frente al brazo robótico y para compensar la altura de la peana sobre la que está instalado el IRB 120 se han colocado una caja y un tablón sobre la mesa como se puede observar en la **figura 6.14**. En la brida de actuación del robot se ha colocado un útil diseñado y fabricado por los profesores del departamento de robótica de la universidad de Zaragoza, el cual nos permite colocar el rotulador en su extremo. Este útil permite el movimiento relativo entre el soporte fijo y el cilindro en el que se inserta el rotulador. Gracias a este sistema se consigue mayor uniformidad en el trazado de líneas.

Como se puede apreciar en la imagen (**Figura 6.14**) para las pruebas se ha dibujado sobre un DinA3:

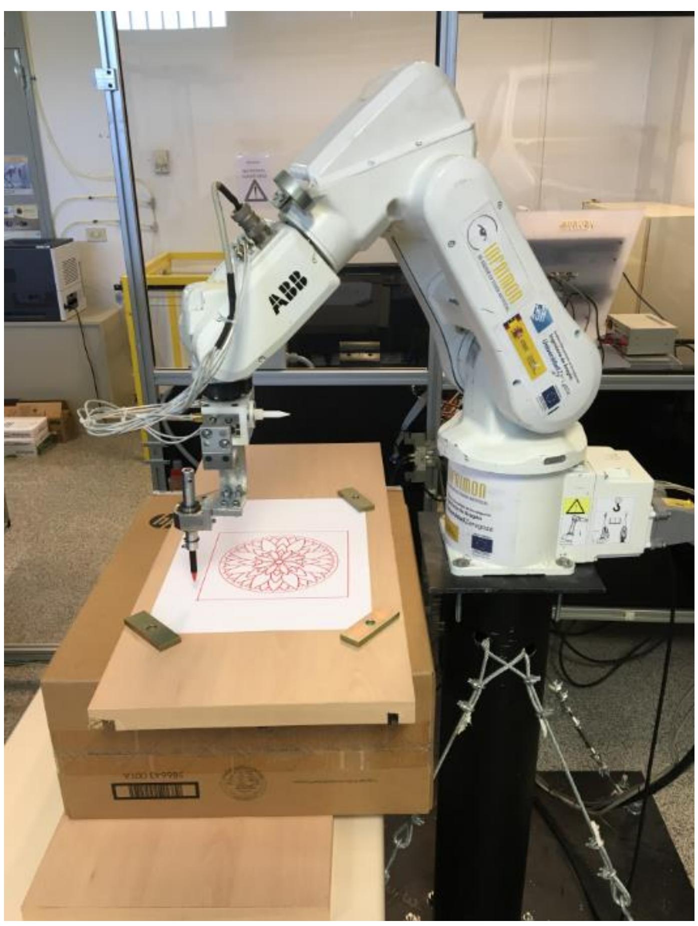


Figura 6.14. *Montaje en el laboratorio*

El principal problema al que nos hemos enfrentado ha sido la calibración del sistema. Cada vez que desmontábamos la instalación o movíamos algunos de sus componentes debíamos invertir tiempo en reconfigurar los parámetros del Robot, sobre todo la altura de dibujo (coordenada Z), por lo que tras varias pruebas se decidió incorporar el cuadro de calibración a la interfaz.

Además de la calibración de la altura se probó con diferentes velocidades de dibujo hasta que se dio con la más apropiada, ya que para altas velocidades (superiores a 50 mm/s) el robot no funcionaba de manera fluida. Además a altas velocidades el rotulador se desgastaba en exceso.

Para conseguir un trazo uniforme se estableció la velocidad de dibujo a 20 mm/s y la velocidad entre trazos a 100 mm/s. Con estos parámetros veamos una por una las figuras con el robot real (**Figuras 6.15 a 6.19**):

