Universidad de Valladolid

UNIVERSIDAD DE VALLADOLID
ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Modelado de una célula robótica con fines educativos usando el programa Robot-Studio

Autor:
Ávila Herrero, Juan Antonio

Tutor:
Herreros López, Alberto
Departamento de Ingeniería de Sistemas y Automática

Valladolid, Septiembre de 2015.

AGRADECIMIENTOS:

En primer lugar a mi tutor, Alberto Herreros López cuyo esfuerzo, superación y ganas de aprender han supuesto todo un ejemplo para mí.

En segundo lugar a mi familia, los que están y los que se fueron, sin los cuales no habría llegado a donde he llegado. No hay palabras suficientes para mostrar mi agradecimiento hacia ellos.

Y por último a mis amigos, presentes en los buenos y malos momentos.

PALABRAS CLAVE:

Simulación Robótica, Robot-Studio, Matlab, OPC, ABB.

RESUMEN:

Se dispone de una célula robótica compuesta por un robot ABB, IRB-120, con una serie de elementos adicionales destinados a la educación. Entre ellos se dispone de una pinza como herramienta para mover objetos, una botonera para señales de entrada-salida, una caja de rotuladores para poder dibujar y dos mesas donde dibujar y mover diferentes elementos. Además, se han añadido sensores externos comunicados con micro-procesador Arduino y la librería OPC de Matlab para poder trabajar en lazo cerrado con el sistema.

El objetivo de este proyecto es simular la célula robótica de que se dispone con el programa Robot-Studio. Dicha simulación constará del robot IRB-120 y de componentes inteligentes programados con Robot-Studio que simulen los diferentes elementos de la célula. Con ello, el alumno podrá realizar las prácticas de robótica en la célula real o en la simulada.

Para probar la célula robótica simulada y su parecido con la real se proponen una serie de prácticas docentes que serán realizadas en ambos entornos comprobando que son equivalentes.

KEYWORDS:

Robotics Simulation, RobotStudio, Matlab, OPC, ABB.

ABSTRACT:

We have an educational robotic cell composed of an ABB robot, the IRB-120, and several additional components. These components are a robotic gripper, an input-output device, several pencils to draw and two tables. We have added external sensors in the system, jointed with the micro-computer Arduino. The communication with the robot is done using the OPC library of Matlab. These sensors will be used to create close-loop systems.

The objective of this project is the simulation of the educational robotic cell using Robot-Studio software. This simulation implies the creation of Robot-Studio intelligent components to simulate the different cell components, and the iteration between them and the IRB-120 robot. With this application, the pupils can develop the same practices with the real robotic cell and the simulated one.

We pose several educational practices with real robotic cell and the simulated one to test the similarities between them.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN Y OBJETIVOS

1.1 INTRODUCCIÓN

La Robótica es una disciplina dedicada al estudio, diseño, realización y manejo de los robots. La robótica combina diversas disciplinas como son: la mecánica, la electrónica, la informática, la inteligencia artificial, la ingeniería de control y la física. Otras áreas importantes en robótica son el álgebra, los autómatas programables, la animatrónica y las máquinas de estados.

Un sistema robotizado engloba todos aquellos dispositivos que realizan tareas de forma automática en sustitución de un ser humano y que pueden incorporar o no a uno o varios robots, siendo esto último lo más frecuente.

En el mundo, el suministro de robots industriales se ha disparado en los últimos años donde China y Japón lideran las estadísticas, tras unos años de estancamiento, España empieza a recuperarse y se ha producido un aumento considerable en las instalaciones de Robots.

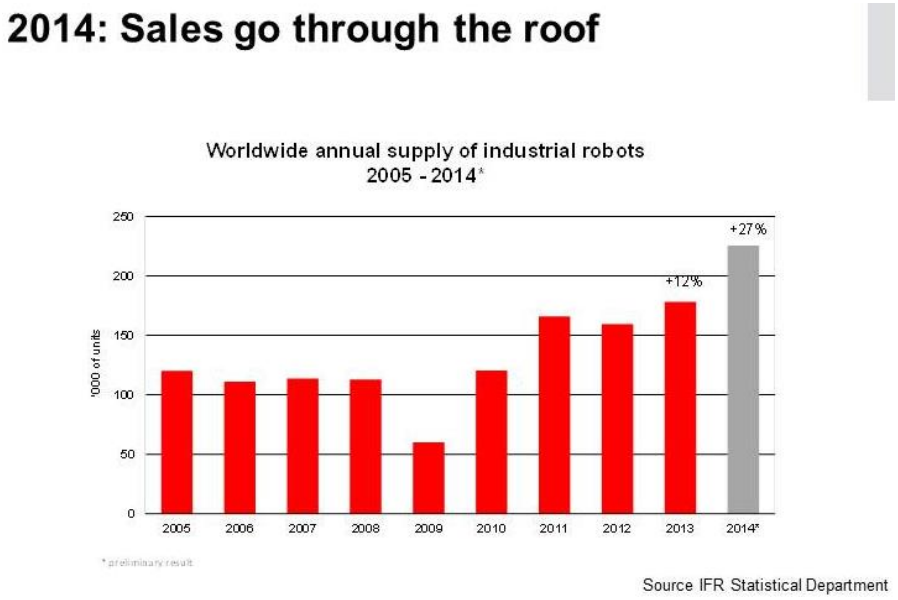


Figura 1.1: Suministro anual mundial de robots industriales

En España, el parque de robots industriales ronda según un estudio las 30.000 unidades y nos posiciona como octava potencia mundial. Casi 19.000 trabajan en el sector automovilístico. Ocupamos además el quinto lugar en densidad de robots por cada 10.000 empleados y en los próximos tres años las inversiones rondarán los 2.500 millones de euros en nuestro país. La clave parece residir en que a pesar de ser el cuarto país europeo en consumo, un mercado que factura solo en robots más de 4.000 millones de dólares al año, 15.000 millones si incluimos el software y la ingeniería, España apenas los fabrica. Los robots españoles solo facturan unos 400 millones al año.

En la actualizar un aspecto muy importante a la hora de desarrollar un sistema robotizado es la programación off-line, la programación off-line es la mejor manera para incrementar la rentabilidad de su sistema robotizado mediante tareas como formación, programación y optimización, sin afectar la producción, lo que proporciona numerosas ventajas, como reducción de riesgos, arranque más rápido, transición más corta e incremento de la productividad.

1.2 JUSTIFICACIÓN DEL PROYECTO

La motivación para la realización de este proyecto atiende a la necesidad de modelar la estación de trabajo del laboratorio de la asignatura de Sistemas Robotizados añadiendo a la estación existente sensores externos comunicados con micro-procesador Arduino y la librería OPC de Matlab para poder trabajar en lazo cerrado con el sistema.

De esta forma, se obtiene el modelo de la estación de trabajo a la que añadiremos sensores externos comunicados con Arduino para que futuros alumnos puedan trabajar en simulación con RobotStudio obteniendo el mismo resultado que si lo hiciesen en la célula real del laboratorio.

1.3 OBJETIVOS

Este trabajo fin de grado tiene como finalidad el modelado de todos los elementos de la estación de trabajo del laboratorio de la asignatura de Sistemas Robotizados mediante el software de simulación RobotStudio, de esta manera la planta real se comportará exactamente igual que la simulación permitiendo a los alumnos de esta asignatura realizar sus programas y pruebas en RobotStudio sin necesidad de utilizar la planta real. A la planta real del laboratorio la añadiremos

sensores externos, por lo que necesitaremos crear un protocolo de comunicación para conectar nuestros sensores a Arduino y comunicarlos con Matlab y Robot-Studio.

Para comprobar el correcto funcionamiento de la estación simulada y comprobar que se comporta como la planta real realizaremos una serie de prácticas didácticas:

- La primera de ellas será jugar con el robot al tradicional juego de las tres en raya y comprobar que el robot se comportar de la misma forma tanto en el entorno real como en el simulado.

- La segunda prueba consistirá en lograr que el robot pueda escribir cualquier texto o dibujar figuras en papel, para ello nos apoyaremos del software matemático Matlab.

- La tercera prueba consiste en utilizar la botonera de entradas y salidas digitales para mover manualmente el robot tanto en simulación como en la planta real.

- En la cuarta aplicación, enviaremos los datos de un sensor de ultrasonidos conectado a Arduino al robot.

- En la última prueba añadiremos un sensor de ultrasonidos a la estación, con este sensor realizaremos un barrido de una superficie y nos permitirá conocer la forma de las piezas que se encuentren en la superficie barrida.

1.4 ESTRUCTURA DE LA MEMORIA

1.4.1 Introducción y objetivos

En este capítulo se presenta una introducción a la temática del proyecto y se tratan tanto la justificación del mismo, como la necesidad de su realización. Asimismo, se detallan los objetivos que se pretenden conseguir en su desarrollo.

Por último se presenta la estructura de la memoria, citando sus partes y explicando brevemente su contenido.

1.4.2 Elementos de la célula robotizada

Aquí se realiza una descripción detallada de la célula robotizada que se pretende modelar en este proyecto, se describe cada uno de sus componentes y se explica el proceso para poder añadir sensores externos a la estación.

1.4.3 Estado del arte

En este apartado se realiza una breve explicación acerca de las tecnologías empleadas en el proyecto como el software de simulación de ABB RobotStudio, la tecnología OPC, la comunicación puerto serie de arduino o Matlab.

1.4.4 Modelado de la célula robotizada

En este capítulo se explica cómo se ha modelado cada uno de los componente de la célula robotizada en Robot-Studio y como se ha realizado las conexiones entre los diferentes elementos para lograr que el modelo se comporte igual que la célula real.

1.4.5 Aplicaciones didácticas

En este capítulo realizaremos una serie de aplicaciones didácticas para verificar el correcto funcionamiento de la estación simulada y pondremos en práctica el trabajo realizado previamente sobre la sensorización de la célula robótica añadiendo sensores externos comunicados con micro-procesador Arduino y la librería OPC de Matlab.

1.4.6 Conclusiones y líneas futuras

Se enumeran las conclusiones obtenidas de este proyecto, tanto en la documentación que ha precedido al desarrollo como en el transcurso del proyecto y se expone las futuras líneas de investigación.

1.4.7 Bibliografía

MEMORIA

2 ELEMENTOS DE LA CÉLULA ROBOTIZADA

La célula robotizada en la que se basa el proyecto se encuentra situada en el laboratorio de la asignatura de Sistemas Robotizados en el sótano de la sede del Paseo del Cauce de la Escuela de Ingenierías Industriales, esta célula robotizada se utiliza para realizar algunas de las prácticas de dicha asignatura permitiendo a los alumnos realizar distintos programas con fines didácticos.

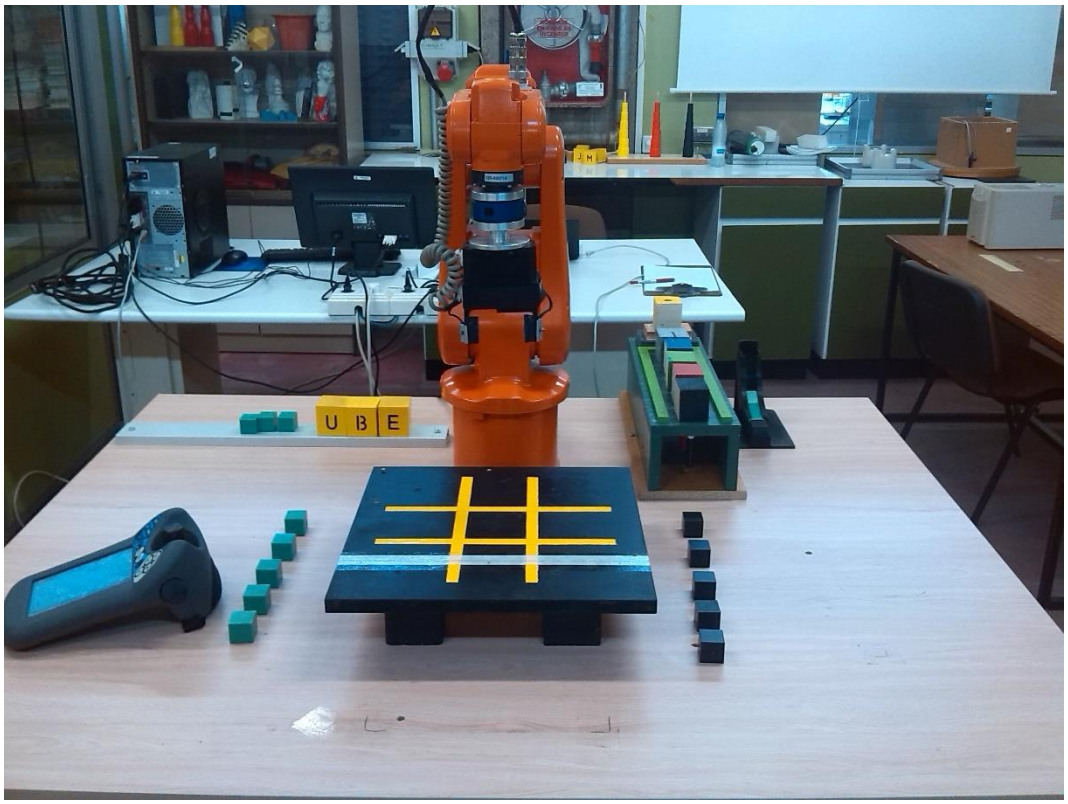


Figura 2.1: Célula robotizada del laboratorio de Sistemas Robotizados

A continuación iremos describiendo cada uno de los elementos con los que cuenta esta estación y a la que hemos añadido sensores externos comunicados con Arduino. Los elementos de la célula son los siguientes:

2.1 ROBOT ABB IRB 120

El laboratorio de sistemas robotizados de la universidad de Valladolid cuenta con un robot ABB IRB-120, es un robot manipulador que permite manipular materiales sin establecer un contacto directo, cuenta con seis grados de libertad (3 en el brazo y 3 en la muñeca), permitiendo que cualquier posición puede ser alcanzada prácticamente con cualquier orientación, es el robot industrial multiusos más pequeño de ABB tan solo pesa 25 kg y puede soportar una carga de 3 kg.

Este robot es usado principalmente para el manejo de pequeños materiales; diseñado principalmente para la industria farmacéutica y electrónica; es decir que para empaquetar medicamentos, jeringas, entre otros; o bien, se aplica para manejar piezas muy pequeñas como en la maquila de transistores, resistencias, carcasas de teléfonos celulares, pantallas de cámaras digitales.

El IRB-120 viene acompañado del controlador IRC5. El controlador ABB IRC5 incluye el control de trayectorias TrueMove, la unidad de programación táctil FlexPendant y el software RobotStudio que permite una programación fuera de línea, la flexibilidad del lenguaje RAPID y potentes capacidades de comunicación entre otras características.

FlexPendant es una unidad de programación muy flexible, basada en el único uso de una pantalla gráfica táctil. La presentación general de FlexPendant se apoya en la tecnología más avanzada de Microsoft, el sistema operativo Windows® CE con su estilo Windows de símbolos en color e interfaz para el operario

Figura 2.2: Robot IRB-120, Controlador IRC5 y Flex Pendant

2.2 PINZA DE AGARRE

La pinza es una herramienta utilizada para tomar un objeto, normalmente la pieza de trabajo, y sujetarlo durante el ciclo de trabajo del robot.

La herramienta utilizada en la célula robótica del laboratorio es la pinza que se muestra en la siguiente figura, su peso es aproximadamente de 2 kg y permite desplazar pequeños objetos con un peso inferior a 1 kg y tengan una anchura por debajo de los 64 milímetros.

Consta de una base fija que la une con el robot, en la parte superior tiene dos eslabones que se desplazan horizontalmente realizando el movimiento de apertura y cierre de la pinza permitiendo coger o soltar los objetos.

Figura 2.3: Pinza de agarre

La pinza tiene un sensor de presión que se activa cuando se ha cogido el objeto a manipular, también cuenta con un sensor laser horizontal entre los dos eslabones superiores para indicarnos la presencia o no de una pieza.

Las señales digitales que tiene la pinza son cuatro:

- Una señal de entrada digital que al activarse abre la pinza.
- Una señal de entrada digital que al activarse cierra la pinza.

- Una señal de salida digital que indica si hay un objeto cortando el sensor laser.
- Una señal de salida digital que indica si la pieza está cogida.

2.3 BOTONERA DE ENTRADAS Y SALIDAS DIGITALES

El controlador IRC5 cuenta con una tarjeta de entradas y salidas digitales situada dentro del controlador, en total nuestro robot cuenta con un total de 16 entradas digitales y 16 salidas digitales alimentadas a 24V.