Figura 6.15. *Logo de la universidad dibujado por el Robot real*

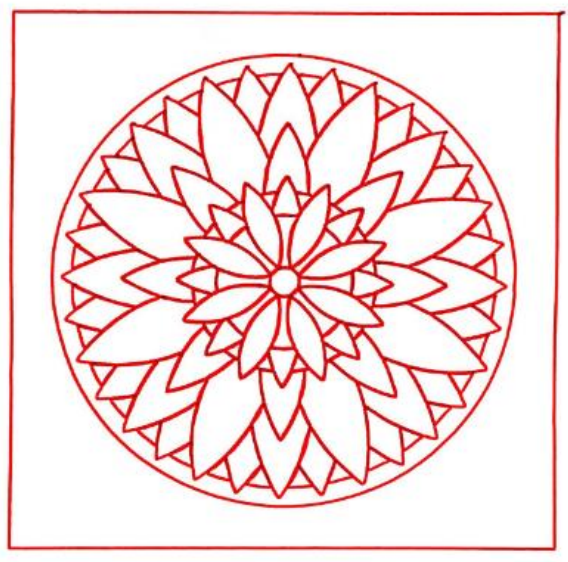


Figura 6.16. *Mandala dibujado por el Robot real*

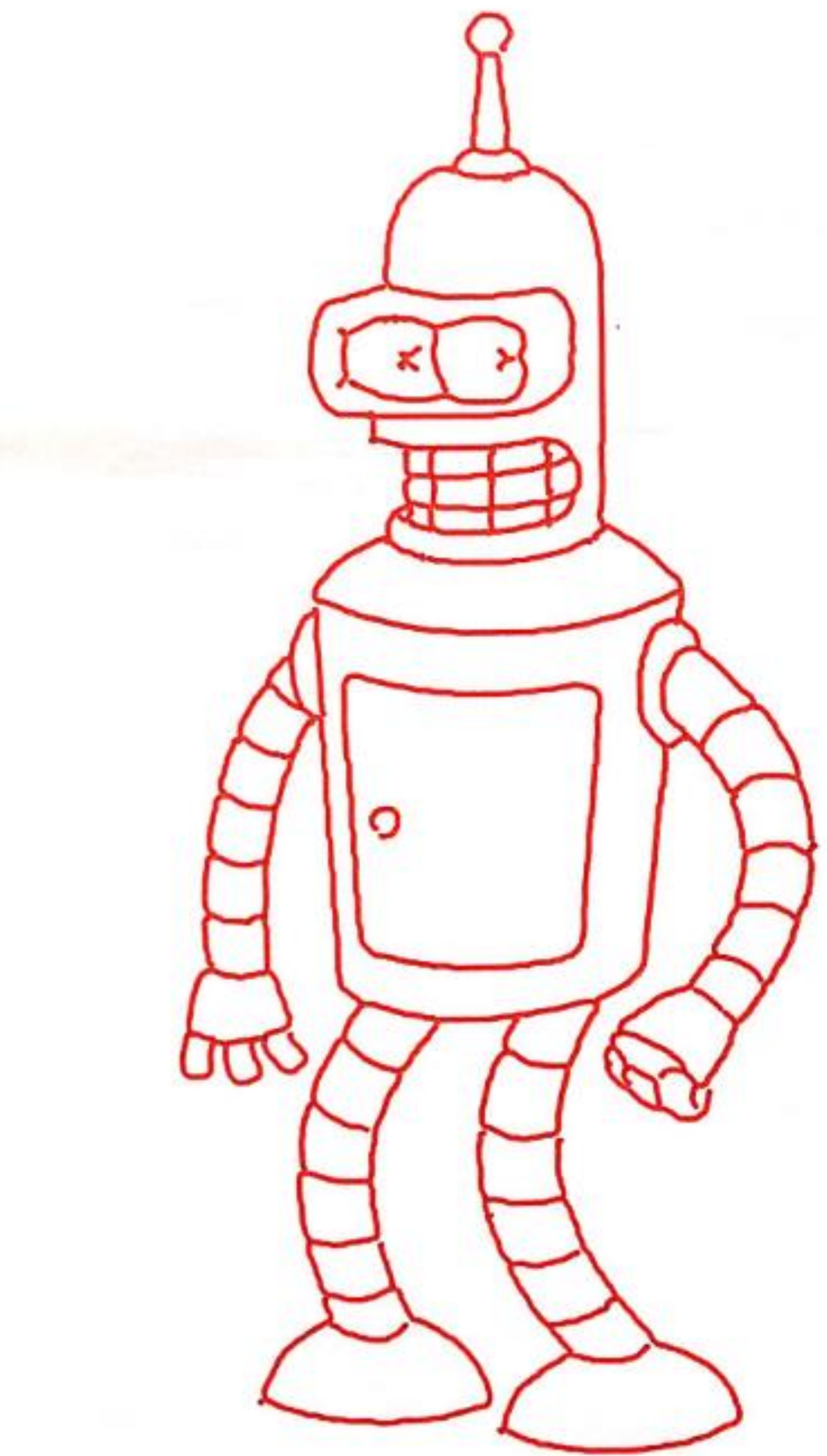


Figura 6.17. *Bender dibujado por el Robot real*

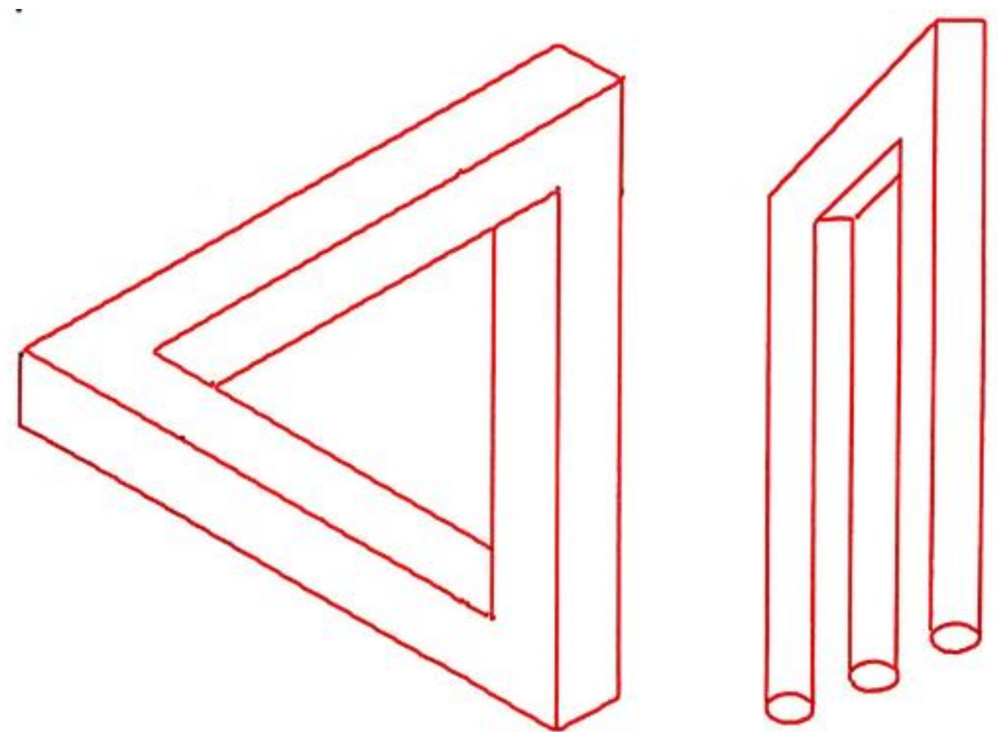


***Figura 6.18.** Figuras de Escher dibujadas por el Robot real*

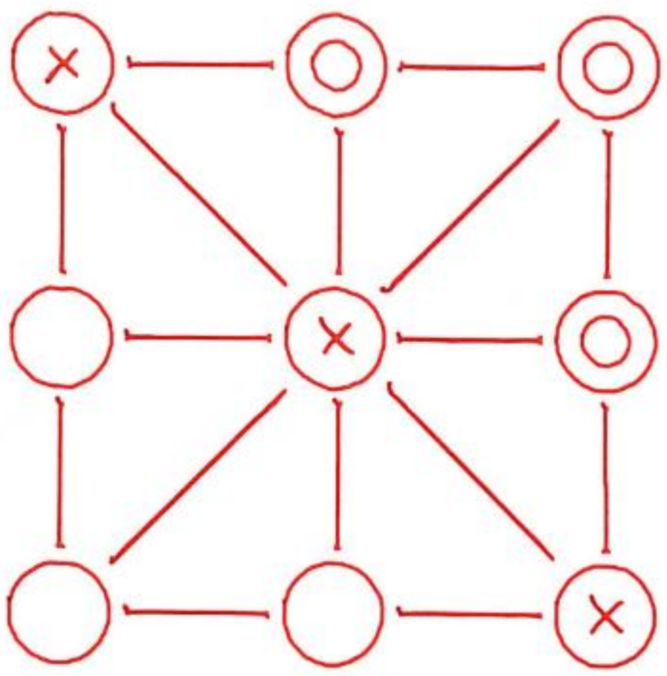


***Figura 6.19.** Tres en raya dibujado por el Robot real*

Como se puede apreciar el resultado obtenido es bueno, mostrando gran similitud entre la imagen de entrada y la obtenida, aunque se pueden apreciar pequeñas distorsiones

de las líneas trazadas. Estas distorsiones son debidas a pequeñas vibraciones del sistema pueden ser causadas tanto por el útil incorporado que sujeta el rotulador como por la ligera inestabilidad caja colocada bajo el folio de dibujo.

Respecto al tiempo de dibujo todos se encuentran entre los 3 minutos y los 10 minutos para el más largo. El tiempo se ha optimizado junto a las velocidades de dibujo, teniendo en cuenta que el principal objetivo era obtener una buena calidad del dibujo de salida.

Con todo esto y teniendo en cuenta los medios disponibles se dan por buenos los resultados obtenidos.

7. Conclusiones y trabajos futuros

Este trabajo tenía como principal objetivo la reproducción de imágenes seleccionadas por el usuario.

El reto al que nos enfrentábamos con este trabajo era el de comunicar dos programas de distintas empresas desarrolladoras y diferente código de programación, para transmitir datos que entran en forma de imagen digital y acabarlos convirtiendo en trayectorias realizadas por el robot para obtener el dibujo final, integrando software con hardware.

Estos problemas me han permitido adquirir mejoras en el manejo de Matlab y en la programación de interfaces gráficas. Además de mejorar mis conocimientos sobre programación de Robots.

Como hemos podido ver en el último apartado de resultados experimentales se ha conseguido un gran parecido entre la imagen tratada y el dibujo realizado por el robot por lo que el resultado en términos de calidad ha sido satisfactorio.

El principal problema detectado es el elevado tiempo de dibujo (entre 3 y 10 minutos dependiendo de la figura). Este tiempo viene marcado por dos factores limitantes, la velocidad de transmisión de datos en el protocolo desarrollado y la precisión del rotulador para trabajar a velocidades elevadas.
En base al primer limitante se plantea como trabajo futuro la optimización del protocolo desarrollado para conseguir que el robot pueda funcionar de forma fluida a mayor velocidad de dibujo.
En base al segundo limitante se puede plantear la mejora del diseño del útil acoplado a la muñeca del robot. Este útil además podría permitir el intercambio de rotuladores durante el dibujo, lo cual permitiría desarrollar una aplicación en la que el robot dibujara en dos o más colores.

Por último y aumentando la dificultad, se puede plantear la variación del formato de los datos de entrada al sistema, como ejemplo se podrían transmitir al sistema grabaciones de voz que sometidas a un tratamiento de reconocimiento de sonidos enviaran al robot las palabras mencionadas por el narrador para que las reproduzca sobre papel.

8. Bibliografía

[1] **Manual del operador RobotStudio 6.04.** ABB Automation Technology Products AB. Publicado 2017

[2] **Manual del operador IRC5 con FlexPendant. RobotWare 6.04.** ABB Automation Technology Products AB. Publicado 2016

[3] **Especificaciones del producto IRB 120-3/0.6.** ABB Automation Technology Products AB. Publicado 2016

[4] **Manual del operador. Introducción a RAPID. RobotWare 6.04.** ABB Automation Technology Products AB. Publicado 2016

[5] **Wikipedia. Varios artículos.**
https://es.wikipedia.org/wiki/

[6] **MATLAB documentation centre**. The MathWorks, Inc. 2017
https://es.mathworks.com/help/matlab/

[7] **Comunidad Mathworks. Varios hilos.**
https://es.mathworks.com/matlabcentral/answers/index/

[8] **Example socket communication between Matlab and Robotstudio.** Autor Miguel López-Guirado Pérez.
https://www.youtube.com/watch?v=JGJt17c69Bk

[9] **Apuntes de la asignatura Automatización Flexible y Robótica .ISA Universidad de Zaragoza.** Autor Josechu Guerrero.

Anexo 1. Manual de usuario

En este manual vamos a explicar el funcionamiento de los diferentes programas y sistemas que componen la aplicación. Los programas usados para la programación han sido RobotStudio 6.07 y Matlab R2017b, ambos para Windows 7 (64 bits).

La aplicación está compuesta por los siguientes ficheros, el código implementado en este trabajo y los archivos necesarios para la puesta en marcha del sistema desarrollado estarán disponibles para su uso futuro dentro del grupo de Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza

- RobotStudio
    - Solution3.rsstn
    - Mod1.mod (programa en RAPID)
- Matlab:
    - PanelControl.m
    - Redimension.m
    - Conexión.m
    - Dibuja.m
    - Thicken.m
    - Skeleton.m
    - Edge.m
    - Pixels.m
    - NextPixel.m
    - Check.m

Puesta en marcha:

- RobotStudio, arrancar la aplicación y abrir la estación Solution3.rsstn, en esa estación está contenida el modulo en Rapid Mod1.mod, cuyo código adjuntamos en el anexo II.

- Matlab, debemos colocar todos los ficheros en una misma carpeta y arrancar MATLAB. Una vez ahí debemos establecer esa carpeta como nuestra carpeta de trabajo (*current Folder*, **Figura I.1**) en el menú de la izquierda de Matlab y ejecutar el fichero PanelControl.m

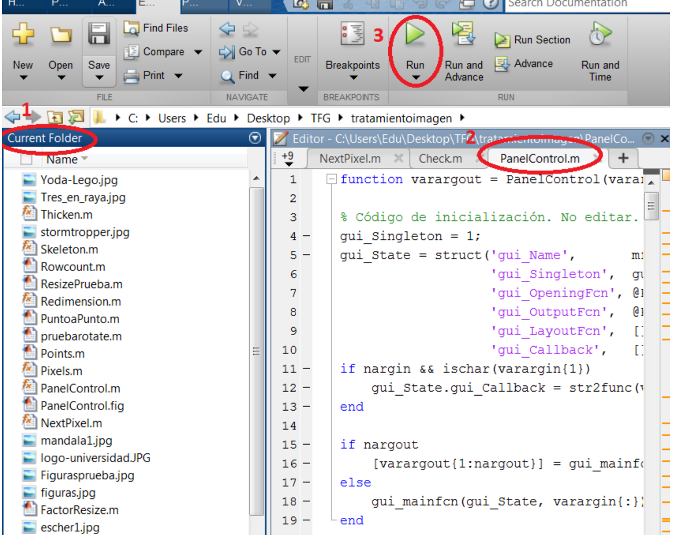


Figura I.1. Inicializar aplicación en Matlab

I.1 Configuración previa de la estación en RobotStudio.

En este apartado se detallará como configurar el programa RobotStudio para poder establecer el socket TCP/IP que permite la comunicación con MATLAB.

Para que el controlador interactúe mediante una conexión de red debemos habilitar la opción de comunicación 'PC Interface'. Esta configuración se establece mediante los siguientes pasos:

1. Seleccionamos administrador de la instalación, en la pestaña Controlador(**Figura I.2**):

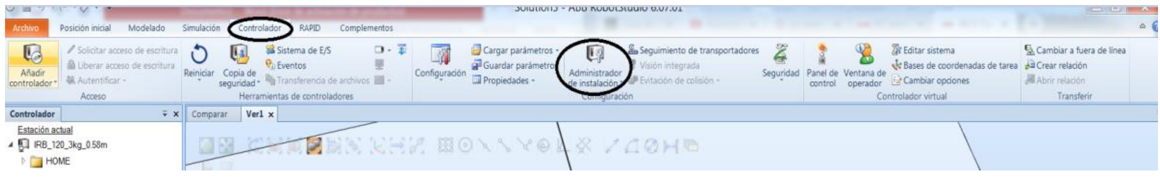


Figura I.2 Administrador de la instalación

2. Se abrirá un desplegable, donde debemos añadir un controlador virtual. Para ello en la pestaña Virtual, pulsaremos sobre el icono '+'. Le daremos un nombre a nuestro nuevo controlador e iremos a buscar la copia de seguridad (**Figura I.3**).

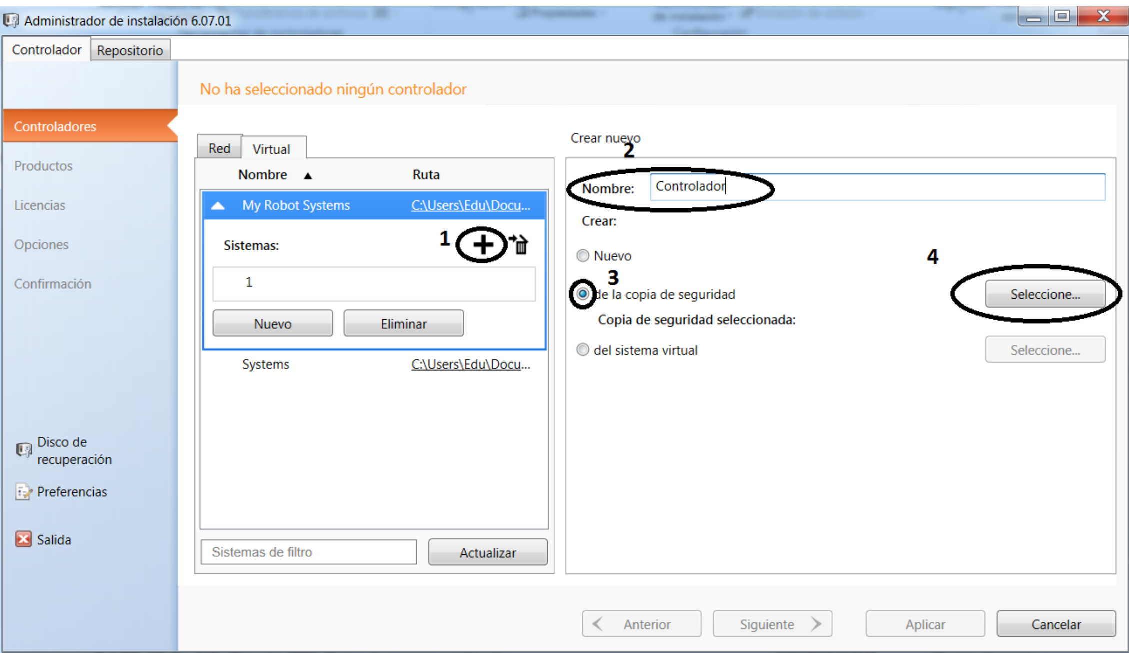


Figura I.3 Seleccionar controlador virtual

3. Seleccionaremos la copia de seguridad 'Backup' de nuestra instalación (en nuestro caso Solution 3->Backups->IRB_120_3kg_0.58m…) y hacemos click en aceptar (**Figura I.4**):

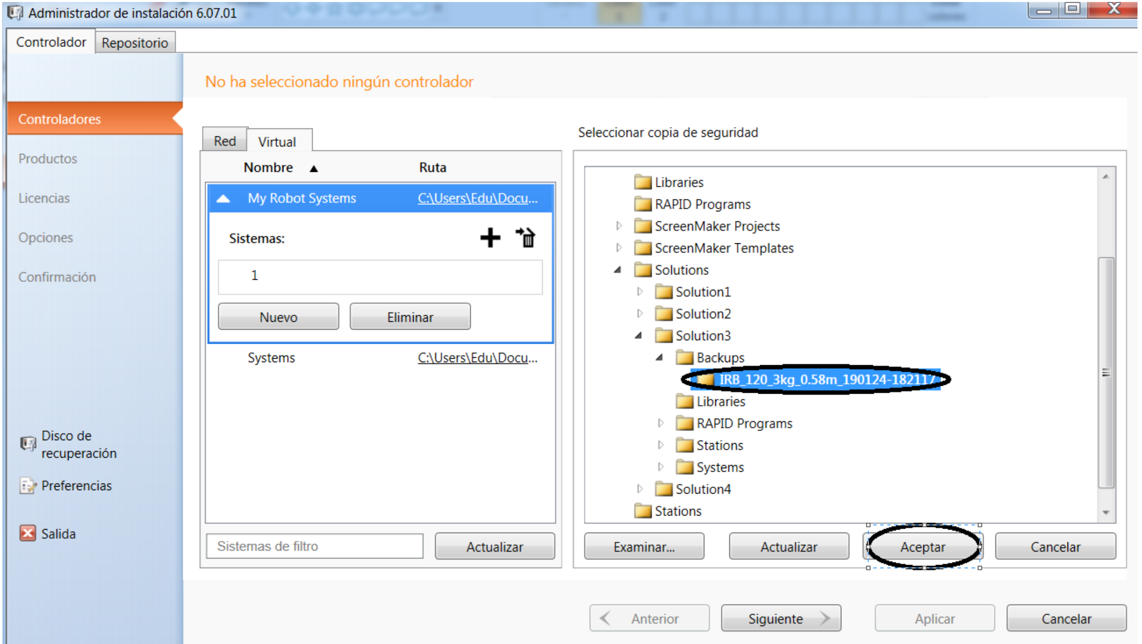


Figura I.4 Selección de la copia de seguridad

4. A continuación, pulsamos en siguiente y debemos ir a la pestaña 'Opciones' situada en el lateral del menú. En esta pantalla se muestra una lista desplegable, ahí tendremos que seleccionar la opción '616-1 PC Interface' en el desplegable de '*Communication*' (**Figura I.5**):

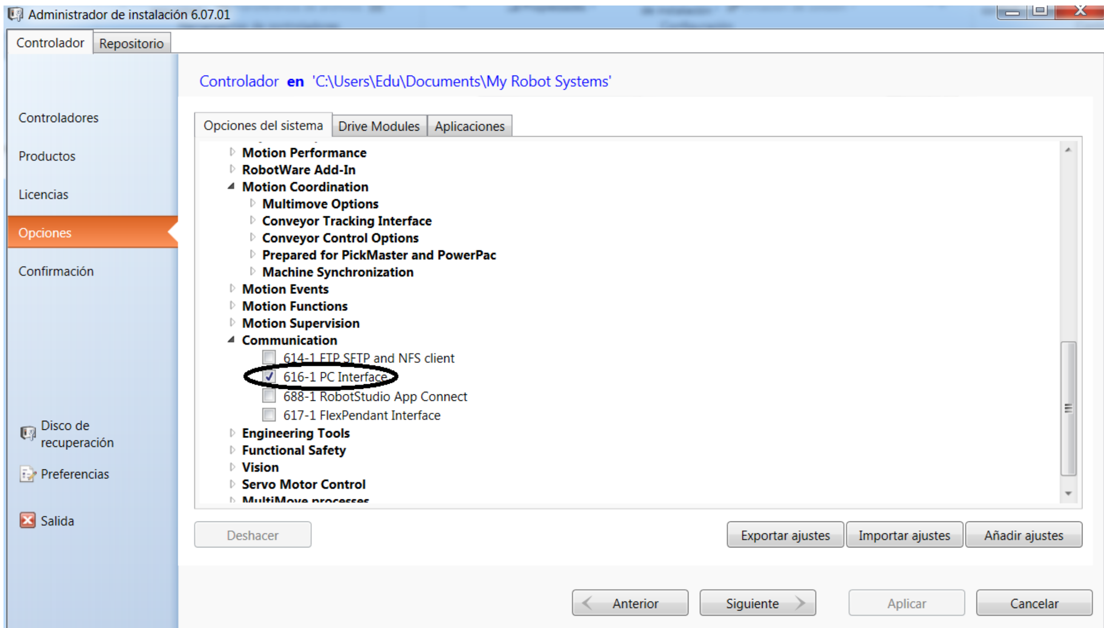


Figura I.5 *Selección de PC Interface en opciones de comunicación*

5. Por último hacemos clic en siguiente una última vez y en la pestaña 'Confirmación' pulsamos sobre 'Aplicar' para establecer los cambios.

I.2 Transferencia de datos al controlador.

Una vez desarrolladas las aplicaciones, debemos transmitir los módulos programados al controlador asociado al robot.