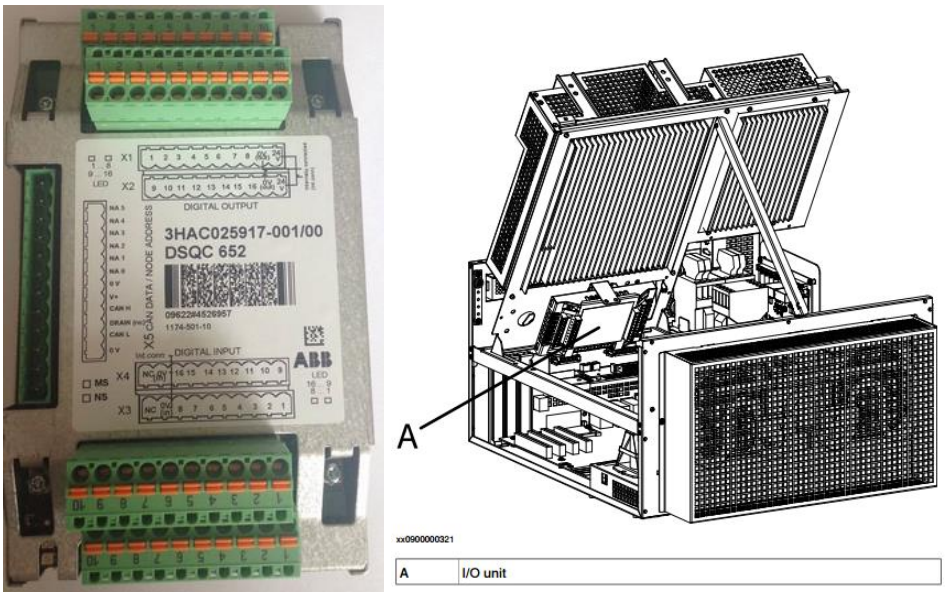


Figura 2.4: Tarjeta de E/S y su localización

La tarjeta de E/S utiliza DeviceNet. DeviceNet es una red digital, multi-punto para conexión entre sensores, actuadores y sistemas de automatización industrial en general. Esta tecnología fue desarrollada para tener máxima flexibilidad entre los equipos de campo e interoperabilidad entre diferentes fabricantes. Sus características principales son: alta velocidad, comunicación a nivel de byte que incluye comunicación con equipos discretos y analógicos y el alto poder de diagnóstico de los dispositivos de la red

El conector XS7 está internamente conectado con la tarjeta de E/S digitales, Un cable conecta las entradas y salidas de la tarjeta con una botonera que cuenta con 16 botones para activar cada una de las entradas digitales y con 16 led unidas a cada una de las señales de salida de tal forma que cuando se active una salida se iluminará el correspondiente led.

Figura 2.5: Botonera de entras y salidas digitales

2.4 AÑADIR SENSORES EXTERNOS A LA ESTACIÓN ROBOTIZADA

Uno de los objetivos de este proyecto es añadir a la célula robotizada existente sensores analógicos externos, el controlador del laboratorio no tiene una tarjeta de entradas y salidas analógicas, solo digitales, por tanto hay que buscar la forma de añadir estos sensores a la estación.

Para solventar este problema comunicaremos los sensores externos analógicos con el micro-procesador Arduino y la librería OPC de Matlab. El esquema sería el que se muestra en la siguiente figura:

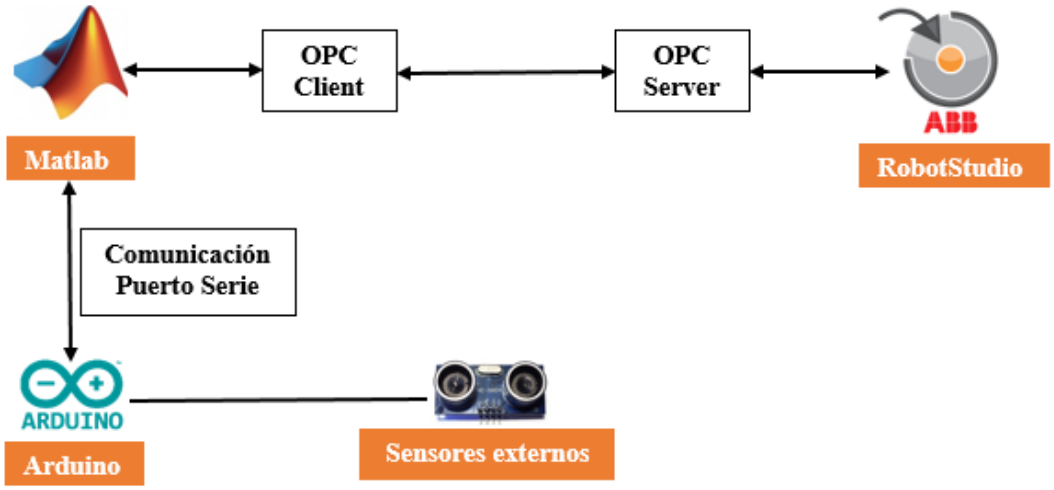


Figura 2.6: Esquema de comunicación para conectar sensores externos

De esta manera se recoge la información de los sensores externos a través de Arduino y se la enviamos a Matlab mediante comunicación serial, posteriormente desde Matlab mandamos la información recibida a RobotStudio mediante comunicación OPC.

2.4.1 Configuración OPC Server

Lo primero que se tiene que hacer es configurar e inicializar el servidor OPC, ABB proporciona el programa "ABB IRC5 OPC Configuration" para poder configurar el servidor de una manera sencilla, hay que tener en cuenta las siguientes consideraciones:

- El robot y el ordenador con al que se comunica deben estar conectado a la misma red.
- ABB IRC5 OPC Server sólo muestra los datos para los robots IRC5 de ABB.
- RobotWare debe de esta correctamente instalado.

Una vez tenidas en cuenta estas consideraciones la forma de configurar el OPC es muy sencilla, en la pestaña Aliases se pulsa el botón "Add new alias"

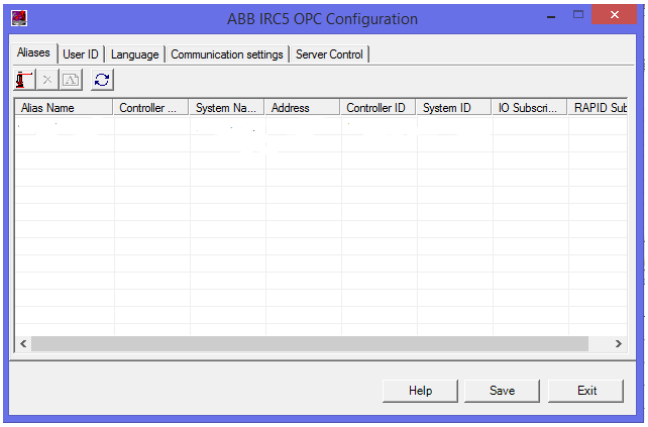


Figura 2.7: Pantalla de configuración del OPC servidor de ABB IRC5

A continuación pulsando el botón scan, buscamos nuestro sistema y creamos el nuevo alias.

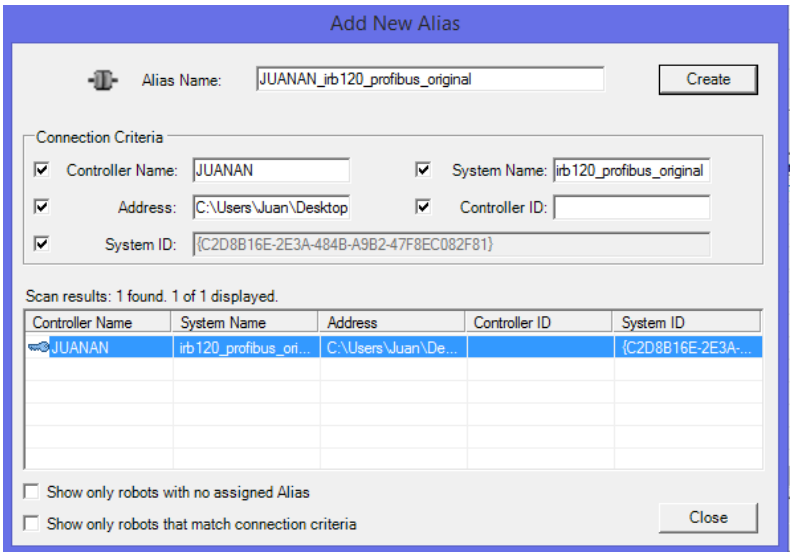


Figura 2.8: Creación de un nuevo Alias

Una vez creado el Alias vamos a la pestaña "Server Control" y se pulsa el botón Start.

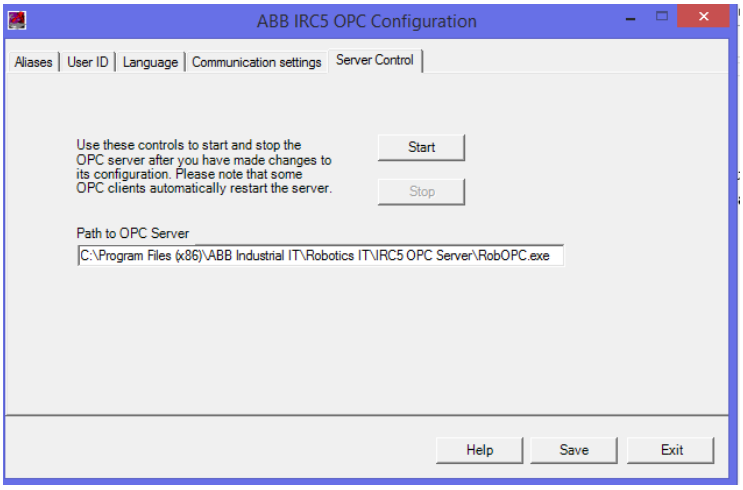


Figura 2.9: Pantalla para iniciar el OPC server

Ya tenemos el OPC Server de ABB funcionando para que desde un OPC Client nos podamos conectar y acceder a la información.

2.4.2 Configuración OPC Client de Matlab

Desde la aplicación OPC cliente de Matlab se puede leer y modificar las variables que se necesiten. No es preciso que la lectura y escritura estén sincronizadas, cuando se modifiquen una variable desde el exterior, dicha variable queda modificada en la estación del robot.

El estándar OPC Data Access define 3 tipos de objetos COM:

• **OPC Server**, es el primer objeto al que se conecta el cliente. Mantiene información sobre el propio servidor, servidores del entorno y contenedores para los objetos OPC Group.

• **OPC Group**, proporciona una manera de organizar los datos. Hay dos tipos: *públicos* (accesibles por todos los clientes), y *locales* (solo accesible por el cliente que lo creo).

• **OPC Item**, representa una conexión con los datos físicos, no son fuentes de datos. No tiene realmente un interfaz hacia los clientes, siendo accesibles tan solo a través de los grupos que lo contienen. Cada ítem lleva asociado el valor, el tipo VARIANT, la calidad y el tiempo de la última actualización, además del identificador ID a través del cual el cliente accede de manera indirecta a los anteriores campos.

Lo primero es conocer que variables son las que queremos conectar entre el controlador IRC5 y Matlab, El código de Matlab para acceder a esas variables (Item) de RobotStudio tanto virtuales como reales es el siguiente:

```
%% Conectarse a OPC Server

%Número se sensores:
N=3;

% Las variables que queremos leer o escribir, deben estar
creadas en el controlador y tener el mismo nombre.
names= {'Sen_1','Sen_2, 'Sen_3'};

% Desconecta y borra todos los objetos de la toolbox de OPC
opcreset;

SenItem= cell(1,N);

% Construimos un objeto de OPC Data Access para el host
% especificado y la ID del OPC server
da= opcda('localhost', 'ABB.IRC5.OPC.Server.DA');

% Conectamos el OPC Data Access
connect(da);

% Añade un grupo al objeto
grp= addgroup(da);

% Añadimos al grupo las tres variables a leer o escribir
 for i= 1:N,
      item= serveritems(da, ['*', names{i}, '*']);
      if length(item)~=1,
          error('Nombre variable erroneo');
      end
      SenItem{i}= additem(grp, item);
 end
```

2.4.3 Conexión serial entre Matlab y Arduino

Con el siguiente código abriremos el Puerto serial en Matlab para poder realizar la comunicación con Arduino, en este ejemplo el puerto que se utiliza es el COM3 pero puede variar dependiendo de cada usuario.

```
%Borrar conexiones previas
delete(instrfind({'Port'},{'COM3'}));

%Crea una conexión serial
puerto_serial=serial('COM3');

%configura la velocidad a 9600 Baudios
puerto_serial.BaudRate=9600;
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');

% Abrimos el puerto serial
fopen(puerto_serial);
```

Si queremos enviar o recibir algún dato desde Arduino las funciones serían las siguientes

```
% Mandamos la información a través del puerto serie
fwrite(puerto_serial,'A');

% Leer del puerto serie
fscanf(puerto_serial);
```

Por último al finalizar el programa hay que cerrar el puerto serie mediante las siguientes funciones

```
% Cerramos el puerto serial
fclose(puerto_serial);

% Eliminamos el puerto serial
delete(puerto_serial);
```

En el código de Arduino la funciones para poder inicializar el puerto serie, enviar y recibir información serían las siguientes:

```
//Configura la transmisión por el puerto serie
void setup(){
Serial.begin(9600);
}
void loop(){

//Imprime por el puerto serie la variable entre paréntesis y
//salta a la línea siguiente
Serial.println();

//Lee del puerto serie
Serial.read();
}
```

2.4.4 Reglas de comunicación para la lectura de sensores externos

Ya hemos visto cómo configurar las distintas conexiones entre Arduino, Matlab y RobotStudio, ahora toca desarrollar unas reglas de comunicación para transmitir la información de los sensores hasta RobotStudio.

Lo primero es conectar Matlab y RobotStudio mediante OPC y Arduino y Matlab mediante Puerto serie:

- Configuramos y arrancamos el OPC Server mediante el programa "ABB IRC5 OPC Configuration"
- Nos conectamos al OPC Server desde el OPC Client de Matlab.
- Abrimos el Puerto Serie en Arduino.
- Abrimos el Puerto Serie en Matlab.

Teniendo ya comunicados Arduino, Matlab y RobotStudio, periódicamente realizamos la lectura y él envió de información de la siguiente forma:

1. Arduino siempre se mantiene a la escucha
2. Desde Matlab se pide a Arduino que envíe la lectura del sensor A
3. Matlab recibe de Arduino la lectura del sensor A
4. Desde Matlab se pide a Arduino que envíe la lectura del sensor B
5. Matlab recibe de Arduino la lectura del sensor B

   …..

   …..

6. Desde Matlab se pide a Arduino que envíe la lectura del sensor N
7. Matlab recibe de Arduino la lectura del sensor N
8. Desde Matlab se mandan todas las lecturas a RobotStudio mediante OPC

Dependiendo de la aplicación que se quiera desarrollar puede que no se quiera enviar la información de los sensores a RobotStudio, quizás se necesite tener la información de los sensores en Matlab y enviar desde RobotStudio ciertas variables a Matlab como puedan ser el Tcp o las posiciones de los ejes.

En el capítulo 5 veremos algunas aplicaciones prácticas de cómo conectar sensores en nuestra estación y transmitir la información de estos.

2.5 MESAS DE TRABAJO

En la célula robotizada existen dos mesas de trabajo:

- **Mesa inclinada**: Es una mesa verde y blanca inclinada, la superficie inclinada sirve como pizarra donde el robot puede dibujar figuras con los distintos rotuladores existentes en el laboratorio.

Figura 2.10: Mesa inclinada

- **Mesa cuadrada negra:** Es una mesa cuadrada de madera de color negra con rayas amarilla cuya función principal es la de ser el soporte de las distintas piezas que el robot manipule.

Figura 2.11: Mesa cuadrada negra

2.6 PIEZAS

En la célula robótica existen diferentes piezas que el robot puede manipular:

- **Dados:** Son un cubo de 28 mm de plástico de diferentes colores que pueden ser fácilmente manipulados por el robot

Figura 2.12: Dado de color verde

- **Rotuladores:** Son varios rotuladores de diferentes colores que sirven para escribir en la pizarra de la mesa inclinada, en la parte superior van unidos mediante un tornillo a un cubo de madera para que puedan ser agarrados por la pinza del robot.

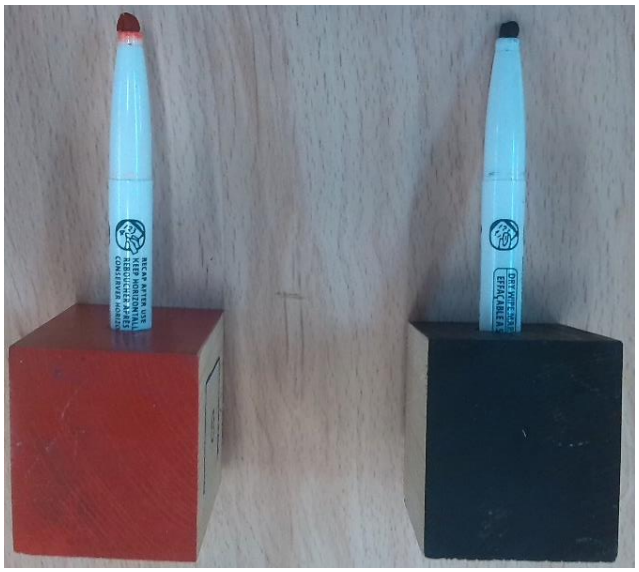


Figura 2.13: Rotuladores de color rojo y negro

- **Soporte de rotuladores:** Los rotuladores descritos anteriormente se encuentran colocados en un soporte de madera con la finalidad de que el robot pueda manipularlos de manera autónoma.