La conexión se establecerá cuando ambos, ordenador y controlador, se encuentren conectados a una misma red. En la pestaña controlador debemos dirigirnos al apartado 'Transferir' y pulsar sobre 'Crear relación'. Se abrirá la siguiente ventana (**Figura I.6**):

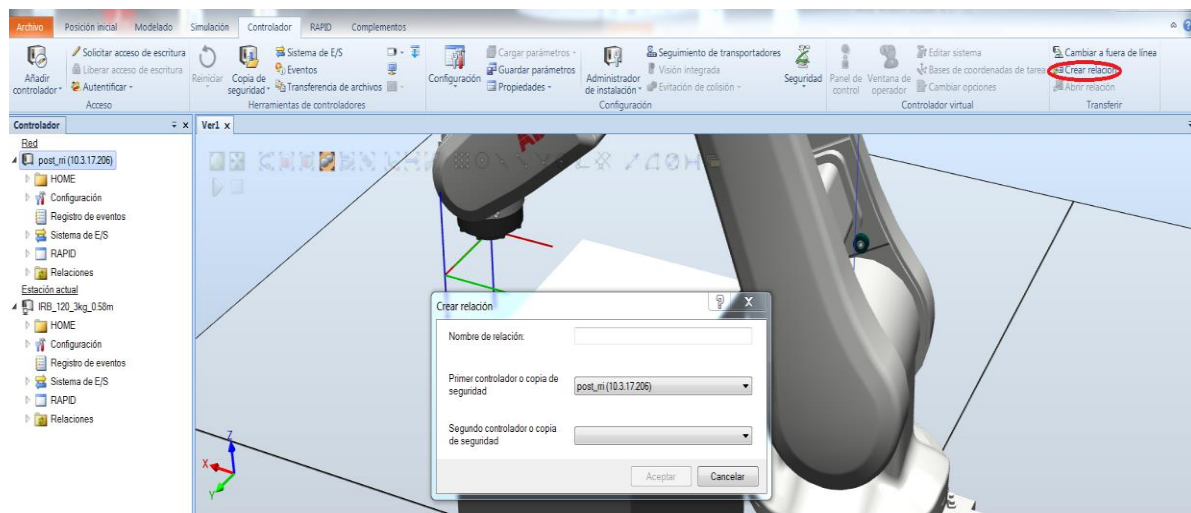


Figura I.6 Selección de los controladores

En ella debemos seleccionar los dos controladores involucrados, el simulado y el real. Una vez que hayamos dado nombre a la relación nos dejará transferir los módulos desde nuestro ordenador al robot real.

I.3 Puesta en marcha del programa

Una vez transferidos los programas al controlador debemos dirigirnos a la ventana de producción del FlexPendant.

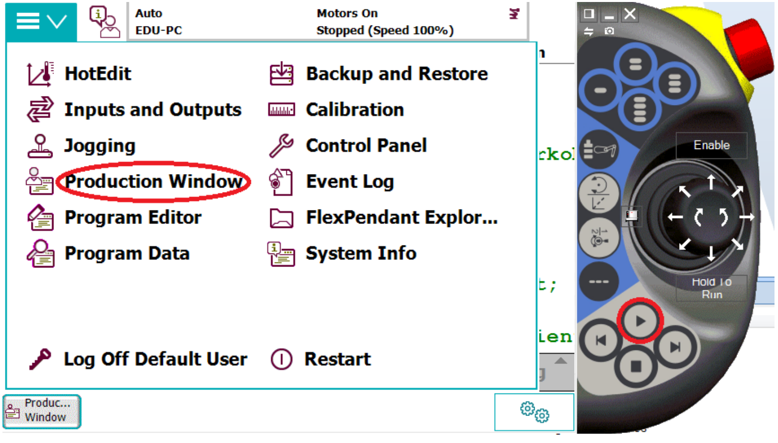


Figura I.7 Selección de la ventana de producción

Para comenzar con nuestra aplicación debemos poner el robot en modo 'Auto' con el siguiente botón del controlador IRC5 y pulsar el botón 'Play' en el Flex Pendant (indicado en la **figura I.6**).

Figura I.8 *Selección del modo automático en el IRC5*

Si quisiéramos ejecutar la aplicación en RobotStudio para su simulación tan solo debemos poner en marcha el robot desde el botón reproducir:

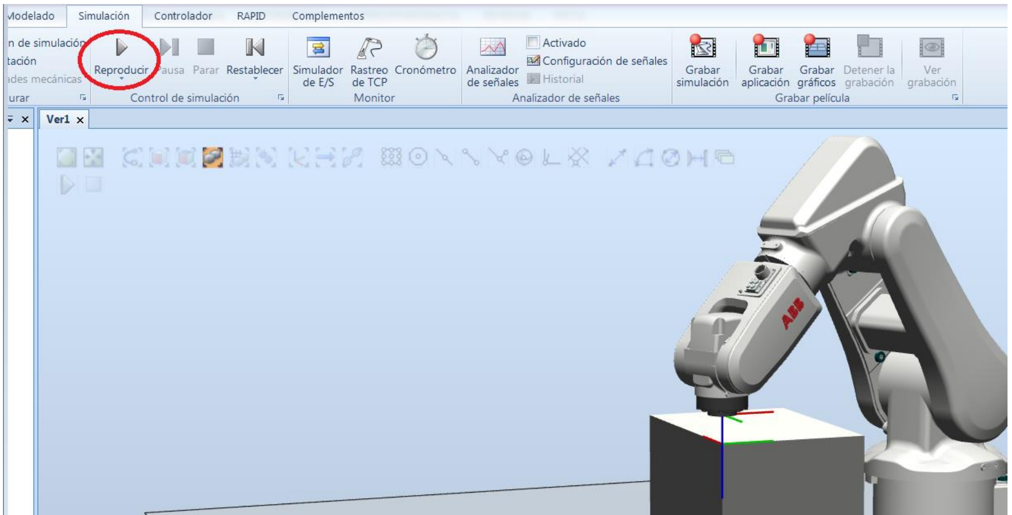


Figura I.9 *Puesta en marcha de la simulación*

Para Matlab es indiferente si conectamos el robot real o el robot simulado, tan solo debemos configurar la IP y el puerto a la que queremos transmitir los datos.

I.4 Funcionamiento de la interfaz ‘Panel de Control’

Vamos a analizar el procedimiento para dibujar nuestra imagen en el Robot, haciendo uso del panel de control mostrado en la **Figura I.9:**

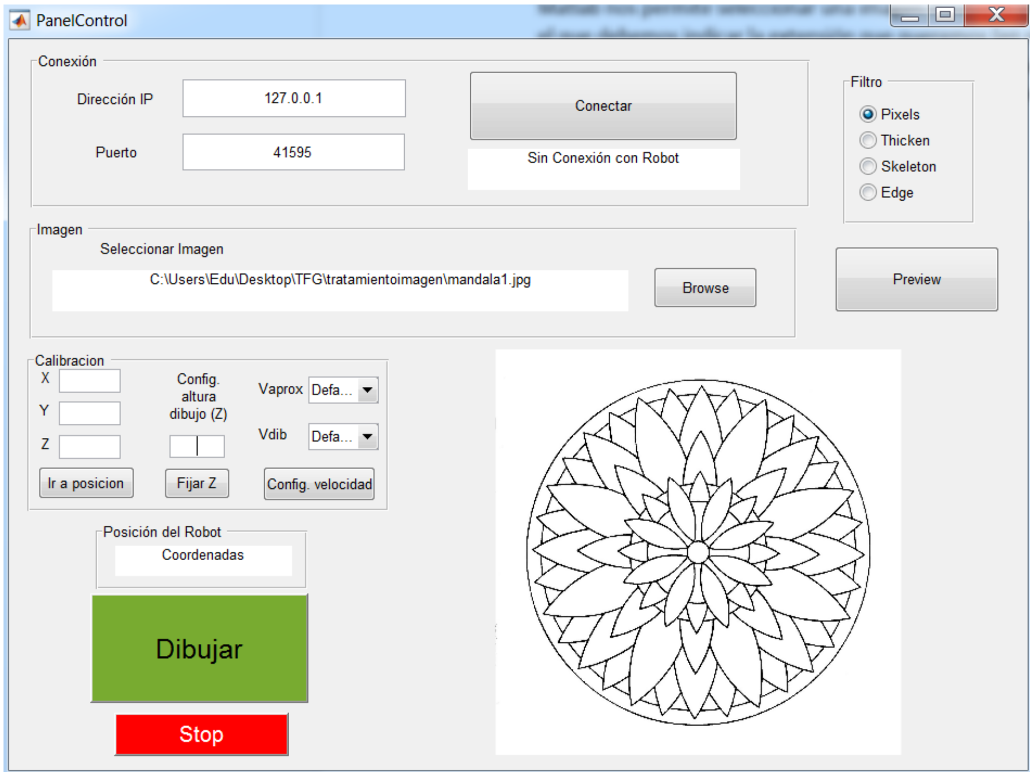


Figura I.9 *Interfaz del panel de control*

I.4.1 Conexión

Primero debemos abrir el fichero *PanelControl.m* y colocarnos en la carpeta de trabajo donde se encuentran el resto de ficheros.

Una vez situados en el panel de control, y habiendo puesto en marcha el robot con nuestro programa (**apartado I.3**) debemos establecer conexión:

1. Introducir la dirección IP del robot en el cuadro habilitado.
2. Introducir el puerto de conexión del Robot en el cuadro habilitado.
3. Pulsar el botón '*Conectar*'
4. Aparecerá un mensaje '*Ok*' en el cuadro inferior al botón 'Conectar'

I.4.2 Selección de imagen

El siguiente paso será seleccionar nuestra imagen:

1. Pulsar el botón '*Browse*'.
2. Se abrirá el navegador de Windows, debemos dirigirnos a la carpeta donde esté guardada nuestra imagen y seleccionarla.
3. La imagen se mostrará en el Panel de Control.

I.4.3 Selección de filtro y previsualización

Para la elección del filtro de dibujo:

1. Debemos seleccionar uno de los 4 filtros (**Figura I.10**):

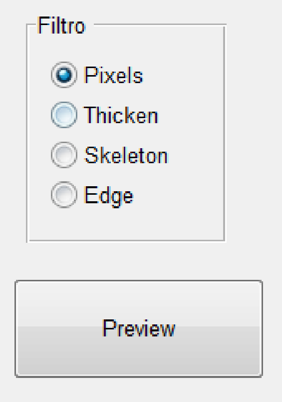


Figura I.10 *Selección de filtro*

2. Una vez elegido pulsar el botón '*Preview*' para ver una previsualización de la imagen, que se mostrará en el Panel de Control.

I.4.4 Calibración

Para el calibrado del Robot (**Figura I.11**), podremos fijar la altura de dibujo y las velocidades tanto de dibujo como de traslación entre figuras. Además podemos controlar el robot en modo manual para que se dirija al punto que le ordenemos.

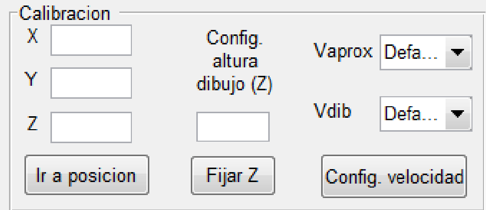


Figura I.11 *Opciones de calibración*

Ir a posición:

1. Indicar en el cuadro correspondiente las coordenadas X, Y, Z del punto al que deseamos mover el extremo del Robot.
2. Hacer clic sobre el botón '*Ir a posición*'. El Robot se desplazara al punto deseado.

Fijar altura de dibujo:

1. Indicar altura de dibujo deseada, el punto de cota Z=0 está situado a +25 cm sobre la altura de la base del robot.
2. Hacer clic sobre el botón '*Fijar Z*'. Ojo, El robot almacenará la variable solo durante un ciclo de dibujo. Al pulsar el botón, dibujará un punto a la altura indicada.

Fijar velocidades:

1. Hacer clic sobre el desplegable de la velocidad que deseemos configurar. *Vaprox* es la velocidad de traslación entre puntos ‘con bolígrafo separado del papel’. *Vdib* es la velocidad con la que dibuja.
2. Pulsar el botón ‘*Config. Velocidad’*, el robot almacenará las variables solo durante un ciclo de dibujo.

I.4.5 Comienzo del dibujo

Una vez fijados los parámetros deseados, esté elegida la imagen y se haya establecido conexión correctamente con el robot. Para comenzar a dibujar:

1. Pulsar el botón ‘Dibujar’ (**Figura I.12**)

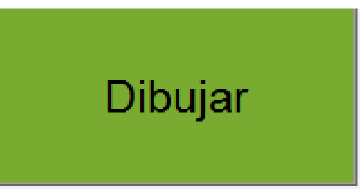


Figura I.12 *Botón dibujar*

2. Se mostrará por pantalla el status ‘Dibujando’
3. Una vez el Robot haya acabado de dibujar, se mostrará el status ‘Dibujo finalizado’

I.4.6 Parada de emergencia

En caso de que se produzca algún problema y se necesite frenar al robot se puede conseguir de 3 formas distintas:

1. Pulsando el botón de ‘Hombre Muerto’ de color rojo del *Flex Pendant*.

Figura I.13 *Seta de emergencia del Flex Pendant*

2. Pulsando la seta de emergencia del controlador IRC5 (**Figura I.14**).

Figura I.14 *Seta de emergencia del controlador IRC5*

3. Pulsando el botón 'Stop' del panel de control (**Figura I.15**).

Figura I.15 *Botón de parada del panel de control*

I.5 Tabla de funcionalidad de los elementos del panel de control.

Vamos a describir la función de cada elemento de la interfaz:

Elemento	**Función**
Dirección IP 127.0.0.1	Cuadro de texto editable en el que se debe indicar la dirección IP del Robot. Se deberá indicar de la forma *127.0.0.1*, como ejemplo.
Puerto 41595	Cuadro de texto editable en el que se debe indicar el puerto de conexión del Robot.
Conectar	Botón que sirve para establecer conexión entre Matlab y el Robot.

Sin Conexión con Robot	Cuadro de texto que mostrará el Status del proceso.
Filtro Pixels Thicken Skeleton Edge	Botonera que permite seleccionar el filtro con el que se tratará la imagen antes de dibujarla.
Preview	Botón de previsualización de la imagen, mostrará la imagen sometida al filtro seleccionado.
Imagen Seleccionar Imagen	Cuadro de texto que mostrará la ruta de la imagen seleccionada en nuestro ordenador.
Browse	Botón de selección de imagen, abrirá el explorador de Windows.
Calibracion X Y Z Ir a posicion	Cuadro de control que nos permite hacer llegar al robot a la posición X, Y, Z deseada. Medidas en mm, origen en la base del robot. El botón 'Ir a posición' enviará al Robot el punto indicado.
Config. altura dibujo (Z) Fijar Z	Cuadro de control que nos permite indicar la altura de dibujo. Medida en mm, origen a una altura de 250mm sobre la base del robot.
Vaprox Defa... Vdib Config Default v5 v10 v20 v30 v50 v80 v100 v200	Desplegable que permite seleccionar la velocidad de traslación entre distintos objetos (trazados). Velocidad en mm/s. Default= 100mm/s.

Vdib Defa... Default v5 v10 v20 v30 v50 v80 v100 v200	Desplegable que permite seleccionar la velocidad de trazados del dibujo. Velocidad en mm/s. Default=20 mm/s.
Config. velocidad	Botón que envía al robot las velocidades *Vdib y Vaprox* seleccionadas.
	Cuadro en el que se mostrará la imagen seleccionada o bien la imagen filtrada a modo de previsualización.
Posición del Robot Coordenadas	Cuadro en el que se muestra la posición X, Y, Z del Robot. Posiciones en mm. Origen en la base del Robot.
Dibujar	Botón para comienzo de dibujo. Solo funcionará si se ha seleccionado una imagen y se ha establecido correctamente conexión con el Robot.
Stop	Botón de parada de emergencia.

Anexo II. Código

En este anexo se adjunta el código fuente programado tanto en MATLAB como en RobotStudio:

II.1 Matlab

II.1.1 Conexión

```
%Comunicacion Socket con RobotStudio

handles=guidata(hObject);


t=tcpip(handles.direccionIP,handles.puerto); %establecemos datos de IP y puerto de
conexión
handles.t= t;
fopen(t); % Abrimos comunicación TCP/IP
message= fread(t,2);% Leemos mensaje de RobotStudio
waitfor(message,'Ok'); %Esperamos el Ok

guidata(hObject, handles);
```

II.1.2 Función ‘Redimensión’

```
function [handles]=Redimension(hObject)

handles=guidata(hObject);

if handles.pix==1
   BW=handles.BWP;
else
   BW=handles.BW;
end

s1=handles.s1;
s2=handles.s2;

Rot=0; %Variable para controlar rotación de la imagen durante binarizacion

if s1==s2  %Comprobamos si es imagen cuadrada
DinA4=imresize(BW,[210 210]); %Redimensionamos a 210*210 para cuadrar en un A4
end

if s1>s2  %Comprobamos si s1 es mayor a s2, para fijar máximo
    if (s2/s1)>0.708  %Comprobamos proporción (0.708= 210/297, dimensiones de un Din A4)
      DinA4=imresize(BW,[NaN 210]); % Fijamos dimensión menor=210, la mayor con NaN para
mantener proporción
    else
      DinA4=imresize(BW,[297 NaN]); %Fijamos dimensión mayor=297, la menor con NaN para
mantener proporción
    end
```

```
end


if s2>s1 %Comprobamos si s2 es mayor a s1, para fijar máximo
    if (s1/s2)>0.708 %Comprobamos proporción (0.708= 210/297, dimensiones de un Din A4)
      Rot=1; %activamos el control de rotación
      DinA4=imresize(BW,[210 NaN]); % Fijamos dimensión menor=210, la mayor con NaN para
mantener proporción
    else
      Rot=1; %activamos el control de rotación
      DinA4=imresize(BW,[NaN 297]); % Fijamos dimensión menor=210, la mayor con NaN para
mantener proporción
    end
end


sout1=size(DinA4,1); %Dimensión 1 de la imagen de salida
sout2=size(DinA4,2); %Dimensión 2 de la imagen de salida


factor=sout1/s1;


handles.DinA4=DinA4; %Almacenamos variable para la GUI
handles.BW=BW;
handles.s1=s1;
handles.s2=s2;
handles.sout1=sout1;
handles.sout2=sout2;
handles.factor=factor;
handles.Rot=Rot;
handles.pix=0;


guidata(hObject, handles); %Almacenamos variables

end
```