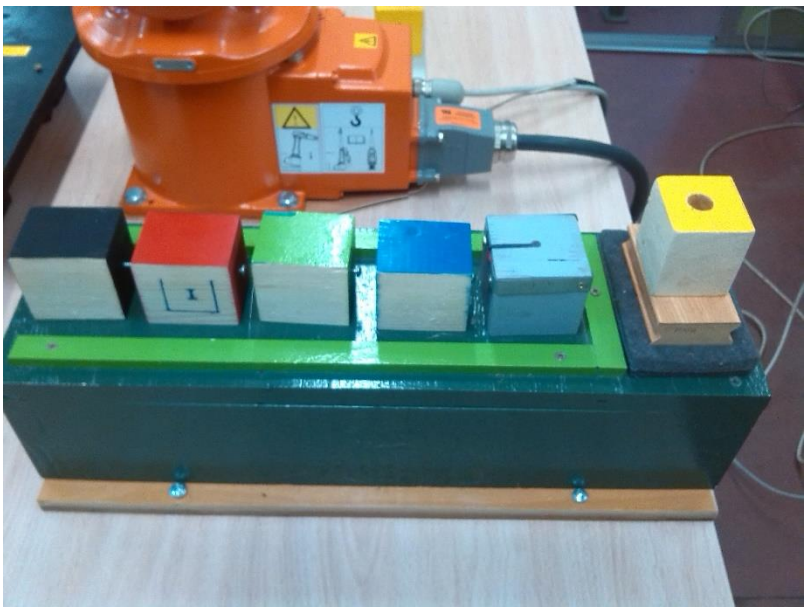


Figura 2.14: Soporte para rotuladores

3 ESTADO DEL ARTE

En este capítulo se van a estudiar y a detallar el estado del arte, es decir, las tecnologías empleadas en este proyecto.

3.1 SOFTWARE DE SIMULACIÓN OFF-LINE ROBOTSTUDIO

Cuando se realiza un estudio completo para la automatización de un proceso industrial es necesario el uso de la simulación del proceso mediante computadores para poder evaluar el sistema una vez diseñado y así evitar los posibles fallos y desajustes con el sistema robótico real montado. El principal motivo para realizar la simulación es principalmente la prevención de fallos y posibles daños en los mecanismos del brazo robótico y en el entorno de producción y así evitar costes derivados de la programación "online".

Hoy en día, las herramientas software, diseñadas en su mayoría por los fabricantes los propios robots industriales, nos permiten analizar el proceso a través de la simulación completa de las células robotizadas mediante computadores.

Para realizar la simulación de la planta del laboratorio usaremos el software de ABB RobotStudio. RobotStudio es un paquete comercial, que trabaja sobre PCs en Windows, de aspecto similar a cualquier producto gráfico de dicho entorno, es una copia exacta del software real que hace funcionar su robot en producción. Ello permite simulaciones muy realistas, con archivos de configuración y programas de robot reales e idénticos a los utilizados en su instalación.

El interfaz que la aplicación ofrece nos permite diseñar sistemas robotizados de una forma muy completa y con cierta facilidad, además de permitir configurar cada uno de los objetos que intervendrán en el proceso de producción. El objetivo final que nos permite alcanzar es la verificación de la operación completa del robot. Podemos detectar fácilmente las colisiones o interferencias producidas entre el robot y el resto de objetos de la célula. Nos permite importar y crear archivos CAD para poder incluir piezas ya existentes o crear unas nuevas en el entorno virtual. Además incluye un "FlexPendant" virtual integrado que simula las consolas reales.

El lenguaje programación empleado por RobotStudio es RAPID (Robotics Application Programming Interactive Dialogue), es muy similar a lenguajes de programación de propósito general de alto nivel (Pascal, C) está estructurado a dos niveles: módulos por una parte y rutinas y funciones por otra. RobotStudio

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

Para simular el comportamiento de los distintos elementos de la célula robotizada RobotStudio ofrece la solución de los componentes inteligentes. Un componente inteligente es un objeto de RobotStudio, con o sin representación gráfica, que presenta el comportamiento que puede o desea implementarse.

También es importante el término "agregación", que es el proceso de conectar varios componentes inteligentes mediante enlazamientos y/o conexiones con el fin de implementar un comportamiento más complejo.

3.1.1 Lenguaje RAPID

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

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

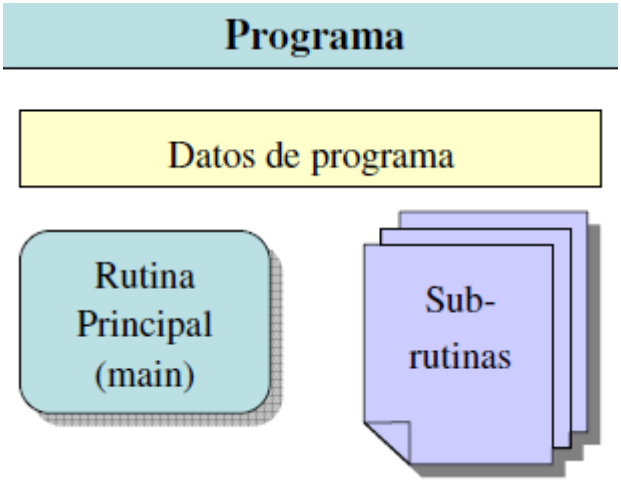


Figura 3.1:Estructura de un programa en RAPID

El programa se estructura en módulos, que son los ficheros donde se escriben las rutinas. Las variables del sistema pueden ser globales a todos los módulos o locales a cada módulo. Con ello se puede obtener unidades

independientes que sólo se comunican por medio de las variables globales, lo que da gran flexibilidad a la hora de programar.

Generalizando, podríamos decir, el programa es una secuencia de instrucciones que controlan el robot y en general consta de tres partes:

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

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

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

Existen tres tipos de rutinas (subprogramas): procedimientos, funciones y rutinas TRAP.

- Los procedimientos no devuelven ningún valor y se utilizan en el contexto de las instrucciones.

- Las funciones devuelven un valor de un tipo concreto y se utilizan en el contexto de las expresiones.

- Las rutinas TRAP proporcionan una forma de responder a las interrupciones. Las rutinas TRAP pueden asociarse con una interrupción determinada. Cuando posteriormente se produce una interrupción concreta, se ejecuta la rutina TRAP correspondiente. Las rutinas TRAP no pueden ejecutarse explícitamente desde el programa.

La información también puede almacenarse en datos, por ejemplo los datos de las herramientas (que contienen información sobre las herramientas, como su TCP y su peso) y datos numéricos (que pueden usarse por ejemplo para contar el número de piezas que deben procesarse). Los datos se agrupan en distintos tipos de datos que describen los distintos tipos de información, como herramientas, posiciones y cargas. Dado que es posible crear estos datos y asignarles nombres arbitrarios, no existe ningún límite en el número de datos (excepto el límite impuesto por la memoria disponible). Estos datos pueden existir de forma global en el programa o sólo localmente dentro de una rutina.

Existen tres tipos de datos: constantes, variables y persistentes.

- Las constantes representan valores fijos y la única forma de asignarles un nuevo valor es manualmente.

- La asignación de nuevos valores a las variables puede realizarse durante la ejecución del programa.

- Un valor persistente puede describirse como una variable "persistente". Cuando se guarda un programa, el valor de inicialización corresponde al valor actual del valor persistente.

3.2 TECNOLOGÍA OPC

OPC (OLE for Process Control) es un estándar de comunicación en el campo del control y supervisión de procesos industriales, basado en una tecnología Microsoft, que ofrece una interfaz común para comunicación que permite que componentes software individuales interactúen y compartan datos. La comunicación OPC se realiza a través de una arquitectura Cliente-servidor. El servidor OPC es la fuente de datos (como un dispositivo hardware a nivel de planta) y cualquier aplicación basada en OPC puede acceder a dicho servidor para leer/escribir cualquier variable que ofrezca el servidor. Es una solución abierta y flexible al clásico problema de los drivers propietarios. Prácticamente todos los mayores fabricantes de sistemas de control, instrumentación y de procesos han incluido OPC en sus productos.

El propósito de OPC es tener una infraestructura estándar para el intercambio de datos de control de procesos. Es típico tener varias fuentes de información en el proceso, las cuales están contenidas en distintos dispositivos tales como controladores programables, medidores, unidades de transferencia remotas, sistemas de control centralizados, base de datos, etc. Antiguamente estos dispositivos sólo intercambiaban datos con aplicaciones provistas por el mismo fabricante, lo que representaba muchas restricciones. Sin embargo, gracias a OPC, hoy podemos intercambiar libre y fácilmente información desde estos dispositivos y aplicaciones de cualquier tipo, como por ejemplo soluciones de HMI (Human Machine Interface), planillas de cálculo, motores de base de datos, ERPs, entre otras.

Aunque OPC ofrece más ventajas que desventajas, estas últimas pueden ser muy importantes y deben ser tenidas en cuenta.

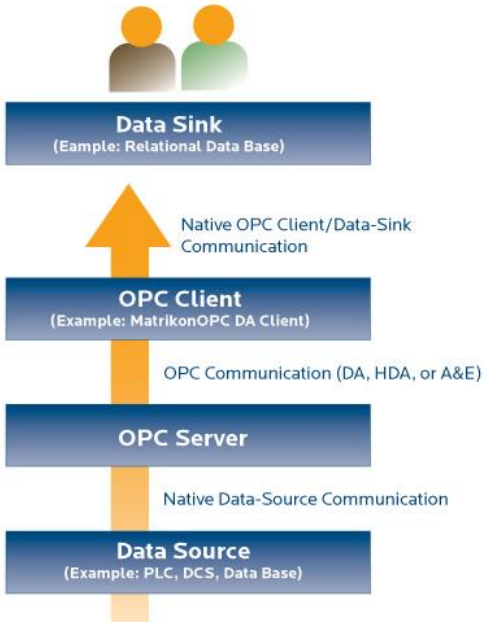


Figura 3.2: Ejemplo de tenología OPC

Ventajas de OPC

- OPC permite la comunicación de plataformas no industriales, como MS Office, logrando realizar soluciones costo-efectivas a procesos particulares.
- Existe una gran variedad de servidores OPC para todas las marcas y estándares, permitiendo elegir el más adecuado para las necesidades o conocimientos de cada uno.
- Los fabricantes de hardware sólo tienen que hacer un conjunto de componentes de programa para que los clientes los utilicen en sus aplicaciones.
- Los fabricantes de software no tienen que adaptar los drivers ante cambios de hardware.

Desventajas de OPC

- Es una solución de software, con lo que el desempeño en términos de tiempo de respuesta y fiabilidad nunca son los mejores.

- El uso de un servidor OPC básico puede ser muy sencillo, pero generalmente son los que tienen menores prestaciones. Los OPCs de calidad industrial (que pueden dar respuestas casi en tiempo real) demandan procedimientos de configuración más engorrosos.
- Muchas veces, utilizar OPC es más caro que adquirir un SCADA con los drivers apropiados integrados. La tentación de desarrollar un SCADA propio basado en OPC, puede ahorrar los costos de licencias de paquetes específicos de desarrollo, pero conviene prestar atención a los costos por horas de ingeniería en un producto final no estándar.

Una aplicación OPC, como cualquier otra aplicación OLE (o DDE), constará de servidores y clientes OPC. Cada cliente, es decir, cada aplicación de usuario, SCADA, módulo histórico, o aplicación de usuario interroga al servidor que contiene los datos que necesita.

Los servidores están organizados en grupos y cada grupo puede contener distintos ítems. Las diferentes partes de la aplicación pueden usar distintos grupos, los cuales pueden tener distinta frecuencia de refresco y pueden ser de acceso secuencial o basado en excepciones (eventos). Los ítems representan conexiones a fuentes de datos dentro del servidor (variables de proceso). A cada ítem se asocia un valor (valor de la variable de proceso), un cualificador (estado de la variable, OK, bajo rango, etc.) y una marca de tiempo.

sistema abierto **Figura 3.3: Organización de objetos en un servidor OPC**

La idea principal, sobre la que se construye todo el sistema, es el aprovechamiento del estándar OPC, como, para asegurar la normalización de nuestro sistema y su aprovechamiento para un amplio rango de software de control (todos aquellos que compartan interfaz como clientes OPC).

OPC nos permite establecer una comunicación estandarizada entre clientes y servidores OPC. Todos los estándares OPC deben tener un conjunto básico de interfaces, de manera que cualquier cliente desarrollado bajo este estándar se pueda comunicar con servidores de distintos fabricantes sin problemas. Esto se consigue gracias a estar desarrollado sobre las librerías COM (Component Object Model).

COM es una tecnología desarrollada por Microsoft basada en una interacción cliente-servidor, con el objetivo de la reutilización de software. Un software que soporte COM podrá ser reutilizado desde otra aplicación a través de una serie de funciones o interfaces. Como extensión de COM para su uso a través de una red es DistributedCOM (DCOM).

COM es un estándar binario que permite el desarrollo de clientes y servidores sobre Visual Basic y C/C++.

COM está basado en objetos. Entendemos por objeto COM, a una pieza de código, perfectamente encapsulada en Componentes/Servidores, de manera que tan solo son accesibles a través de interfaces. La comunicación entre los objetos y los clientes COM se hará tan solo utilizando estos interfaces. La definición de los interfaces se debe hacer en lenguaje IDL (Interface Definition Language), pero la implementación de los métodos de este se puede hacer en cualquier lenguaje de programación.

Las especificaciones OPC definen 3 tipos de servidores OPC:

- **Data Access**, define interfaces para la lectura y escritura de datos en el sistema de control.

- **Alarms and Events**, permite mantener vigilancia sobre posibles condiciones anormales en el sistema.

- **Historical Data Access**, mantiene una base de datos sobre un servidor OPC DA.

El estándar OPC Data Access define 3 tipos de objetos COM:

- **OPC Server**, es el primer objeto al que se conecta el cliente. Mantiene información sobre el propio servidor, servidores del entorno y contenedores para los objetos OPC Group.

- **OPC Group**, proporciona una manera de organizar los datos. Hay dos tipos: *públicos* (accesibles por todos los clientes), y *locales* (solo accesible por el cliente que lo creo). En nuestro caso el objetivo del cliente no será crear grupos, será el propio servidor el encargado de crear grupos públicos.

• **OPC Item**, representa una conexión con los datos físicos, no son fuentes de datos. No tiene realmente un interfaz hacia los clientes, siendo accesibles tan solo a través de los grupos que lo contienen. Cada ítem lleva asociado el valor, el tipo VARIANT, la calidad y el tiempo de la última actualización, además del identificador ID a través del cual el cliente accede de manera indirecta a los anteriores campos.

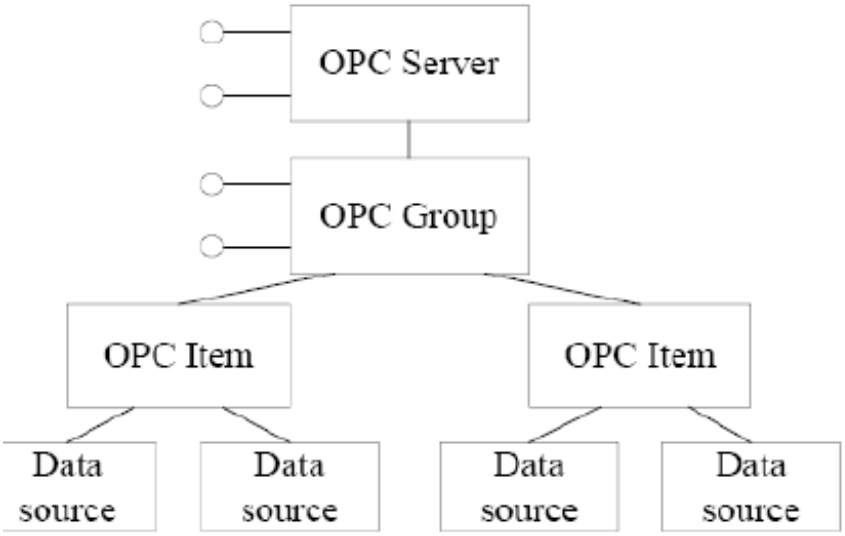


Figura 3.4: Estructura de objetos OPC

Todos estos valores se actualizan también en la cache, para su lectura por la propia fuente emisora y receptora de señales físicas.

El fabricante ABB cuenta con un servidor OPC que puede ser ejecutado desde un PC. Esta aplicación llamada "ABB IRC5 OPC Configuration" muestra los datos para los robots IRC5 de ABB conectados a una misma red en cada periodo de muestreo fijado. Desde una aplicación OPC cliente se puede leer y modificar dichas variables. No es preciso que la lectura y escritura estén sincronizadas, cuando se modifique una variable desde el exterior, dicha variable queda modificada en la estación del robot.

3.3 MATLAB

MATLAB (Matriz Laboratory) es el lenguaje de alto nivel y el entorno interactivo utilizado por millones de ingenieros y científicos. Permite explorar y visualizar ideas, así como colaborar interdisciplinarmente en procesamiento de señales e imagen, comunicaciones, sistemas de control y finanzas computacionales. Matlab presenta un entorno de programación para computación técnica basado en el modelo matemático de cálculos sobre matrices. El sistema Matlab consta de:

• **Lenguaje Matlab**, nos permite manipular datos en forma de escalares, vectores, matrices, etc. El mismo lenguaje se usa tanto en el espacio de trabajo como en el desarrollo de programas (*M-file*).

• **Espacio de trabajo de Matlab**, en él se realizan los cálculos y se almacenan los datos, para su intercambio y manipulación.

• **Entono gráfico**, nos permite la visualización de datos en 2 y 3 dimensiones, así como animaciones. Su uso se apoya el procesamiento matemático tanto desde el espacio de trabajo como de programas M-file o toolboxes que lo precisen.