III.1.3 Función 'Dibuja'

```
function [handles]=Dibuja(hObject)

handles = guidata(hObject);

BW=handles.BW;

[B,~,N,~]=bwboundaries(BW,8,'noholes');

s1=handles.s1;
s2=handles.s2;
factor=handles.factor;
t=handles.t;
handles.z=1;
Rot=handles.Rot;
guidata(hObject, handles);
```

```
set(handles.statusrobot,'string','Dibujando'); %Mostramos status 'Dibujando' en pantalla

for k = 1:N

     if k==1 %Comprobamos para el primer punto que dibuje
    fwrite(t,'next'); %Enviamos señal 'next' al robot para que comience a dibujar el
objeto
    Check(hObject); %Procedimiento de comprobación de recepción de datos
     end

   boundary = B{k}; %Creamos variable para una unica cell a partir de 'B' obtenida con
bwboundaries
   [sz1,~]=size(boundary); %obtenemos tamaño para implementar el bucle for

    for i= 1:3:sz1

        if Rot==1  %Comprobamos si estamos en el caso s2>s1
            Line=[num2str(boundary(i,2)*factor),',',num2str(boundary(i,1)*factor)];
        else
            Line=[num2str(boundary(i,1)*factor),',',num2str(boundary(i,2)*factor)];
%Forma standard para enviar info del punto
        end
   fwrite(t,Line); %Enviamos punto
   Check(hObject);

    end

  if k<N
   fwrite(t,'next'); %Si aun quedan objetos por dibujar enviaremos de nuevo el comando
'next' al robot
   Check(hObject);
  end
end

fwrite(t,'fin');

handles = guidata(hObject);
handles.B=B;
handles.N=N;

guidata(hObject, handles);
```

III.1.4 Función ‘Check’

```
function [handles]=Check(hObject)

handles=guidata(hObject);
t=handles.t;
z=handles.z;


   message=fread(t,2); %Leemos el mensaje de 2 digitos maximo (ok)
   waitfor(message,'Ok'); %Esperamos el Ok
   poslen = char(fread(t,[1,2])); %Leemos la longitud del mensaje de la posición
   poslen = str2num(poslen);   %Lo transformamos a Double
   message = char(fread(t,[1,poslen])); %Leemos la posición de longitud poslen
```

```
    handles.pos(z,:) = str2num(message); %Almacenamos posición recibida
    z=z+1;
    guidata(hObject,handles); %Actualizamos valores para guardar variable pos antes de
drawnow
    set(handles.coordenadas,'string',message); %Mostramos coordenadas por pantalla
    drawnow; %Actualizamos dibujos de la interfaz para mostrar coordenadas


handles.z=z;
guidata(hObject,handles);
```

III.1.5 Función 'Thicken'

```
function [handles]= Thicken(hObject)

handles=guidata(hObject);

cla reset;

Ient = imread(handles.filename); %Lee imagen de entrada
umbralOpt=graythresh(Ient);  %Busca umbral optimo para binarizar
BW = im2bw(Ient,umbralOpt);  %Binariza, BW es una matriz de 0s y 1s
BW=bwmorph(BW,'thicken',Inf);

imshow(BW);

[B,L,N]=bwboundaries(BW,8,'holes');
hold on %Mantiene el plot cuando genera nuevos plots
for k = 1:N
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'b', 'LineWidth', 2)
end

s1=size(BW,1); %Obtenemos dimensión x de la matriz
s2=size(BW,2); %Obtenemos dimensión y de la matriz


handles.s1=s1;
handles.s2=s2;
handles.BW=BW;


guidata(hObject, handles);
```

III.1.6 Función 'Skeleton'

```
function [handles]= Skeleton(hObject)

handles=guidata(hObject);

cla reset;

Ient = imread(handles.filename); %Lee imagen de entrada
umbralOpt=graythresh(Ient);  %Busca umbral optimo para binarizar
```

```
BW = im2bw(Ient,umbralOpt);  %Binariza, BW es una matriz de 0s y 1s
BW=bwmorph(~BW,'skel',Inf);
s1=size(BW,1); %Obtenemos dimensión x de la matriz
s2=size(BW,2); %Obtenemos dimensión y de la matriz


imshow(BW);

[B,L,N]=bwboundaries(BW,8,'holes');
hold on %Mantiene el plot cuando genera nuevos plots
for k = 1:N
   boundary = B{k};
   plot(boundary(:,2), boundary(:,1), 'b', 'LineWidth', 2)
end

handles.s1=s1;
handles.s2=s2;
handles.BW=BW;


guidata(hObject, handles);
```

III.1.7 Función 'Edge'

```
function [handles]= Edge(hObject)

handles=guidata(hObject);

cla reset;

Ient = imread(handles.filename); %Lee imagen de entrada
umbralOpt=graythresh(Ient);  %Busca umbral optimo para binarizar
BW = im2bw(Ient,umbralOpt);  %Binariza, BW es una matriz de 0s y 1s
BW=edge(BW,'Canny');

s1=size(BW,1); %Obtenemos dimensión x de la matriz
s2=size(BW,2); %Obtenemos dimensión y de la matriz


imshow(BW);

[B,L,N]=bwboundaries(BW,8,'holes');
hold on %Mantiene el plot cuando genera nuevos plots
for k = 1:N
   boundary = B{k};
   plot(boundary(:,2), boundary(:,1), 'b', 'LineWidth', 2)
end

handles.s1=s1;
handles.s2=s2;
handles.BW=BW;



guidata(hObject, handles);
```

III.1.8 Función 'Pixels'

```
function [handles]= Pixels(hObject)

handles=guidata(hObject);

cla reset;

Ient = imread(handles.filename); %Lee imagen de entrada
umbralOpt=graythresh(Ient);  %Busca umbral optimo para binarizar
BWP = im2bw(Ient,umbralOpt);  %Binariza, BW es una matriz de 0s y 1s
BWP = bwmorph(~BWP,'skel',Inf);

s1=size(BWP,1); %Obtenemos dimensión x de la matriz
s2=size(BWP,2); %Obtenemos dimensión y de la matriz

[x,y]=find(BWP>0);
N=length(x);
puntos=[x,y, zeros(N,1)];

%ordenar puntos por distancia
%Poner como siguiente punto el más cercano y así sucesivamente
for i=1:N-2,
    p1=puntos(i,1:2);

    vp1=repmat(p1,N-i,1);
    vp2=puntos(i+1:N,1:2);
    %calcular distancias
    a= vp1(:,1)-vp2(:,1);
    b= vp1(:,2)-vp2(:,2);
    distancia = sqrt(a.*a + b.*b);
    puntos(i+1:N,3) = distancia;

    [val,vali]=min(puntos(i+1:N,3)); %Val=minimo valor de la 3ª columna de puntos, vali
nº de posiciones en la matriz respecto la actual
    tempi=puntos(i+1,1:3);
    puntos(i+1,1:3)=puntos(i+vali,1:3);
    puntos(i+vali,1:3)=tempi;

end

handles.puntos=puntos;
handles.s1=s1;
handles.s2=s2;
handles.BWP=BWP;
handles.N=N;
handles.pix=1;

imshow(BWP);


hold
for ii=1:5:N
    plot(puntos(ii,2),puntos(ii,1),'.b')
    pause(0.001)
end
```

```
guidata(hObject, handles);
```

II.1.9 Función 'NextPixel'

```
function [handles]= NextPixel(hObject)

handles=guidata(hObject);
puntos=handles.puntos;
N=handles.N;
factor=handles.factor;
Rot=handles.Rot;
t=handles.t;
handles.z=1;
guidata(hObject, handles);


if Rot==0  %Comprobamos si se ha rotado la imagen al redimensionar
 j1=1;
 j2=2;
else
 j1=2;
 j2=1;
end

set(handles.statusrobot,'string','Dibujando'); %Mostramos status 'Dibujando'

fwrite(t,'next');
Check(hObject);

for ii=3:2:N    %Recorremos el listado de puntos

    a=puntos(ii,1)-puntos(ii-2,1);
    b=puntos(ii,2)-puntos(ii-2,2);
    distancia=sqrt(a*a+b*b);       %Calculamos la distancia entre pixeles

   if distancia>5  %Si la distancia es superior a 5 pixeles, enviamos 'next'

    fwrite(t,'next');
    Check(hObject);
    Line=[num2str(puntos(ii,j1)*factor),',',num2str(puntos(ii,j2)*factor)];
    fwrite(t,Line);
    Check(hObject);

   else

   Line=[num2str(puntos(ii,j1)*factor),',',num2str(puntos(ii,j2)*factor)];
   fwrite(t,Line);
   Check(hObject);
   end
end

fwrite(t,'fin');   %Fin del dibujo

handles = guidata(hObject);
```

```
guidata(hObject, handles);
```

II.1.10 Interfaz gráfica (GUI) 'PanelControl'

```
function varargout = PanelControl(varargin)

% Código de inicialización. No editar.
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PanelControl_OpeningFcn, ...
                   'gui_OutputFcn',  @PanelControl_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

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

% Choose default command line output for PanelControl
handles.output = hObject;
% Update handles structure

guidata(hObject, handles);
set(handles.axes1,'xtick',[], 'ytick',[]);
set(handles.stop,'UserData',0);

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


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

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

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

% Función para la parada del programa
fwrite(handles.t,'stop'); %envía señal de stop al robot
set(handles.stop,'UserData',1);
set(handles.statusrobot,'string','Stop');
guidata(hObject, handles);



function Browse_Callback(hObject, eventdata, handles)

%Función que nos permite seleccionar una imagen de nuestro ordenador para dibujarla

cla reset;
[filename,pathname]=uigetfile({'.jpg'},'File selector');  %Consigue el nombre del
archivo y la ruta
handles.filename=filename;  %Guardamos el nombre del archivo en la estructura handles
fullpath=strcat(pathname,filename); %Concatenamos la ruta y el nombre del archivo
set(handles.text2,'string',fullpath);  %Lo mostramos en el cuadro de texto
axes(handles.axes1);
A=imread(filename,'jpg'); %Leemos la imagen, formato JPG
imshow(A); %Mostramos la imagen en el panel de control
set(handles.axes1,'xtick',[], 'ytick',[]);  %Mantenemos los ejes sin numeración
handles.output = hObject;
guidata(hObject, handles);   %Actualizamos la estructura handles

function dirIP_Callback(hObject, eventdata, handles)

% Función que nos permite obtener la IP que introduce el usuario

direccionIP = get(handles.dirIP,'string'); %Lee y almacena la dirección IP
handles.direccionIP = direccionIP; %Guardamos en estructura handles
guidata(hObject, handles); %actualizamos guidata

% --- Executes during object creation, after setting all properties.

function dirIP_CreateFcn(hObject, eventdata, handles)

% Muestra la celda para recoger la IP con fondo blanco

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


function Port_Callback(hObject, eventdata, handles)

% Obtenemos el puerto indicado por el usuario

puerto = get(handles.Port,'string');
puerto=str2double(puerto);
handles.puerto = puerto;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
```

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

% Muestra la celda para recoger el nº de puerto con fondo blanco

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


% --- Executes on button press in Conectar.
function Conectar_Callback(hObject, eventdata, handles)

%Establece conexión con RobotStudio/Robot

Conexion;
set(handles.statusrobot,'string','Ok');
guidata(hObject, handles);



% --- Executes when selected object is changed in Filtro.
function Filtro_SelectionChangedFcn(hObject, eventdata, handles)

% Obtenemos que opción de filtro a elegido el usuario (líneas o pixels)

h=get(handles.Filtro,'SelectedObject');
type=get(h,'Tag');
handles.type=type;
handles.output = hObject;
guidata(hObject, handles);


% --- Executes on button press in Dibujar.
function Dibujar_Callback(hObject, eventdata, handles)

handles=guidata(hObject);

% Función que envía los datos a RobotStudio/Robot para que dibuje
handles.pix=0;
h=get(handles.Filtro,'SelectedObject');
guidata(hObject,handles);
filter=get(h,'Tag'); %Comprobamos que filtro usar
set(handles.stop,'UserData',0);
%handles.pos=[0,0,0];

switch filter

    case 'Pixels'
        Pixels(hObject);
        Redimension(hObject);
        NextPixel(hObject);

    case 'Thicken'
        Thicken(hObject);
        Redimension(hObject); %Binarizamos imagen
        Dibuja(hObject);
```

```
    case 'Skeleton'
        Skeleton(hObject)
        Redimension(hObject); %Binarizamos imagen
        Dibuja(hObject);

    case 'Edge'
        Edge(hObject)
        Redimension(hObject); %Binarizamos imagen
        Dibuja(hObject);
end

set(handles.statusrobot,'string','Dibujo finalizado');

handles=guidata(hObject);

pos=handles.pos;
x = pos(:,1);
y = pos(:,2);
z = pos(:,3);

figure(1);
scatter3(x,y,z)

handles.output = hObject;
guidata(hObject, handles);
fclose(handles.t);


% --- Executes on button press in Preview.
function Preview_Callback(hObject, eventdata, handles)

h=get(handles.Filtro,'SelectedObject');
type=get(h,'Tag'); %Comprobamos con que filtro trazar
set(handles.stop,'UserData',0);

switch type    %Ejecutamos la función correspondiente al filtro seleccionado

    case 'Pixels'
        Pixels(hObject)

    case 'Thicken'
        Thicken(hObject)

    case 'Skeleton'
        Skeleton(hObject)

    case 'Edge'
        Edge(hObject)

end

guidata(hObject, handles);

function xcalib_Callback(hObject, eventdata, handles)
%xcalib = get(handles.xcalib,'string');
%handles.xcalib = xcalib;
%guidata(hObject, handles);
```

```
% Hints: get(hObject,'String') returns contents of xcalib as text
%        str2double(get(hObject,'String')) returns contents of xcalib as a double


% --- Executes during object creation, after setting all properties.

function xcalib_CreateFcn(hObject, eventdata, handles)
% hObject    handle to xcalib (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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



function ycalib_Callback(hObject, eventdata, handles)

%ycalib = get(handles.ycalib,'string');
%handles.ycalib = ycalib;
%guidata(hObject, handles);

% Hints: get(hObject,'String') returns contents of ycalib as text
%        str2double(get(hObject,'String')) returns contents of ycalib as a double


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

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

end



function zcalib_Callback(hObject, eventdata, handles)
%zcalib = get(handles.zcalib,'string');
%handles.zcalib = zcalib;
%guidata(hObject, handles);

% Hints: get(hObject,'String') returns contents of zcalib as text
%        str2double(get(hObject,'String')) returns contents of zcalib as a double


% --- Executes during object creation, after setting all properties.
function zcalib_CreateFcn(hObject, eventdata, handles)
% hObject    handle to zcalib (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

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



% --- Executes on button press in irposicion.
function irposicion_Callback(hObject, eventdata, handles)

handles=guidata(hObject);
t = handles.t;

xcalib = char(get(handles.xcalib,'string'));
ycalib = char(get(handles.ycalib,'string'));
zcalib = char(get(handles.zcalib,'string'));


fwrite(t,'calib');
   message=fread(t,2);
   waitfor(message,'Ok');
   Line=[xcalib,',',ycalib,',',zcalib];
   fwrite(t,Line);
   poslen= char(fread(t,[1,2]));
   poslen = str2double(poslen);
   message= char(fread(t,[1,poslen]));
   set(handles.coordenadas,'string',message);
   handles.message;
   drawnow;
   guidata(hObject, handles);





function Vpos_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
content=cellstr(get(hObject,'String'));
pop_choice=content(get(hObject,'Value'));
if (strcmp(pop_choice,'Default'))
    handles.Vdib='100';
elseif (strcmp(pop_choice,'v5'))
    handles.Vpos='5';
elseif (strcmp(pop_choice,'v10'))
    handles.Vpos='10';
elseif (strcmp(pop_choice,'v20'))
    handles.Vpos='20';
elseif (strcmp(pop_choice,'v30'))
    handles.Vpos='30';
elseif (strcmp(pop_choice,'v50'))
    handles.Vpos='50';
elseif (strcmp(pop_choice,'v80'))
    handles.Vpos='80';
```

```
elseif (strcmp(pop_choice,'v100'))
    handles.Vpos='100';
elseif (strcmp(pop_choice,'v200'))
    handles.Vpos='200';
end
guidata(hObject, handles);

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

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



function Vdib_Callback(hObject, eventdata, handles)
handles=guidata(hObject);
content=cellstr(get(hObject,'String'));
pop_choice=content(get(hObject,'Value'));
if (strcmp(pop_choice,'Default'))
    handles.Vdib='100';
elseif (strcmp(pop_choice,'v5'))
    handles.Vdib='5';
elseif (strcmp(pop_choice,'v10'))
    handles.Vdib='10';
elseif (strcmp(pop_choice,'v20'))
    handles.Vdib='20';
elseif (strcmp(pop_choice,'v30'))
    handles.Vdib='30';
elseif (strcmp(pop_choice,'v50'))
    handles.Vdib='50';
elseif (strcmp(pop_choice,'v80'))
    handles.Vdib='80';
elseif (strcmp(pop_choice,'v100'))
    handles.Vdib='100';
elseif (strcmp(pop_choice,'v200'))
    handles.Vdib='200';
end
guidata(hObject, handles);


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

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

```
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Velbutton.
function Velbutton_Callback(hObject, eventdata, handles)

handles=guidata(hObject);
t = handles.t;

Vpos = handles.Vpos; %Tomamos el valor almacenado en handles, obtenido del desplegable
Vdib = handles.Vdib;

fwrite(t,'calibVEL'); %Enviamos el mensaje de calibración de velocidad
message=fread(t,2);   %Recibimos el OK

vel=[Vpos,',',Vdib]; %Enviamos ambas velocidades
fwrite(t,vel);
guidata(hObject, handles);



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

% Hints: get(hObject,'String') returns contents of Zbase as text
%        str2double(get(hObject,'String')) returns contents of Zbase as a double


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

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


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

Zbase = char(get(handles.Zbase,'string'));


fwrite(t,'Zcalib');
   message=fread(t,2);
```

```
    waitfor(message,'Ok');
    fwrite(t,Zbase);
    poslen = fread(t,[1,2]);
    poslen= char(poslen);
    poslen = str2num(poslen);
    message = fread(t,[1,poslen]);
    message= char(message);
    set(handles.coordenadas,'string',message);
    handles.message;
    drawnow;
    guidata(hObject, handles);
```