• **Funciones de librería**, funciones que nos permiten un gran número de operaciones útiles. Utilice la orden "help" para comprender la utilidad y el resultado de cada una.

• **Matlab Application Program Interface**, nos permite extender el uso de las facilidades ofrecidas por Matlab a otro tipo de entornos de programación, permitiendo la *comunicación con otras aplicaciones.*

• **Toolboxes de Matlab**, herramientas en forma de aplicaciones ejecutables desde Matlab y apoyadas en la potencia de cálculo del propio lenguaje Matlab, pero facilitando el interfaz con el usuario o incrementando la potencia del sistema Matlab.

3.4 ARDUINO

Arduino es una plataforma de prototipos electrónica de código abierto (open-souce) basada en hardware y software flexibles y fáciles de usar. El hardware consiste en una placa con un microcontrolador Atmel y puertos de entrada y salida.

Arduino puede tomar información del entorno a través de sus entradas analógicas y digitales, puede controlar luces, motores y otros actuadores. El microcontrolador en la placa Arduino se programa mediante el lenguaje de programación Arduino (basado en Wiring) y el entorno de desarrollo Arduino

(basado en Processing). Los proyectos hechos con Arduino pueden ejecutarse sin necesidad de conectar a un ordenador.

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones.

```
void setup()
{
        Instrucciones;
}
void loop()
{
        Instrucciones;
}
```

En donde setup() es la parte encargada de recoger la configuración y loop() es la que contienen el programa que se ejecutará cíclicamente (de ahí el termino loop –bucle-). Ambas funciones son necesarias para que el programa trabaje.

Algunas de las ventajas que Arduino ofrece son:

- **Barato:** Arduino al ser hardware de código abierto es más baratas que otras plataformas microcontroladoras.

- **Multiplataforma:** El software de Arduino se puede ejecutar en diversos sitemas operativos como Windows, Linux.

- **Open-source:** Arduino es una plataforma de código y hardware abierto, es decir, puedes acceder a todo aspecto del funcionamiento circuital y algorítmico de las placas, y mucho mejor que eso, te dan todos los archivos Eagle, diagramas y componentes para que tú mismo crees tú versión de Arduino.

- **Shields y periféricos:** Arduino también ofrece una gran gama de Shields o placas extras que cumplen funcionalidades específicas como Ethernet, GSM, Control de Reles, Wi-fi y pueden ser acopladas a las tarjetas de forma sencilla, aumentando considerablemente el rango de aplicaciones disponibles.

3.4.1 Comunicación puerto serie en arduino

Un puerto es el nombre genérico con que denominamos a los interfaces, físicos o virtuales, que permiten la comunicación entre dos ordenadores o dispositivos.

Un puerto serie envía la información mediante una secuencia de bits. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos, RX (recepción) y TX (transmisión). No obstante, pueden existir otros conductores para referencia de tensión, sincronismo de reloj, etc.

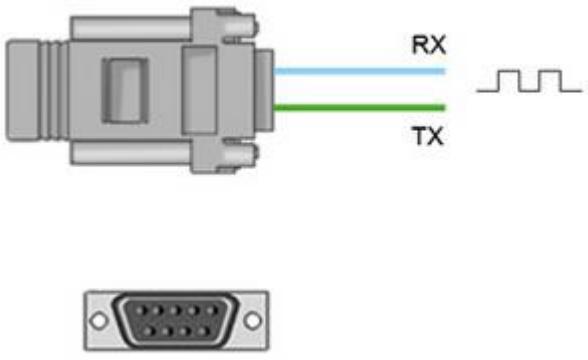


Figura 3.5: Comunicación serie

El puerto serie del Arduino Uno usa los pins 0(RX) y 1(TX). Estos están conectados al controlador FTDI que es el que permite la traducción del formato serie TTL a USB. Estos pins no pueden ser utilizados mientras se usa la comunicación serie. La comunicación se realiza mediante variaciones en la señal entre 0V y Vcc (donde Vcc suele ser 3.3V o 5V).

Muchos modelos de placas Arduino disponen de un conector USB o Micro USB conectado a uno de los puertos de serie, lo que simplifica el proceso de conexión con un ordenador. Sin embargo algunas placas, como por ejemplo la Mini Pro, prescinden de este conector por lo que la única forma de conectarse a las mismas es directamente a través de los pines correspondientes.

El puerto serie del Arduino utiliza un buffer de 64 bytes.

Las funciones relacionadas con el puerto serie en Arduino son las siguientes:

- **Serial.begin(rate).** Abre un Puerto serie y especifica la velocidad de transmisión. La velocidad típica para comunicación con el ordenador es de 9600 aunque se pueden soportar otras velocidades.

- **Serial.println(data).** Imprime datos al puerto serie seguido por un retorno de línea automático. Este comando tiene la misma forma que ´serial.print() pero este último sin el salto de línea al final. Este comando puede emplearse para realizar la depuración de programas. Para ello puede mandarse mensajes de depuración y valores de variables por el puerto serie. Posteriormente, desde el entorno de programación de Arduino, activando el "Serial Monitor" se puede observar el contenido del puerto serie, y, por lo tanto, lso mensajes de depuración. Para observar correctamente el contenido del puerto serie se debe tener en cuenta que el "Serial Monitor" y el puerto serie han de estar configurados a la misma velocidad.

- **Serial.read().** Lee o captura un byte (un carácter) desde el puerto serie. Devuelve -1 si no hay ningún carácter en el puerto serie.

- **Serial.available().** Devuelve el número de caracteres disponibles para leer desde el puerto serie.

4 MODELADO DE LA CÉLULA ROBOTIZADA

4.1 INTRODUCCIÓN

El modelado de la estación de trabajo del laboratorio de sistemas robotizados es el primer paso para poder trabajar off-line, una vez modelada, cualquier persona podrá trabajar off-line en la estación de trabajo realizando sus propios programas y pruebas sin poner en riesgo la estación de trabajo real y con la seguridad de que la estación de trabajo real y la simulada se comportan exactamente de la misma manera.

Como hemos mencionado anteriormente la estación de trabajo del laboratorio cuenta con los siguientes elementos que modelaremos:

- **Robot ABB IRB-120**
- **Pinza**:
- **Mesa blanca inclinada**
- **Mesa negra**
- **Porta rotuladores**
- **Rotuladores de distintos colores**
- **Piezas**
- **Botonera de E/S:**

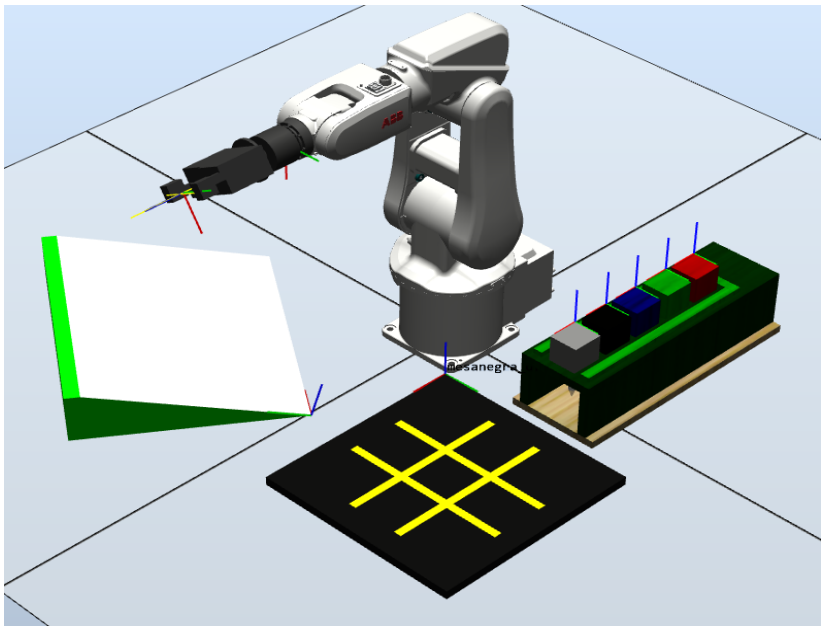


Figura 4.1: Modelado de la estación de trabajo del laboratorio de Sistemas Robotizados

4.2 MODELADO DE LA PINZA

La pinza es una herramienta utilizada para tomar un objeto, normalmente la pieza de trabajo, y sujetarlo durante el ciclo de trabajo del robot.

4.2.1 Creación de la geometría de la pinza

RobotStudio permite importar modelos CAD, si el fabricante nos proporcionase el modelo CAD de la pinza podríamos trabajar con ella simplemente importándola, este no es el caso y hay que modelar la pinza desde cero, esto se puede hacer mediante cualquiera de los programas de modelado que hay en el mercado y posteriormente importar la pieza a RobotStudio o bien modelar la pieza directamente en RobotStudio, esta última opción es la que se ha elegido.

La pestaña Modelado de RobotStudio permite crear la geometría de la pinza, la pinza está compuesta por 11 Piezas simples (cilindros y prismas rectangulares) conectados entre sí para obtener la forma deseada.

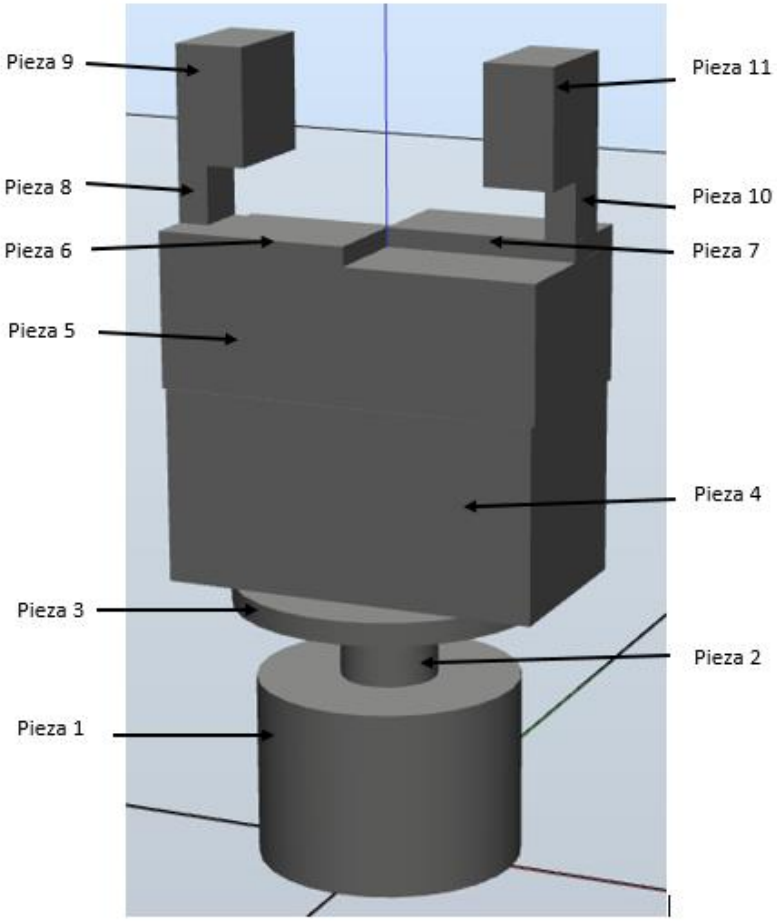


Figura 4.2: Geometría de la pinza

4.2.2 Creación de la herramienta a partir de la geometría

Una vez que se ha creado las geometrías de la pinza en la pestaña de modelado, RobotStudio nos permite crear un mecanismo, para el caso que nos ocupa, el tipo de mecanismo es el de una herramienta.

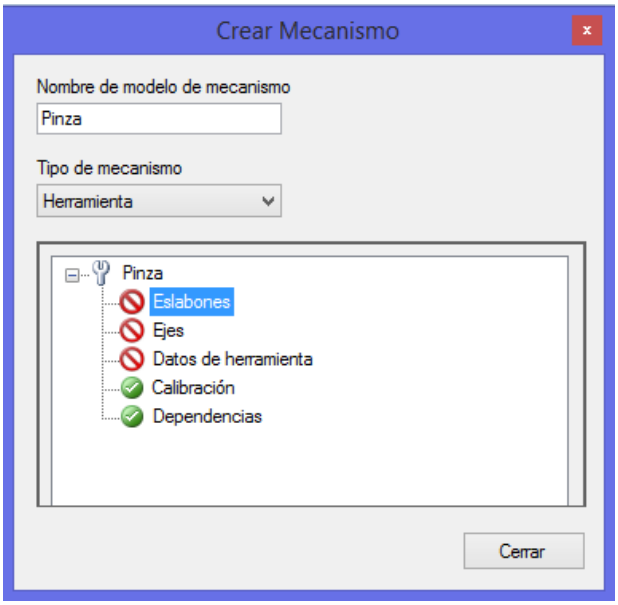


Figura 4.3: Ventana de creación de mecanismos en RobotStudio

La pinza consta de tres eslabones:

- Eslabón 1 (L1): Es el eslabón base y consta de las piezas 1, 2, 3, 4 y 5.
- Eslabón 2 (L2): Este eslabón consta de las piezas 6, 7 y 8.
- Eslabón 3 (L3): Este eslabón consta de las piezas 9, 10 y 11.

A continuación definimos los ejes, la pinza cuenta con dos ejes:

- Eje 1 (J1): Es un eje de tipo prismático que permite el movimiento lineal entre el eslabón principal (L1) y el eslabón secundario (L2).

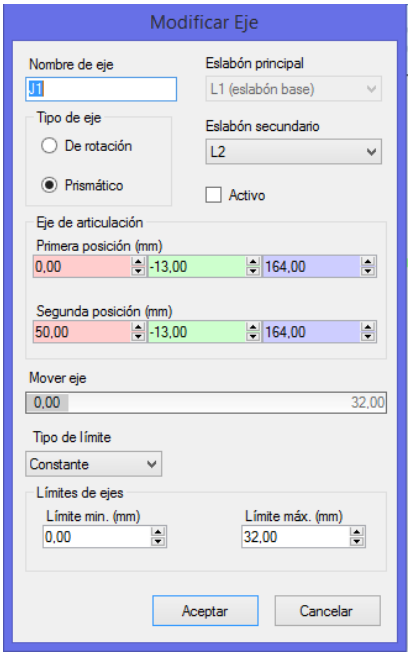


Figura 4.4: Definición de uno de los ejes de la herramienta

- Eje 2 (J2): Es un eje de tipo prismático que permite el movimiento lineal entre el eslabón principal (L1) y el eslabón secundario (L3).

Los datos de la herramienta contiene la masa de nuestra pinza (2.2 kg), el momento de inercia y el centro de gravedad.

Por ultimo creamos una dependencia entre el eje 1 y el eje 2 para provocar que los dos eslabones secundarios L2 y L3 se muevan a la vez.

Una vez definidos todos los parámetros de nuestra pinza la compilamos y ya tenemos la herramienta lista para usar.

4.2.3 Dotar a la pinza de componentes inteligentes

Por si sola la pinza no se abrirá ni cerrará durante la simulación, hay que añadir las señales, además, la pinza tendrá que detenerse si coge un objeto, transportarlo y soltarlo, todas estas acciones se consiguen con lo que RobotStudio llama componentes inteligentes

El Editor de componentes inteligentes permite crear, editar y agregar componentes inteligentes mediante una interfaz gráfica de usuario.

Lo primero que definimos en nuestro componente inteligente son las señales de E/S que tiene la pinza modelada, las señales son las siguientes:

Nombre de la señal	Tipo de la señal	Descripción
Abrir	Entrada digital	Señal que al estar en nivel alto provoca la apertura de la pinza
Cerrar	Entrada digital	Señal que al estar en nivel alto provoca el cierre de la pinza
Activar_sensor_lineal	Entrada digital	Señal requerida por RobotStudio para activar el componente inteligente, siempre está en nivel alto
Pieza_cogida	Salida digital	Cuando la pinza coge un objeto esta señal se pone en nivel alto
Pieza_detectada	Salida digital	Cuando la pinza detecta un objeto por su sensor lineal laser esta señal se pone en nivel alto

Tabla 4-1: Señales de E/S de la pinza modelada

Un componente inteligente esta compuestos por la unión de los componentes básicos que vienen en RobotStudio. Los componentes básicos constituyen un conjunto completo de componentes modulares básicos

Los componentes utilizados en la pinza modelada son los siguientes:

- **Attacher:** conectará el objeto Child a Parent cuando se activa la señal Execute. Si Parent es un mecanismo, también es necesario especificar la brida Flange a la que conectarse. Si la entrada Execute está activada, el objeto subordinado se conecta al objeto superior. Si Mount está activado, el objeto subordinado también se montará sobre el objeto superior, con los parámetros Offset y Orientation especificados. La salida Executed se activa al finalizar.

- **JointMover:** utiliza como propiedades un mecanismo, un conjunto de valores de eje y una duración. Cuando se activa la señal de entrada Execute, los valores de eje del mecanismo se mueven hasta la pose indicada. Una vez alcanzada la pose, se activa la señal de salida Executed. La señal GetCurrent obtiene los valores de eje actuales del mecanismo.

- **LogicSRLatch** – Mantiene estable el pulso de entrada.

- **Detacher:** desconectará el objeto Child del objeto cuando se activa la señal Execute. Si Keep position está activado, la posición se mantendrá. De lo contrario, el objeto subordinado se posiciona con respecto a su objeto superior. Al finalizar, la señal Executed se activa.