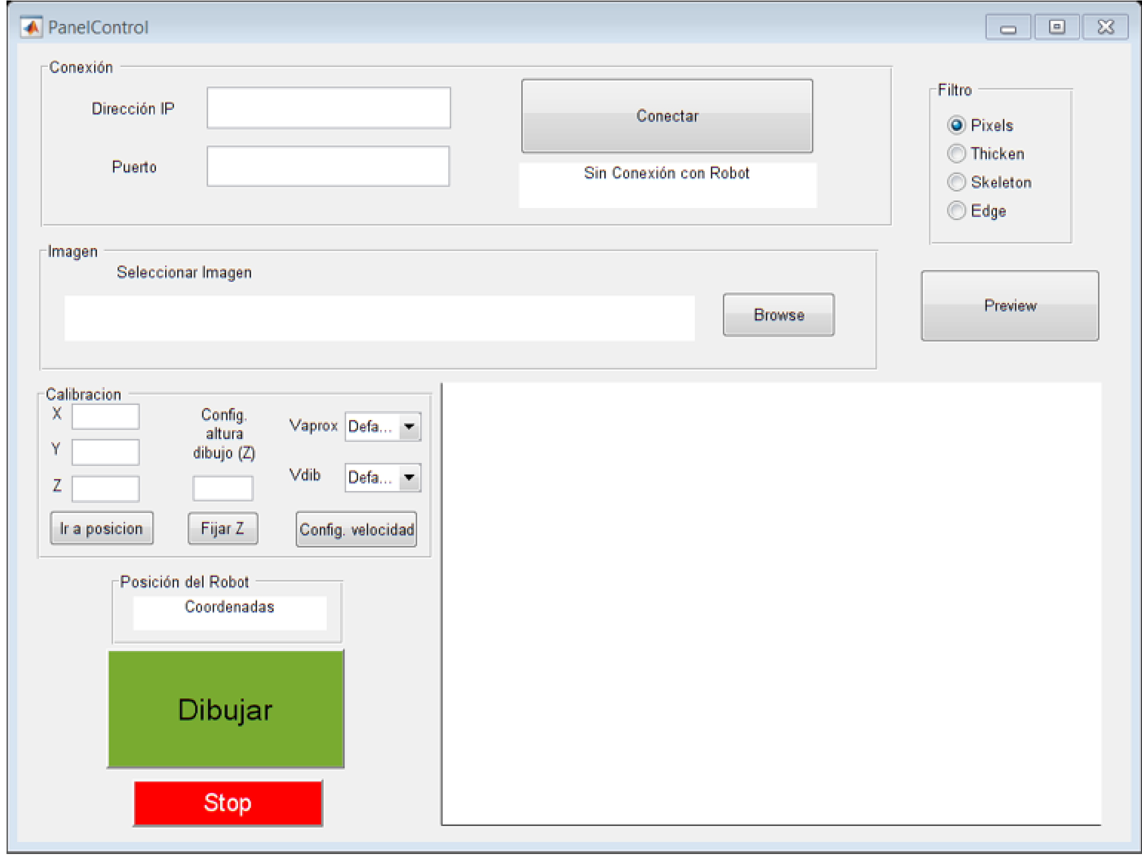


Figura II.1. *Interfaz de Control*

II.2 RobotStudio

II.2.1 Modulo principal 'Mod1'

```
MODULE Mod1

    VAR socketdev server;
    VAR socketdev client;
    VAR string NumberOfPoints;
    VAR string message1;
```

```
    VAR string posstring;
    VAR speeddata Vpos:=v100;
    VAR speeddata Vdib:=v20;
    VAR num Vpos1;
    VAR num Vdib1;
    VAR rawbytes data;
    VAR num Ypos;
    VAR num Xini;
    VAR num Xfin;
    VAR num Xp;
    VAR num Yp;
    VAR num Zp;
    VAR num Zbase;
    VAR num draw;
    VAR pos Position;
    VAR num Pos1;
    VAR num Pos2;
    VAR num Pos3;
    VAR num poslen;
    VAR string poslen2;
    VAR string ipadress:= "127.0.0.1";       !"10.3.17.206";
    VAR num port:=41595;
    VAR bool Ok;

    PROC main()

        Zbase:=-50;

        Movej [[0,0,Zbase-
50],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],v50,fine,tool0\WObj:=Workobject_1;



         socketcreate server;
         socketbind server,ipadress,port;
         socketlisten server;
         socketaccept server,client,\ClientAddress:=ipadress,\Time:=WAIT_MAX;
         Socketsend  client,\str:="Ok";

         Communication;


    ENDPROC
```

II.2.2 Módulo 'Communication'

```
PROC Communication()

       message1:="Inicio";

         while message1<>"stop" DO   !No parar hasta que no reciba el mensaje
'stop'


         SocketReceive client, \str:=message1 \Time:=40; !Recibir mensaje
```

```
draw:=1;

 IF message1="calibVEL" THEN   !Calibrar velocidades

Socketsend  client,\str:="Ok"; !Enviar 'Ok'
SocketReceive client, \str:=message1 \Time:=20;

Pos1:=StrFind(message1,1,",");
Pos2:=StrLen(message1);
Ok:= StrtoVal(StrPart(message1,1,Pos1-1),Vpos1);
Ok:= StrtoVal(StrPart(message1,Pos1+1,Pos2-Pos1),Vdib1);
VelRobot;
draw:=0;

 ELSE

IF message1="Zcalib" THEN   !Calibrar altura de dibujo Z

Socketsend  client,\str:="Ok";
SocketReceive client, \str:=message1 \Time:=20;

Ok:= StrToVal (message1,Zbase);


Movej
[[0,0,Zbase],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,
9E+09,9E+09]],Vpos,z5,tool0\WObj:=Workobject_1;
Movej [[0,0,Zbase-
40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],Vdib,z5,tool0\WObj:=Workobject_1;

SendPosition;

draw:=0;



ELSE

IF message1="calib" THEN    !Mover Robot a punto X,Y,Z

Socketsend  client,\str:="Ok";
SocketReceive client, \str:=message1 \Time:=20;

Pos1:=StrFind(message1,1,",");
Pos2:=StrFind(message1,Pos1+1,",");
Pos3:=StrLen(message1);
Ok:= StrToVal (StrPart(message1,1,Pos1-1),Xp);
Ok:= StrToVal (StrPart(message1,Pos1+1,Pos2-Pos1-1),Yp);
Ok:= StrToVal (StrPart(message1,Pos2+1,Pos3-Pos2),Zp);

Movej
[[Xp,Yp,Zp],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9
E+09,9E+09]],Vpos,z5,tool0\WObj:=Workobject_1;

SendPosition;

draw:=0;


ELSE

IF message1= "next" THEN   !Pasar a dibujar el siguiente trazo
```

```
          MoveJ [[Xp,Yp,Zbase-
40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],Vpos,Z5,tool0\WObj:=Workobject_1;

          Socketsend  client,\str:="Ok";
          SendPosition;

          SocketReceive client, \str:=message1 \Time:=20;

          Pos1:=StrFind(message1,1,",");
          Pos2:=StrLen(message1);
          Ok:= StrToVal (StrPart(message1,1,Pos1-1),Xp);
          Ok:= StrToVal (StrPart(message1,Pos1+1,Pos2-Pos1),Yp);

          MoveJ [[Xp,Yp,Zbase-
40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],Vpos,Z5,tool0\WObj:=Workobject_1;
          MoveL [[Xp,Yp,Zbase-
10],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],Vpos,Z5,tool0\WObj:=Workobject_1;
          MoveL
[[Xp,Yp,Zbase],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+0
9,9E+09,9E+09]],Vdib,fine,tool0\WObj:=Workobject_1;
          Socketsend  client,\str:="Ok";
          SendPosition;
          draw:=0;


          ELSE

          IF message1="fin" THEN

          MoveL [[Xp,Yp,Zbase-
40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],Vpos,Z5,tool0\WObj:=Workobject_1;
          MoveJ [[0,0,Zbase-
40],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+
09]],Vpos,Z5,tool0\WObj:=Workobject_1;

          Socketsend  client,\str:="Ok";

          message1:="stop";
          socketclose server;

          draw:=0;


          ELSE

          IF draw=1 THEN    !Dibujar siguiente punto

          Pos1:=StrFind(message1,1,",");
          Pos2:=StrLen(message1);
          Ok:= StrToVal (StrPart(message1,1,Pos1-1),Xp);
          Ok:= StrToVal (StrPart(message1,Pos1+1,Pos2-Pos1),Yp);
          MoveL
[[Xp,Yp,Zbase],[0.707106781,0,0,0.707106781],[0,0,0,0],[9E+09,9E+09,9E+09,9E+0
9,9E+09,9E+09]],Vdib,Z10,tool0\WObj:=Workobject_1;
          Socketsend  client,\str:="Ok";
          SendPosition;


          ENDIF

          ENDIF
```

```
            ENDIF

            ENDIF

            ENDIF

            ENDIF

            ENDWHILE

    ENDPROC
```

III.2.3 Módulo 'SendPosition'

```
PROC SendPosition()

  Position := CPos(\Tool:=tool0 \WObj:=wobj0); !Obtenemos posición de la
herramienta del robot respecto a centro de coordenadas

   Position.x:=round(Position.x\Dec:=0); !Redondeamos a un decimal la
coordenada x

   Position.y:=round(Position.y\Dec:=0); !Redondeamos a un decimal la
coordenada y

   Position.z:=round(Position.z\Dec:=0); !Redondeamos a un decimal la
coordenada z
         posstring := ValToStr(Position);  !Cambiamos posición a string
         poslen:= StrLen(posstring); !obtenemos la longitud del string
         poslen2:= ValToStr(poslen); !Pasamos la longitud a string
         Socketsend  client,\Str:=poslen2; !Enviamos el string de longitud
   Socketsend  client,\str:=posstring; !Enviamos las coordenadas de posición


    ENDPROC
```

III.2.4 Módulo 'VelRobot'

```
PROC VelRobot ()
        IF Vpos1=5 THEN
             Vpos:=v5;
        ELSEIF Vpos1=10 THEN
             Vpos:=v10;
        ELSEIF Vpos1=20 THEN
             Vpos:=v20;
        ELSEIF Vpos1=30 THEN
             Vpos:=v30;
        ELSEIF Vpos1=50 THEN
             Vpos:=v50;
        ELSEIF Vpos1=80 THEN
             Vpos:=v80;
        ELSEIF Vpos1=100 THEN
             Vpos:=v100;
        ELSEIF Vpos1=200 THEN
             Vpos:=v200;
        ENDIF

        IF Vdib1=5 THEN
             Vdib:=v5;
        ELSEIF Vdib1=10 THEN
```

```
            Vdib:=v10;
        ELSEIF Vdib1=20 THEN
            Vdib:=v20;
        ELSEIF Vdib1=30 THEN
            Vdib:=v30;
        ELSEIF Vdib1=50 THEN
            Vdib:=v50;
        ELSEIF Vdib1=80 THEN
            Vdib:=v80;
        ELSEIF Vdib1=100 THEN
            Vdib:=v100;
        ELSEIF Vdib1=200 THEN
            Vdib:=v200;
        ENDIF

    ENDPROC
ENDMODULE
```