- **CollisionSensor:** detecta colisiones y eventos de casi colisión entre el primer objeto y el segundo. Si no se especifica uno de los objetos, el otro se comprueba frente a toda la estación. Si la señal Active está en el nivel alto y se produce una colisión o un evento de casi colisión y el componente está activo, la señal SensorOut se activa y las piezas implicadas en la colisión o en el evento de casi colisión se indican como primera pieza en colisión y segunda pieza en colisión del Editor de propiedades.

- **Positioner:** toma un objeto, una posición y una orientación como propiedades. Cuando se activa la señal Execute, el objeto es reposicionado en la posición determinada con respecto a Reference. Al finalizar, se activa la salida Executed.

- **LineSensor:** define una línea por sus parámetros Start, End y Radius. Cuando una señal Active tiene el valor alto, el sensor detecta los objetos que están en intersección con la línea. Los objetos que están en intersección se muestran en la propiedad ClosestPart y el punto de la pieza en intersección más cercana al punto inicial del sensor de línea se muestra en la propiedad ClosestPoint. Cuando se produce la intersección, se activa la señal de salida SensorOut.

- **LogicGate** La señal Output es activada por la operación lógica especificada Operator en las dos señales InputA y InputB, con el retardo especificado en Delay.

- **PositionSensor:** monitoriza la posición y orientación de un objeto.

- **VectorConverter:** Convierte entre Vector3 y valores X, Y y Z.

Cada uno de estos componentes básicos tiene que tener definidas unas propiedades, las propiedades de los componentes básicos de la pinza modelada son los siguientes:

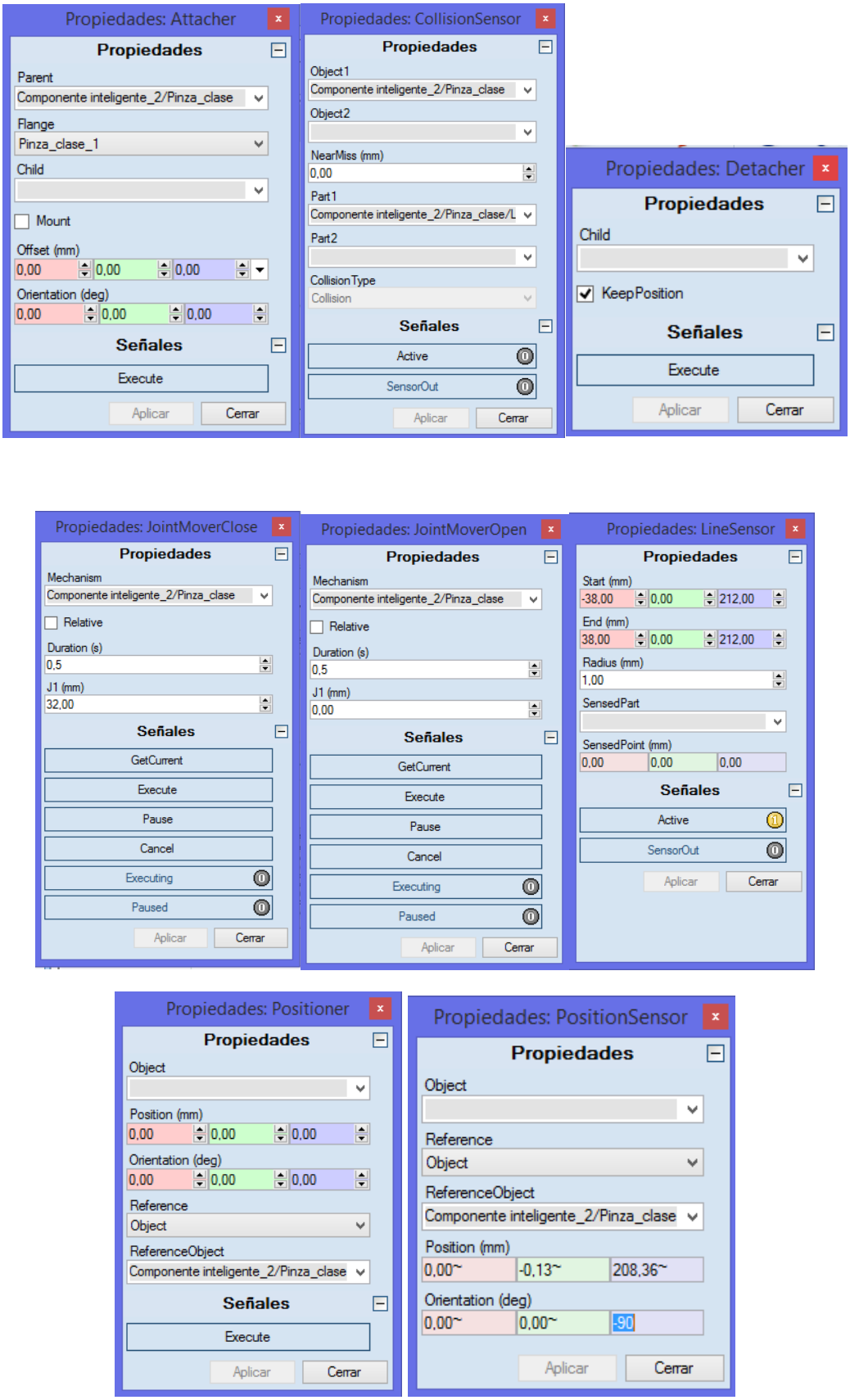


Figura 4.5: Propiedades de los componentes básicos de la pinza

Las conexiones entre estos componentes básicos es la siguiente:

Objeto de origen	Señal de origen	Objeto de destino	Señal de destino
Componente_inteligente	Abrir	JoinMoverOpen	Execute
Componente_inteligente	Abrir	Detacher	Execute
Componente_inteligente	Cerrar	CollisionSensor	Active
CollisionSensor	SensorOut	Attacher	Execute
CollisionSensor	SensorOut	JoinMoverClose	Cancel
LineSensor	SensorOut	Componente_inteligente	Pieza_detectada
Componente_inteligente	Cerrar	JoinMoverClose	Execute
Attacher	Execute	LogicSRLatch	Set
LogicSRLatch	Output	Componente_inteligente	Pieza_cogida
Componente_inteligente	Activar_sensor_lineal	LineSensor	Active
Componente_inteligente	Cerrar	LogicGate [AND]	InputA
LineSensor	SensorOut	LogicGate [AND]	InputB
LogicGate [AND]	Output	Positioner	Execute
Detacher	Execute	LogicSRLatch	Reset

Tabla 4-2: Tabla de conexión entre componentes básicos

El diagrama de flujo simplificado que simula el cierre de la pinza es el siguiente:

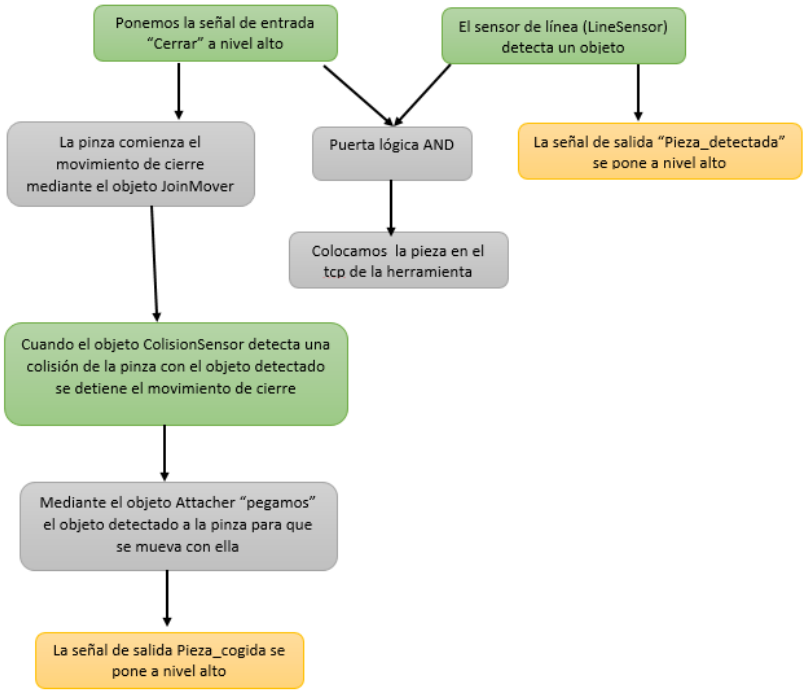


Figura 4.6: Diagrama de flujo del cierre de la pinza

El diagrama de flujo simplificado que simula la apertura de la pinza es el siguiente:

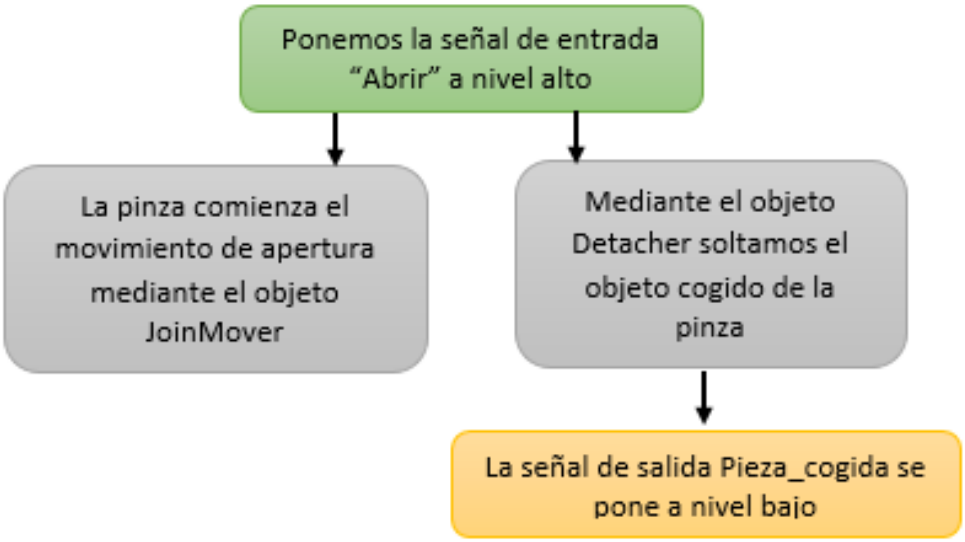


Figura 4.7: Diagrama de flujo de la apertura de la pinza

Una vez que tenemos el modelo de la pinza ya completado solo quedaría conectar las señales de E/S de la pinza con las E/S del controlador del robot IRB-120.

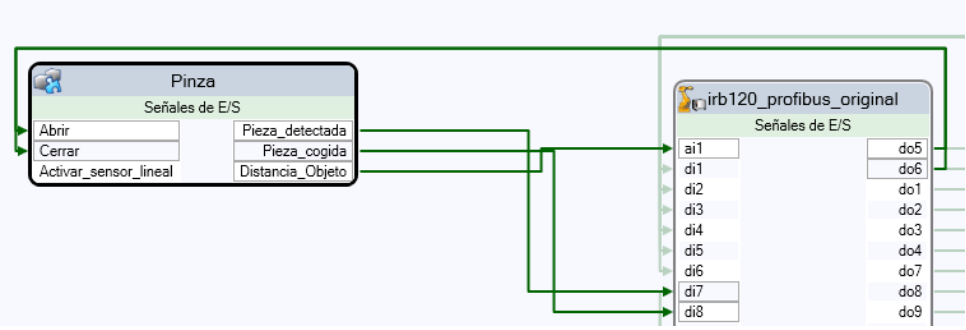


Figura 4.8: Esquema de conexiones de la pinza

Objeto de origen	Señal de origen	Objeto de destino	Señal de destino
Irb120_profibus_original	do5	Pinza	Abrir
Irb120_profibus_original	do6	Pinza	Cerrar
Pinza	Pieza_detectada	Irb120_profibus_original	di7
Pinza	Pieza_cogida	Irb120_profibus_original	di8

Tabla 4-3: Tabla de conexiones de la pinza

Un ejemplo del código en RAPID para la apertura y cierre de la pinza podría ser el siguiente:

Cuando el robot llega hasta la posición donde queremos soltar el objeto, esperamos un segundo y ponemos la salida digital "do5" a nivel alto mediante la función SetDO provocando la apertura de la pinza, esperamos 0.2 segundos y volvemos a poner la salida digital "do5" a nivel bajo.

```
! Abre la pinza
   PROC AbrirPinza()
      WaitTime 1;
      SetDO do5,1;
      WaitTime 0.2;
      SetDO do5,0;
    ENDPROC
```

```
! Cierra la pinza
   PROC CerrarPinza()
      WaitTime 1;
      SetDO do6,1;
      WaitTime 0.5;
      SetDO do6,0;
   ENDPROC
```

4.3 MODELADO DE LOS OBJETOS DE TRABAJO

Un objeto de trabajo es un sistema de coordenadas utilizado para describir la posición de una pieza de trabajo. El objeto de trabajo se compone de dos bases de coordenadas: la base de coordenadas del usuario y la del objeto. Todas las posiciones que programe serán relativas a la base de coordenadas del objeto, que a su vez es relativa a la base de coordenadas del usuario, a su vez relativo al sistema de coordenadas mundo.

En la estación real del laboratorio existen dos objetos de trabajo que hay que modelar, una mesa blanca inclinada y una mesa negra. Para ello primero creamos la geometría de los objetos de trabajo y posteriormente en la pestaña de inicio de RobotStudio podemos definir los parámetros de nuestros objetos de trabajo como el sistema de coordenadas del objeto o el sistema de coordenadas del usuario.

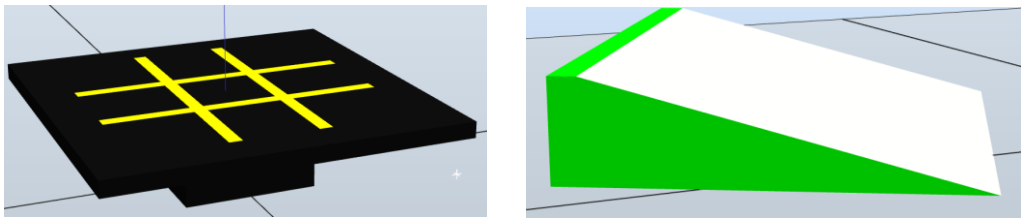


Figura 4.9: Modelado de los objetos de trabajo de la estación

4.4 MODELADO DE LA BOTONERA

La botonera real está compuesta por 16 botones negros que van conectados a las entradas digitales del robot IRB-120 y por 16 luces led que van conectadas a las 16 salidas digitales del robot.

Cuando se activa una salida digital del robot la luz led correspondiente se enciende indicando de una forma visual que la salida está activa. Cuando pulsamos uno de los botones, ponemos a nivel alto la entrada digital correspondiente del robot.

Por tanto el componente inteligente que modela la botonera creada cuenta con 16 entradas digitales y otras 16 salidas digitales que se conectarán con las E/S digitales del robot. Para la simulación del encendido de las luces contamos con la propiedad Highlighter que nos permite cambiar de color un objeto, de esta forma cuando se active una salida del robot, el elemento correspondiente de la botonera modelada cambiará de color simulando el encendido de la luz led.

En la siguiente imagen podemos ver el modelo de la botonera, los botones son los cilindros de color negro y los led son los cilindros de color blanco, cuando una salida del robot se active el cilindro de color blanco correspondiente cambiara a color rojo.

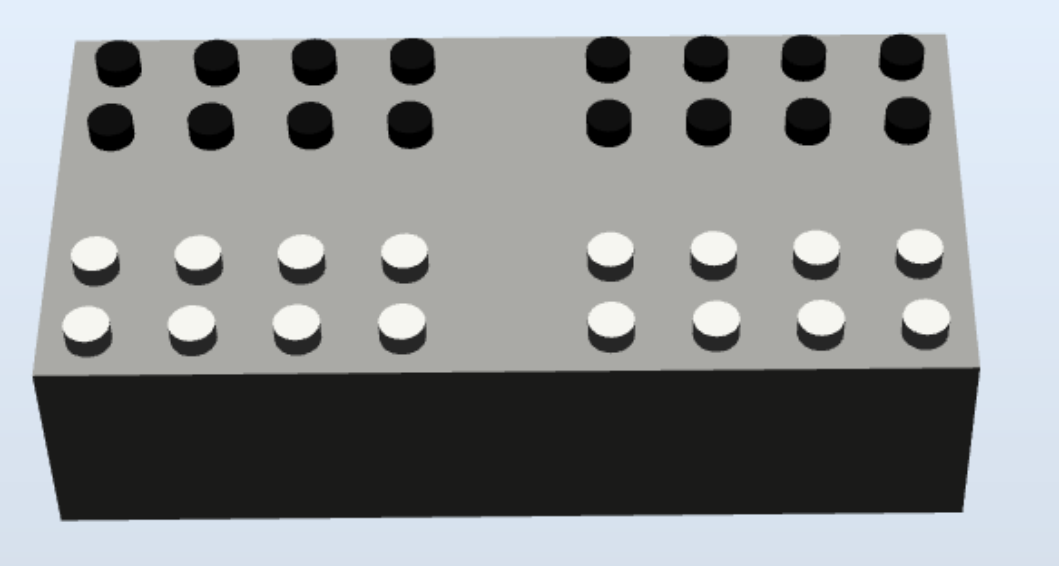


Figura 4.10: Modelado de la botonera

4.5 MODELADO DE PIEZAS

Por último queda modelar los dados, los rotuladores y el soporte de rotuladores.

- **Dados:** En la pestaña de modelado, RobotStudio permite crear un sólido con forma de tetraedro el que tanto la longitud, la altura y la anchura son 28 mm.

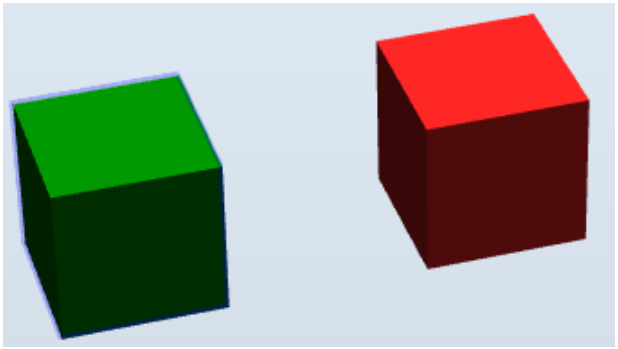


Figura 4.11: Modelo de los dados del laboratorio

- **Rotuladores:** Los rotuladores también han sido modelados en RobotStudio, está compuesto de un tetraedro, un cilindro y un cono. El resultado es el que se muestra en la siguiente figura.

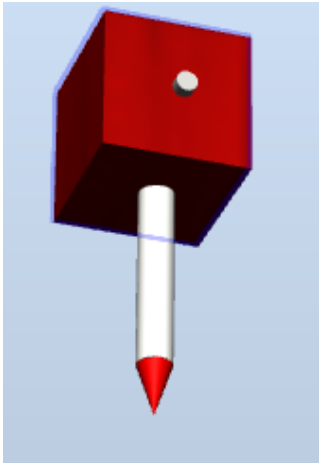


Figura 4.12: Modelo del rotulado rojo

- **Soporte de rotuladores:** También creado en RobotStudio a partir de figuras geométricas más simples, el resultado es el de la siguiente figura.

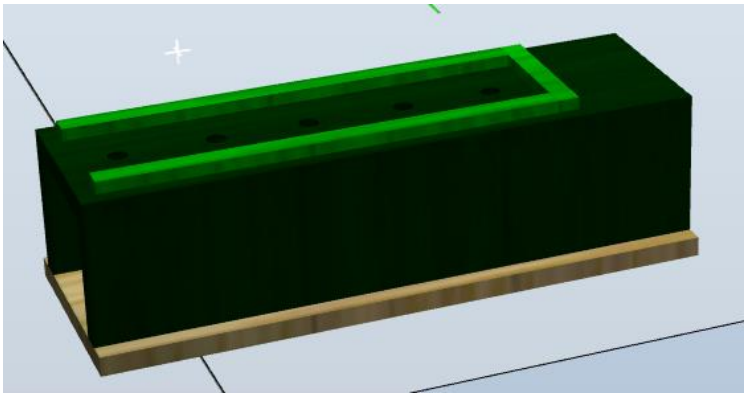


Figura 4.13: Modelo del soporte de rotuladores

4.6 CONEXIÓN ENTRE LOS ELEMENTOS DE LA ESTACIÓN Y EL ROBOT.

Una vez modelado todos los elementos de la estación robotizada hay que conectarlos con el controlador para el correcto funcionamiento de la estación. En nuestro caso los elementos a conectar son la pinza y la botonera, y la conectamos como se ve en la siguiente figura.

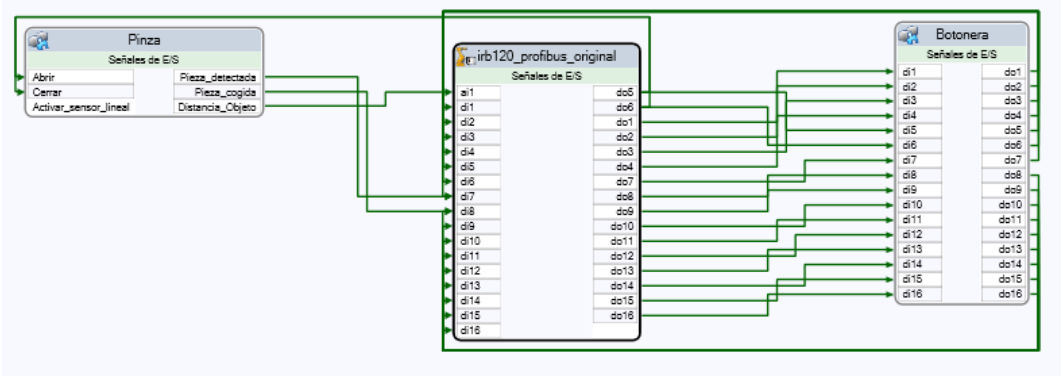


Figura 4.14: Esquema de las conexiones de la estación

Las conexiones entre los elementos y el controlador son las siguientes:

Objeto de origen	Señal de origen	Objeto de destino	Señal de destino
Irb120_profibus_original	do5	Pinza	Abrir
Irb120_profibus_original	do6	Pinza	Cerrar
Pinza	Pieza_detectada	Irb120_profibus_original	di7
Pinza	Pieza_cogida	Irb120_profibus_original	di8
Irb120_profibus_original	do1	Botonera	di1
Irb120_profibus_original	do2	Botonera	di2
Irb120_profibus_original	do3	Botonera	di3
Irb120_profibus_original	do4	Botonera	di4
Irb120_profibus_original	do5	Botonera	di5
Irb120_profibus_original	do6	Botonera	di6
Irb120_profibus_original	do7	Botonera	di7
Irb120_profibus_original	do8	Botonera	di8
Irb120_profibus_original	do9	Botonera	di9
Irb120_profibus_original	do10	Botonera	di10
Irb120_profibus_original	do11	Botonera	di11
Irb120_profibus_original	do12	Botonera	di12
Irb120_profibus_original	do13	Botonera	di13
Irb120_profibus_original	do14	Botonera	di14
Irb120_profibus_original	do15	Botonera	di15
Irb120_profibus_original	do16	Botonera	di16
Botonera	do1	Irb120_profibus_original	di1
Botonera	do2	Irb120_profibus_original	di2
Botonera	do3	Irb120_profibus_original	di3
Botonera	do4	Irb120_profibus_original	di4
Botonera	do5	Irb120_profibus_original	di5
Botonera	do6	Irb120_profibus_original	di6
Botonera	do7	Irb120_profibus_original	di7
Botonera	do8	Irb120_profibus_original	di8
Botonera	do9	Irb120_profibus_original	di9
Botonera	do10	Irb120_profibus_original	di10
Botonera	do11	Irb120_profibus_original	di11
Botonera	do12	Irb120_profibus_original	di12
Botonera	do13	Irb120_profibus_original	di13
Botonera	do14	Irb120_profibus_original	di14
Botonera	do15	Irb120_profibus_original	di15
Botonera	do16	Irb120_profibus_original	di16

Tabla 4-4: Tabla de conexiones entre los elementos de la estación y el controlador

5 APLICACIÓNES DIDÁCTICAS

5.1 JUEGO DE LAS TRES EN RAYA

Las pruebas de simulación nos permitirá comprobar si el modelo de la estación robotizada se comporta como la estación real, para ello realizaremos distintas pruebas, la primera de ellas consiste en programar mediante RAPID el conocido juego de las tres en raya y jugar con el robot, primero mediante simulación y posteriormente en la planta real.

Para el juego de las tres en raya, la mesa negra se sitúa en el centro de la estación, cinco dados de color verde a la izquierda que serán las fichas de un jugador y cinco dados de color rojo a la derecha de la mesa que serán los dados del segundo jugador.

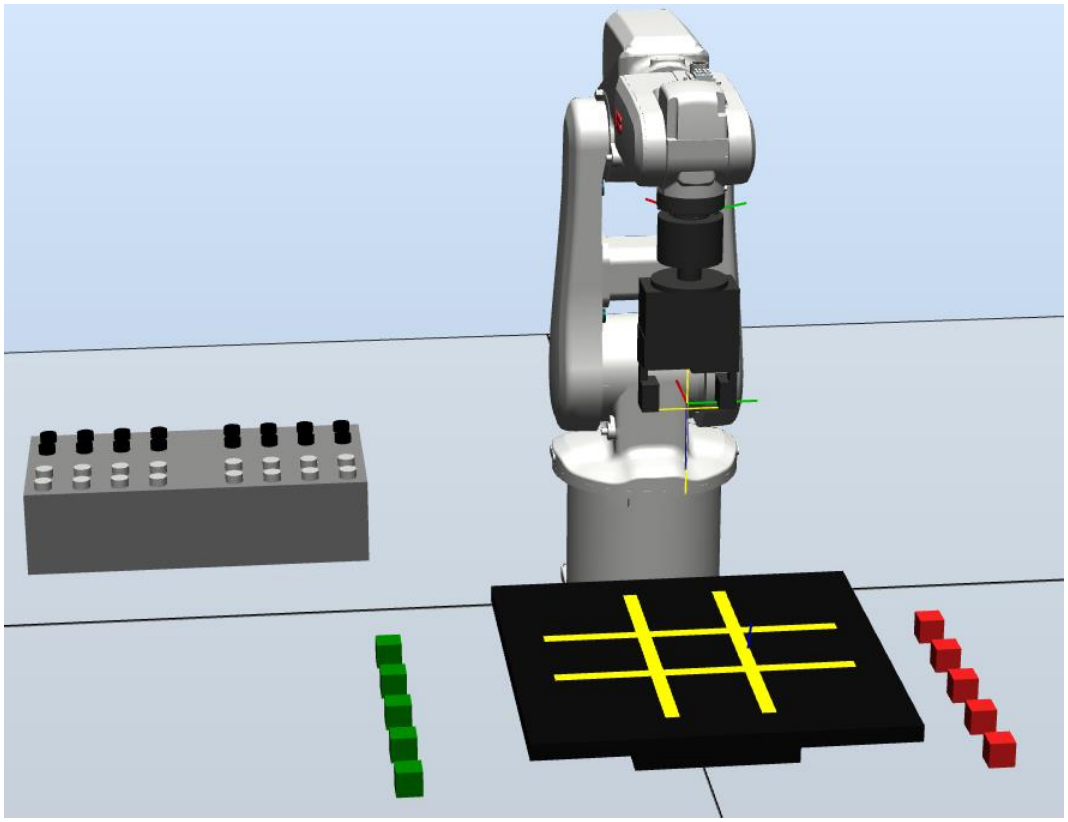


Figura 5.1: Estación robotizada para el juego de las tres en raya

Cuando se ejecute el programa lo primero que se muestra en la pantalla es un mensaje que nos permite calibrar el tablero, calibrar las piezas o continuar si realizar ninguna calibración. La calibración nos permitirá comprobar que las piezas o la mesa se encuentran situadas en la posición exacta y si no es así corregir su posición.

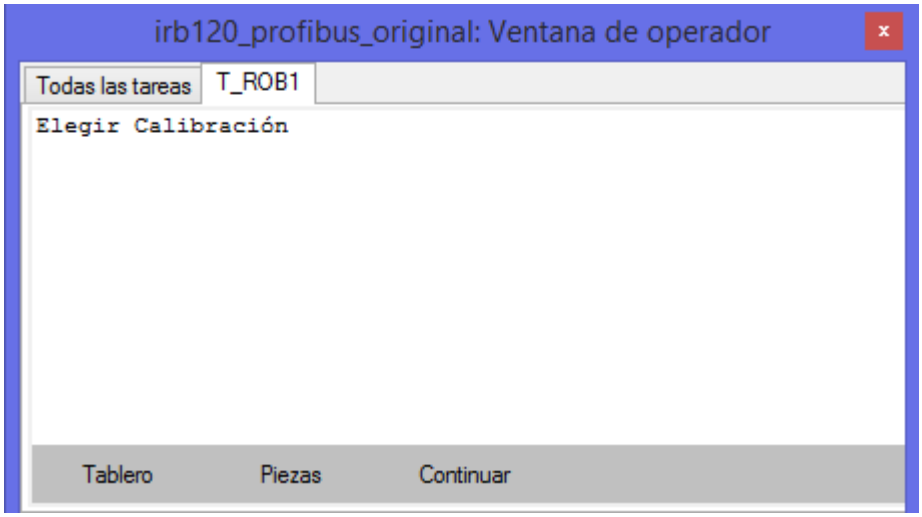


Figura 5.2: Mensaje mostrado por pantalla para elegir la calibración

Tras la calibración de las piezas y del tablero la siguiente opción que se muestra por pantalla es la elección de los dos jugadores. Se puede elegir entre una persona, un robot inteligente o un robot aleatorio. El robot inteligente posee un algoritmo que provoca que el mejor resultado posible contra él sea un empate, mientras que el robot aleatorio coloca las piezas aleatoriamente en cualquier posición.

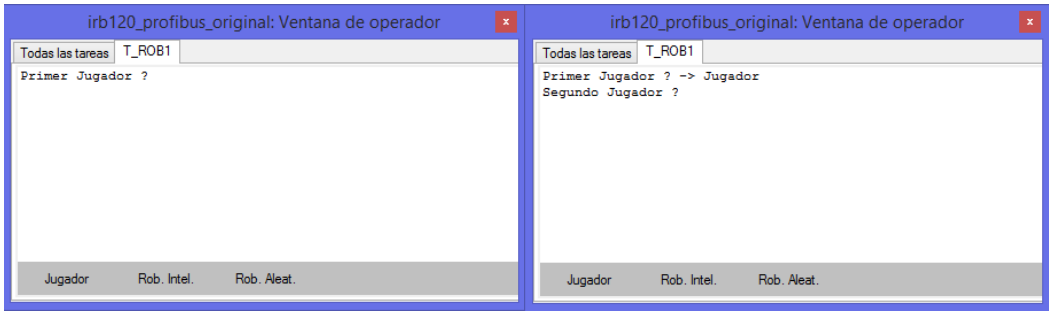


Figura 5.3: Mensaje por pantalla para la elección de los jugadores

Tras la elección de los jugadores la partida comienza, si el jugador es una persona, deberá introducir por pantalla la posición donde quiere colocar su dado y el robot la colocará, en caso de que el jugador sea el robot, esté cogerá el dado y lo colocará donde el considere. Por pantalla también se muestra cómo se desarrolla la partida mediante el dibujo del tablero.

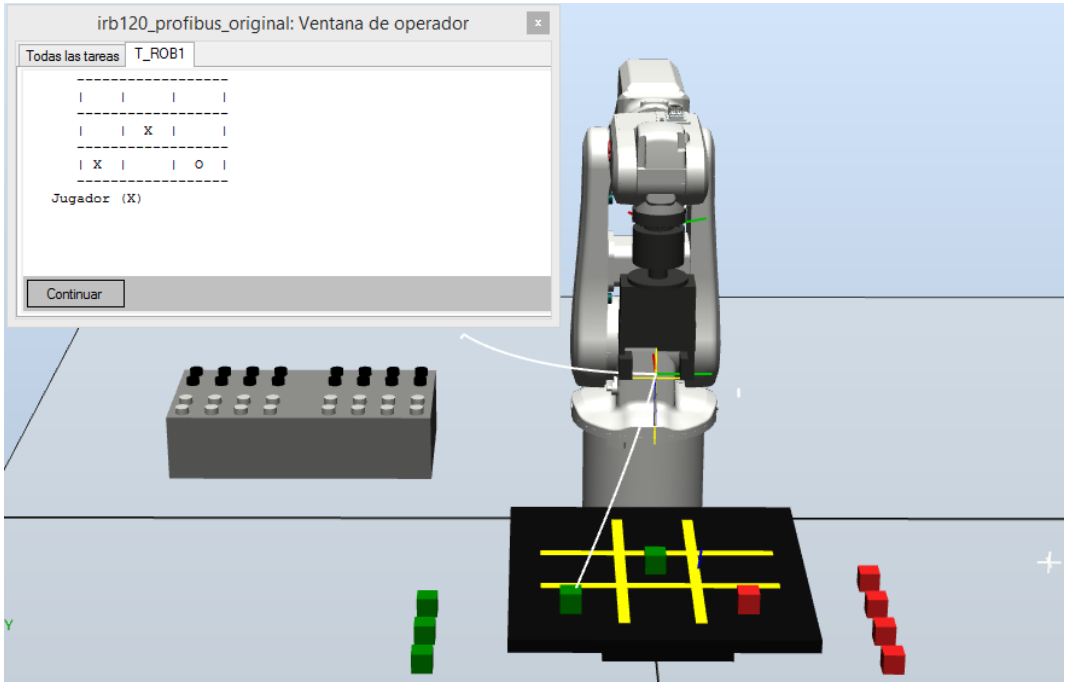


Figura 5.4: Desarrollo del juego

Cuando la partida finalice, por pantalla se nos mostrará el resultado y se nos preguntará si queremos que el robot recoja las piezas o no, si elegimos que el robot recoja las piezas este colocará las piezas una a una en su sitio y podremos volver a jugar de nuevo.

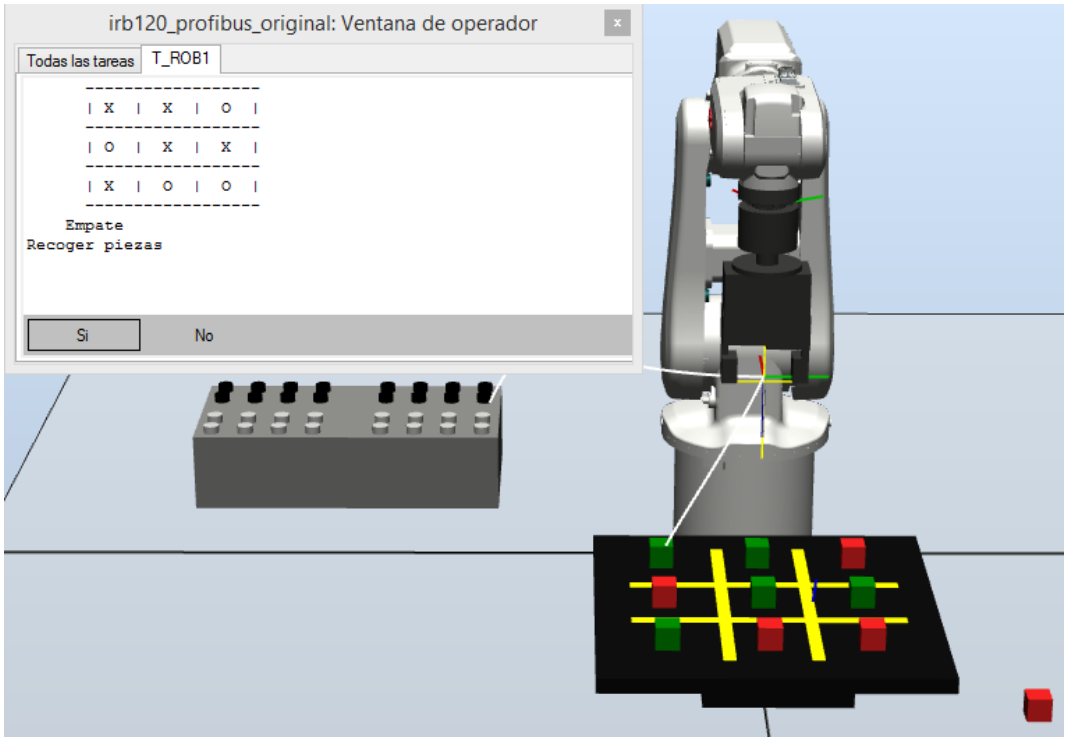


Figura 5.5: Finalización del juego de las tres en raya

Tras realizar la simulación cargamos el mismo programa en la estación real del laboratorio, la conclusión que obtenemos es que apenas se aprecian diferencias entre la simulación y el comportamiento de la estación real.

5.2 DIBUJO DE TEXTO Y FIGURAS

El objetivo de esta segunda prueba consiste en lograr que la pinza del robot del laboratorio coja un lapicero y dibuje un texto en un folio. Además podrá dibujar cualquier figura de un archivo JPG.

En primer lugar para poder escribir un texto se ha realizado una biblioteca con las instrucciones que necesita el robot para escribir cada uno de los caracteres del alfabeto, de esta manera cuando se escriba un texto, el robot leerá carácter por carácter el texto a escribir, irá a la biblioteca y leerá las reglas para poder escribir ese carácter.

5.2.1 Creación de una biblioteca de caracteres

Existen numerosas fuentes en las que cada carácter puede ser representado en una matriz 8x8 pixeles, para crear la biblioteca nos apoyaremos en la herramienta para fuentes "8x8 ROM Pixel Font Editor" que nos indicará que pixeles tienen que estar sombreados y cuales no en cada carácter. Este programa permite seleccionar un carácter de la lista de caracteres en la parte superior de la ventana. Este carácter es ahora de color verde y un gran mapa de bits de este carácter se visualiza en la parte inferior de la ventana que actúa como área de trabajo. Aquí se puede modificar píxel a píxel el carácter seleccionado pudiendo guardar los cambios haciendo clic en "Apply".

Otras funciones son "Clear" que limpia el área de trabajo, "Invert" que invierte la zona de trabajo y Copiar y Pegar. Copiar copia de la zona de trabajo en el portapapeles como mapa de bits, así que es posible pegarlo en otra aplicación, como MS Paint. Pero también copia el carácter en una serie de bytes hexadecimales en el portapapeles.

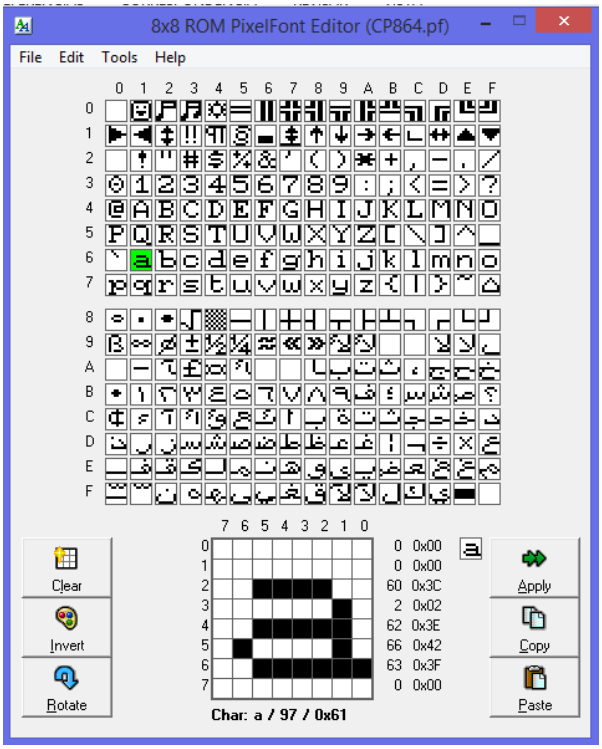


Figura 5.6: Programa 8x8 ROM Pixel Font Editor

Se ha realizado 5 bibliotecas para cada tipo de fuente elegida, los tipos de fuente elegidas han sido: "CP861", "CP864", "FANTASY", "THING8X8", "TINYTYPE". En la siguiente figura se puede ver la letra "A" representada en dos fuentes diferentes dentro una matriz de 8x8 pixeles, se puede apreciar las deferencias al seleccionar un tipo de fuente u otra.

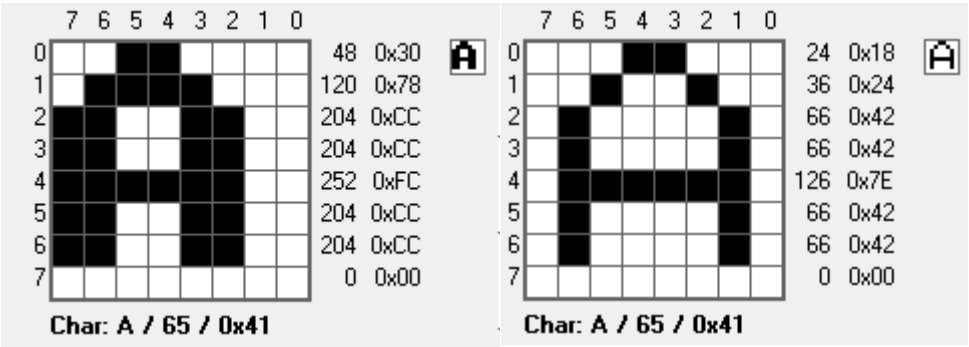


Figura 5.7: Representación de la letra "A" en el software 8x8 Pixel ROM Font Editor mediante las fuentes "CP861" y "CP864" respectivamente.

Una de las caracterizas que este programa ha incluido en sus últimas versiones es la poder exportar cada una de las fuentes a un archivo de cabecera C con la información completa de cada fuente. En el ejemplo siguiente se puede ver parte de este archivo.

```
71      0x18, 0x24, 0x42, 0x42, 0x7E, 0x42, 0x42, 0x00, // Char 065 (A)
72      0x7C, 0x22, 0x22, 0x3C, 0x22, 0x22, 0x7C, 0x00, // Char 066 (B)
73      0x1C, 0x22, 0x40, 0x40, 0x40, 0x22, 0x1C, 0x00, // Char 067 (C)
74      0x78, 0x24, 0x22, 0x22, 0x22, 0x24, 0x78, 0x00, // Char 068 (D)
75      0x7E, 0x22, 0x28, 0x38, 0x28, 0x22, 0x7E, 0x00, // Char 069 (E)
76      0x7E, 0x22, 0x28, 0x38, 0x28, 0x20, 0x70, 0x00, // Char 070 (F)
77      0x1C, 0x22, 0x40, 0x40, 0x4E, 0x22, 0x1E, 0x00, // Char 071 (G)
78      0x42, 0x42, 0x42, 0x7E, 0x42, 0x42, 0x42, 0x00, // Char 072 (H)
79      0x1C, 0x08, 0x08, 0x08, 0x08, 0x08, 0x1C, 0x00, // Char 073 (I)
```

Una vez que tenemos el archivo de cabecera C con toda la información de la fuente hay que recoger la información de cada carácter y transformarla en un formato que el robot pueda entender para dibujar cada carácter. Para ello nos apoyamos en el software Matlab.

La información que el robot necesita para dibujar un carácter es la siguiente:

- El número de trazas que se necesitan para dibujar ese carácter, dependiendo del carácter es posible que el robot no pueda hacerlo de una sola traza, es decir, dibujar el carácter de una sola vez sin levantar el bolígrafo, a veces se necesita levantar el bolígrafo del papel una, dos o tres veces para hacer una sola letra, cada trozo del carácter dibujado sin levantar el bolígrafo del papel se denomina traza.

- El número de puntos que contiene cada traza.

- El valor de X e Y de cada uno de los puntos. Hemos visto como representamos cada carácter en una matriz 8x8, por lo que podemos identificar cada elemento de la matriz con un valor X e Y.

En la siguiente figura podemos ver la letra “v” representada en la matriz 8x8, la información que el robot necesita para poder dibujar esta letra es por un lado el número de trazas, en este caso la letra “v” se puede dibujar de una sola vez sin necesidad de levantar el bolígrafo del folio por lo que el número de trazas será 1. Por otro lado tendremos el número de puntos de cada traza, en este caso todos los puntos pertenecen a la traza 1 y son un total de 9 puntos, por último el robot necesitaría las coordenadas X e Y de los nueve puntos, estos puntos son (3,2) (4,2) (5,3) (6,4) (7,5) (6,6) (5,7) (4,8) (3,8).

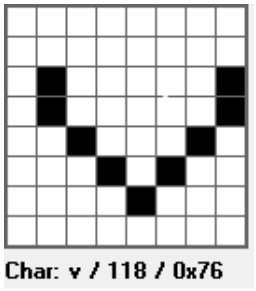


Figura 5.8: Representación del carácter "v" en una matriz 8x8

Todos estos datos son almacenados en un fichero, por lo que tendremos un fichero por cada carácter, el conjunto de todos estos ficheros constituye la librería de esa fuente. El robot cuando vaya a escribir un carácter, leerá fichero correspondiente que contiene todos los datos anteriormente explicados y necesarios para poder dibujar el carácter. El fichero del carácter "v" anteriormente es el siguiente:

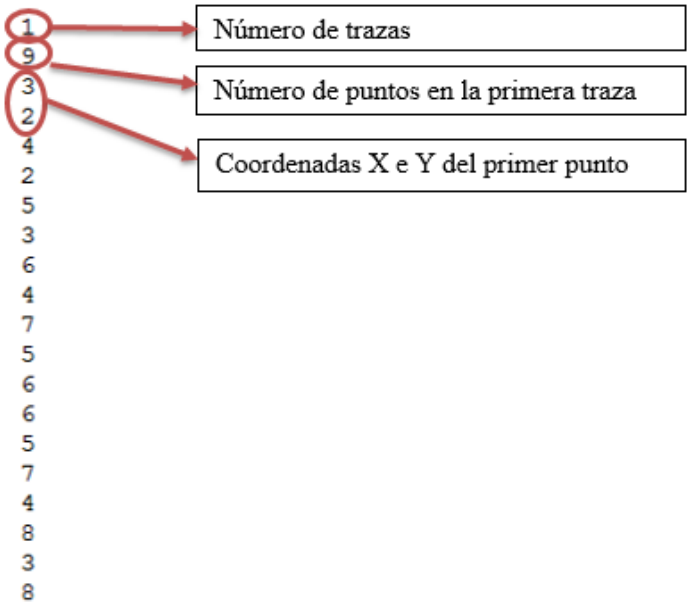


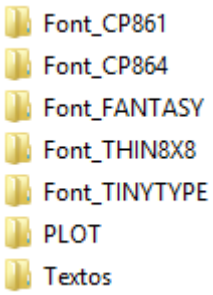
Figura 5.9: Fichero con las reglas para que robot pueda dibujar el carácter "v"

Como se ha mencionado antes para hacer cada fichero primero se ha desarrollado un programa en Matlab que cogiendo el fichero de cabecera C que nos proporciona el programa "8x8 ROM Pixel Font Editor" nos permita obtener los fichero de cada carácter como el anteriormente visto. Existen muchas formas en las que el robot puede dibujar cada carácter, las bibliotecas se han creado optimizando el número de trazas que se necesita para crear un carácter, es decir que el número de veces que el robot levanta el bolígrafo del papel para dibujar un carácter sean las mínimas posibles.

5.2.2 Escritura del texto mediante un robot

Anteriormente se ha visto como se ha creado una biblioteca con las reglas que necesita el robot para la escritura de cada carácter, ahora vamos a ver como el robot interpreta esas reglas para poder escribir cualquier texto, carácter a carácter.

Para que el programa funcione, el robot necesita leer el fichero de cada carácter ante de escribirlo, por lo que la ubicación de la biblioteca de caracteres no es algo trivial, la biblioteca se encuentra dentro de la carpeta HOME de la estación. Como vemos en la siguiente imagen, tendremos cinco carpetas con las bibliotecas de las 5 tipos de fuentes de letras creadas, una carpeta llamada Textos donde tendremos un Texto con formato *.txt* que será lo que escriba el robot. En este caso tenemos escrito las primeras líneas de El Quijote de la Mancha.



Al ejecutar el programa RAPID, lo primero que aparece por la ventana del Flexpendant son las distintas fuentes en las que podemos escribir nuestro texto, seleccionamos una y automáticamente el robot coge el cubo con el bolígrafo y lo posiciona para comenzar a escribir.

Tras coger el bolígrafo, el programa abre el documento *.txt* donde se encuentra el texto que ha de escribir el robot, leyendo su contenido letra por letra, se identifica cual es la letra que corresponde escribir, el programa abre el fichero de ese carácter y obtiene la información necesaria para que el robot pueda dibujar ese carácter, una vez recogida la información lo escribe, a continuación repite el mismo proceso para el siguiente carácter hasta que el texto haya sido escrito por completo.

En la siguiente imagen se puede ver el resultado, se ha reproducido las dos primeras líneas de El Quijote de la Mancha con cuatro fuentes diferentes de letra

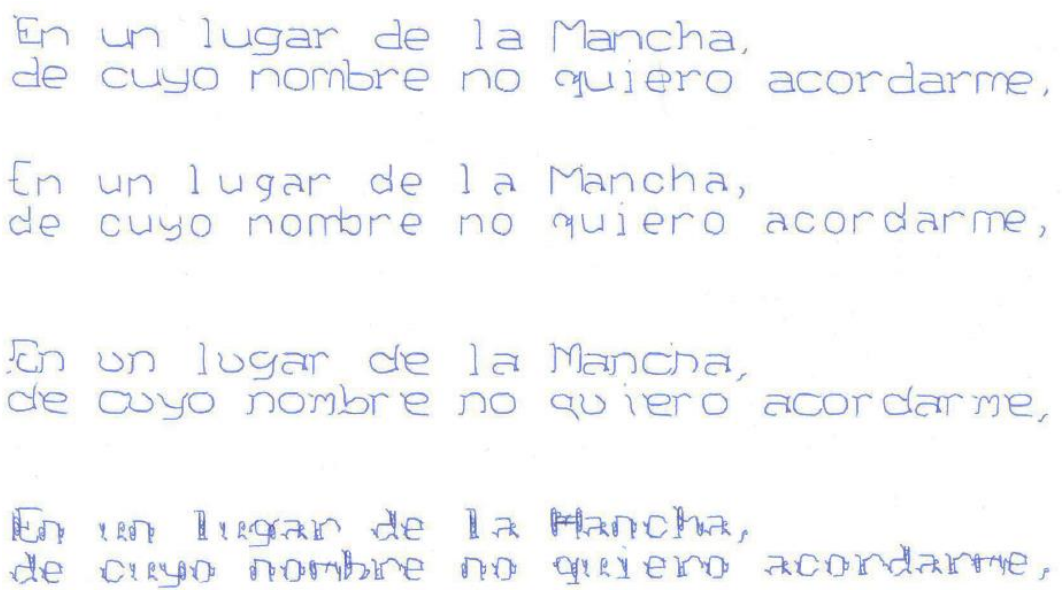


Figura 5.10: Texto escrito por el robot con diferentes tipos de fuentes.

5.2.3 Dibujo de figuras simples con el robot

El objetivo es que el robot pueda reproducir una figura simple que nosotros tengamos en formato *.jpg*.

Como hemos visto anteriormente, para poder dibujar un carácter teníamos que tener almacenado en un archivo con formato *.txt* la información necesaria para que el robot pudiese dibujar ese carácter, para el caso que nos ocupar necesitaremos conocer también el número de trazas, el número de puntos en cada traza, y las coordenadas "X "e "Y" de cada punto, la diferencia con el caso anterior es que en el ejemplo de los caracteres utilizábamos una matriz 8x8 y para este caso, simplemente pondremos las coordenadas reales en mm mapeadas.

Por tanto para que el robot reproduzca cualquier imagen necesitaremos almacenar en un archivo *.txt* la información de la imagen. Para ello se ha realizado un programa en Matlab que obtiene este archivo *.txt* a partir de una imagen con formato *.jpg*.

Si ejecutamos el programa de Matlab, se nos muestra en una ventana gráfica la figura a reproducir por el robot, en este caso es una flor. En la ventana grafica aparece un puntero para ir seleccionando los puntos de la figura, cuando se selecciona un punto, el programa muestra el siguiente mensaje:

```
Desea Cont (1), Salto (2), Fin (3):
```

Si se desea seguir añadiendo puntos se introduce el 1, si para el siguiente punto se necesitaría levantar el bolígrafo se pulsa el 2 y para terminar se pulsa el 3.

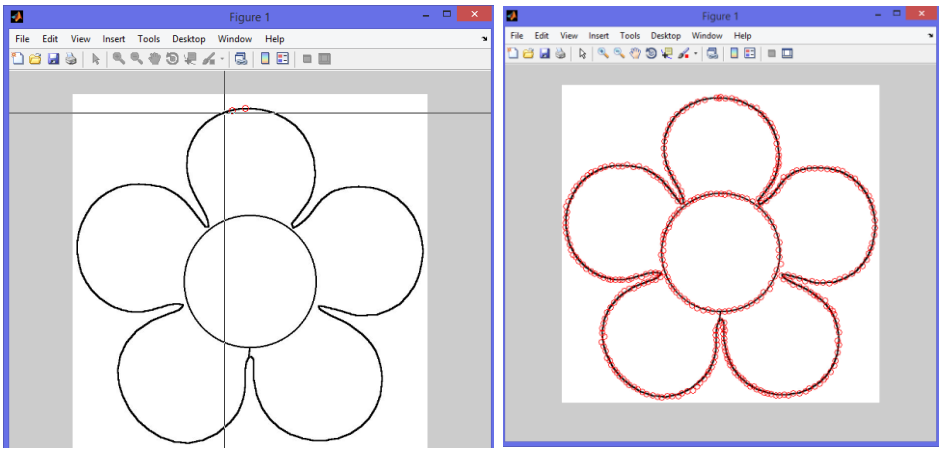


Figura 5.11: Ventana grafica de Matlab con la figura

Como vemos, en esta figura tendremos dos trazas, primero se dibujaran los pétalos, entonces se necesitará levantar el bolígrafo para hacer la segunda traza que corresponde con el círculo de dentro. Una vez terminados de marcar los puntos, el programa los almacena en un archivo *.txt* con el formato necesario para que el robot pueda dibujar la figura. Si observamos el archivo *.txt* en la siguiente figura vemos como el primer número indica el número de trazas, como vemos son dos, los dos siguientes números indican los puntos en cada traza, vemos que para dibujar la primera traza necesitaremos 281 puntos mientras para la siguiente traza necesitaremos solo 69, el resto de los puntos son las coordenadas "X" e "Y" mapeadas.

```
2
281
69
-0.23
-48.77
-0.92
-44.97
-1.61
-42.73
-2.53
-40.72
-3.46
-39.15
-4.61
```

El programa de RAPID lo que hace es leer el archivo *.txt* que debe de esta alojado en la carpeta HOME del sistema. Interpreta la información y va uniendo punto por punto hasta completar la figura. En la siguiente imagen se puede observar la marca que deja la trayectoria del robot durante la simulación y podemos comprobar como coincide con la forma de la flor.

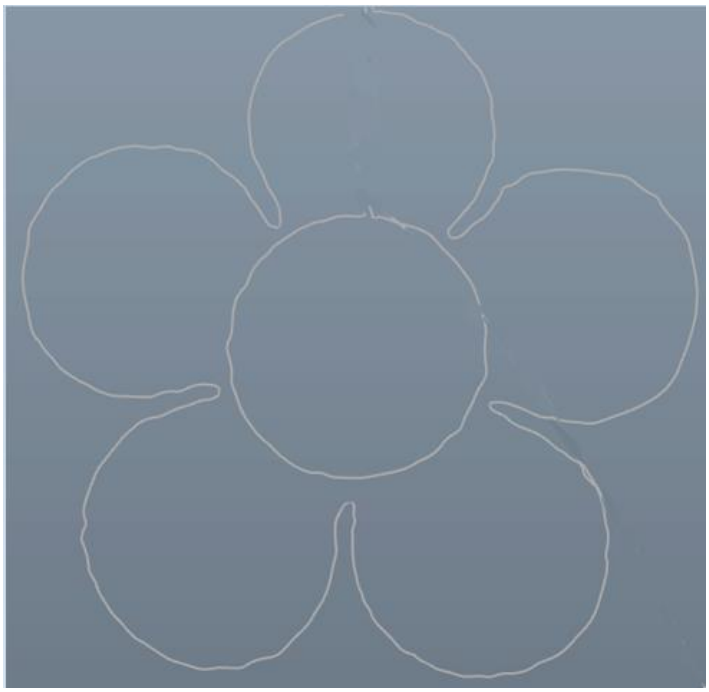


Figura 5.12: Trayectoria seguida por el robot durante la simulación

5.3 UTILIZACIÓN DE LA BOTONERA PARA MOVER EL ROBOT MANUALMENTE

Con el joystick del FlexPendant se puede mover el robot cuando este se encuentre en posición manual. El objetivo de este ejercicio es mover el robot manualmente pero en vez de utilizar el joystick usar la botonera de E/S, detectando antes de realizar un movimiento si es viable y realizándolo en caso de que lo sea. Este programa se realizará en automático, por lo que se puede realizar tanto con el robot real como con el simulado.

Uno de los problemas que se presentan a la hora de simular en RobotStudio los programas es la imposibilidad de usar el joystick del FlexPendant virtual en

simulación, para determinadas aplicaciones didácticas puedes ser interesante poder mover el robot durante la simulación de manera manual.

Los tipos de movimientos que se permiten hacer con el joystic son los siguientes:

- **Ejes:** El movimiento se robot se realiza a través de los ejes 1-3 o 4-6 del robot con el joystick.

- **Lineal:** El robot se mueve linealmente respecto de un sistema de referencia elegido.

- **Reorientación**: El robot se mueve en torno al TCP con distintas orientaciones.

De estos tres tipos de movimientos, nosotros solo implementaremos los dos primeros, el movimiento por ejes y el movimiento lineal respecto del sistema de referencia mundo. Además el robot se podrá mover en dos velocidades, rápida y lenta.

Cuando se comienza la ejecución del programa por la pantalla del FlexPendant nos aparece un menú con las diferentes opciones y configuraciones para poder mover el robot con los pulsadores de la botonera.

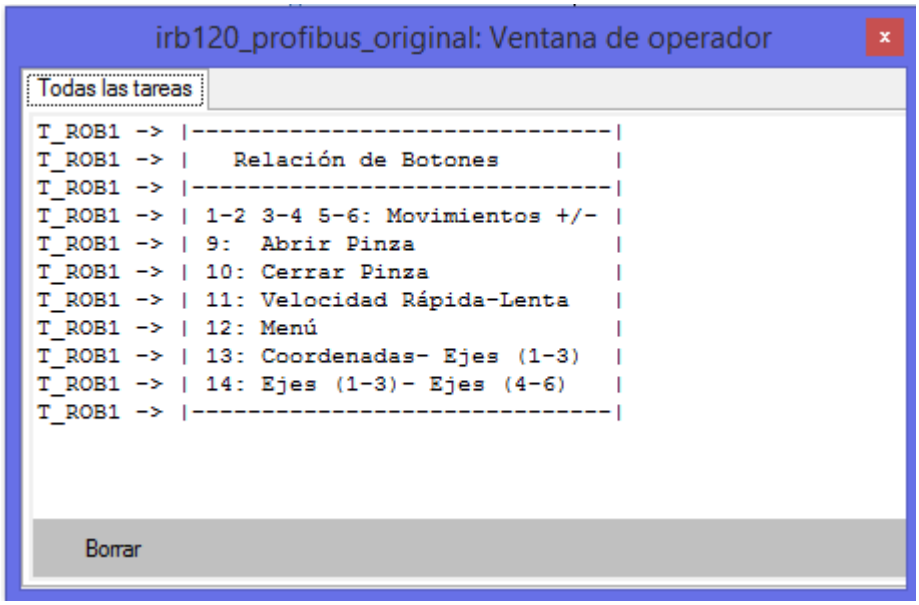


Figura 5.13: Pantalla del FlexPendant con el menú de la aplicación

- Las 6 primeras entradas sirven para mover el robot, dependiendo de la configuración seleccionada el tipo de movimiento será por ejes o por coordenadas.

- La entrada 9 realiza la apertura de la pinza.
- La entrada 10 permite el cierre de la pinza cogiendo el objeto.
- La entrada 11 permite cambiar la velocidad de rápida a más lenta o viceversa.
- La entrada 12 muestra el menú por pantalla.
- Con la entrada 13 permite cambiar el tipo de movimiento del robot, los dos movimientos son por ejes, o por coordenadas.
- Con la entrada 14 de la botonera se puede cambiar que ejes es el que se quiere mover, los ejes 1,2 y 3 o los ejes 4, 5 y 6.

Cabe recordar que las entradas 7 y 8 no se pueden utilizar ya que se corresponden con el sensor de línea de la pinza y con el sensor de presión que indica si una pieza ha sido cogida.

El funcionamiento del programa se basa en interrupciones, al pulsar una de las entradas se produce una interrupción y se ejecuta una rutina TRAP específica para esa entrada.

Además, se ha creado una función para garantizar que el robot no sobrepasa los límites y se produzca algún fallo, antes de que el robot realice el movimiento, esta función verifica que el robot no va a sobrepasar los límites, si la función determina que se van a sobrepasar los límites, impide el movimiento y muestra un mensaje de error por la pantalla del FlexPendant.

5.4 MOSTRAR POR EL FLEXPENDANT EL VALOR DE UN SENSOR DE ULTRASONIDOS

El objetivo de esta aplicación es poner en práctica el trabajo realizado previamente sobre la sensorización de la célula robótica añadiendo sensores externos comunicados con micro-procesador Arduino y la librería OPC de Matlab. Esta aplicación consistirá en mostrar por el FlexPendat el valor de un sensor en cada instante.

El sensor con el que contamos es el sensor de ultrasonidos HC-SR04, este sensor incorpora un par de transductores de ultrasonido que se utilizan de manera conjunta para determinar la distancia del sensor con un objeto colocado enfrente de este. Una de sus características principales es que es un sensor de muy bajo coste, quizás ese sea el motivo de su popularidad.

La interfaz digital se logra mediante 2 pines digitales: el pin de trigger (disparo) y echo (eco).

- El primero recibe un pulso de habilitación de parte del microcontrolador, mediante el cual se le indica al módulo que comience a realizar la medición de distancia.
- A través de un segundo pin (echo) el sensor "muestra" al microcontrolador un puso cuyo ancho es proporcional al tiempo que tarda el sonido en viajar del transductor al obstaculo y luego de vuelta al módulo.

Mediante una sencilla formula puede estimarse entonces la distancia entre el sensor y el obstáculo si se conoce el tiempo de viaje del sonido así como la velocidad de propagación de la onda sonora

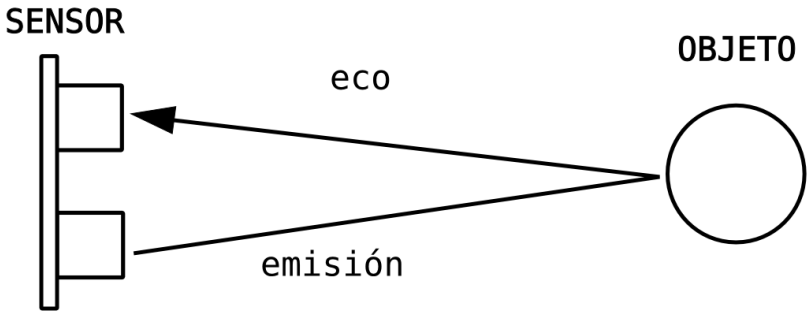


Figura 5.14: Funcionamiento de un sensor de ultrasonidos

Distancia = {(Tiempo entre Trig y el Echo) * (V.Sonido 340 m/s)}/2

5.4.1 Esquema de conexiones del sensor HC-SR04 con Arduino

El sensor consta de 4 pines:

- "VCC" conectado a la salida de 5V de la placa.
- "Trig" conectado al pin digital 12.
- "Echo" al pin de entrada digital 13.
- "GND" a tierra.

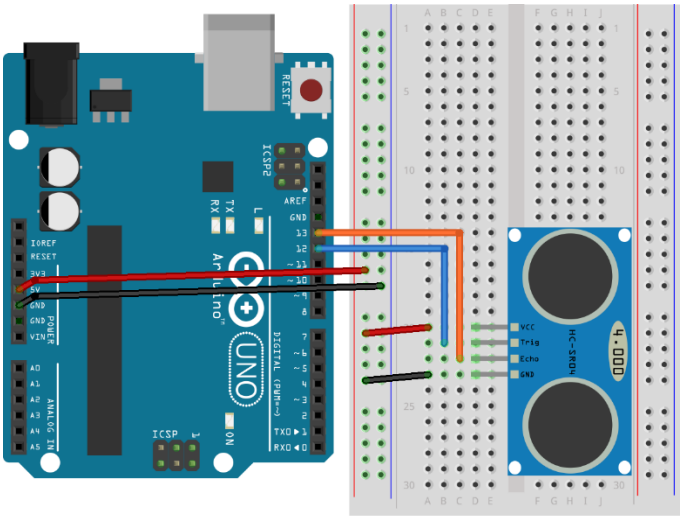


Figura 5.15: Esquema de conexiones entre Arduino y el sensor HC-SR04

5.4.2 Código Arduino

El código de Arduino que se muestra en el Anexo E consiste en iniciar la comunicación serial y mantenerse a la escucha hasta que recibe por el puerto serie un carácter (A, B, C,…) indicándole cual es el sensor del que queremos obtener la lectura, en nuestro caso solo tendremos un sensor que es el sensor de ultrasonidos.

Por lo tanto cuando se pide a Arduino que envié el valor del sensor A, Arduino realiza los cálculos necesarios para enviar el valor de la distancia que hay entre el sensor de ultrasonidos y el objeto detectado, Arduino emite un pulso de disparo con el pin Trig y calcula cuanto tiempo tarda la señal en llegar a ECHO, con ese tiempo, sabiendo la velocidad del sonido lo transformamos a milímetros, lo enviamos por el puerto serial y Arduino se mantiene a la espera de recibir la siguiente petición .

5.4.3 Código Matlab

Lo primero que se realiza en Matlab es, por un lado abrir el puerto serie para la comunicación con Arduino y por otro conectarnos al OPC Server de ABB que previamente tiene que estar configurado y habilitado.

```
%% Abrir puerto serie

delete(instrfind({'Port'},{'COM3'}));
puerto_serial=serial('COM3');
puerto_serial.BaudRate=9600;
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');

fopen(puerto_serial);
```

```
%% Conectarse a OPC

%N° de sensores, en nuestro caso el de ultrasonidos
N=1;

% La variable del controlador IRC5 donde almacenaremos
% el valor del sensor de ultrasonidos
names= {'Sen_1'};

opcreset;

SenItem= cell(1,N);

% Construimos un objeto de OPC Data Access para el host
% especificado y la ID del OPC server
da= opcda('localhost', 'ABB.IRC5.OPC.Server.DA');

% Conectamos el OPC Data Access
connect(da);
% Añade un grupo al objeto
grp= addgroup(da);

% Añadimos al grupo la variable que queremos comunicar por OPC
 for i= 1:N,
```

Tras abrir la comunicación puerto serial y establecer la conexión con el OPC Server, desde Matlab realizamos cada 0.5 segundos una petición de lectura del sensor de ultrasonidos y esperamos a que Arduino nos envíe la distancia. En el caso de tener más de un sensor se realizaría una petición de lectura del siguiente sensor y nos mantendríamos de nuevo a la espera de que Arduino nos diera la lectura de este segundo sensor.

Tras obtener la lectura del sensor de ultrasonidos, mediante OPC actualizamos el valor de la variable "Sen_1" del controlador IRC5 al valor recibido del sensor. El código sería el siguiente:

```
%% Recibimos los datos de los sensores de arduino y lo enviamos
mediante OPC
 while true,

   %Realizamos la petición de lectura del sensor A
   fwrite(puerto_serial,'A','uchar');
   %Recibimos la lectura del sensor A
   distancia = fscanf(puerto_serial);

   %Realizamos la petición de lectura del sensor B
   %Recibimos la lectura del sensor B


   %Enviamos por OPC  a RobotStudio la lectura del sensor A
   write(SenItem{1},distancia);
   %Enviamos por OPC  a RobotStudio la lectura del sensor B

  pause(0.5);

 end
```

5.4.4 Resultado final

Como se ha visto anteriormente la variable del controlador IRC5 "Sen_1" se actualiza desde Matlab cada 0.5 con el valor del sensor de ultrasonidos, por lo tanto ya tenemos disponible en el controlador una variable "Sen_1" que contiene el valor del sensor de ultrasonidos actualizada cada 0.5 segundos.

Para este ejemplo propuesto, se pretende mostrar por pantalla del FlexPendant el valor del sensor de ultrasonidos, por lo que el código en RAPID simplemente leerá valor de la variable "Sen_1" y la mostrará por pantalla.

El código RAPID es el que se muestra a continuación:

```
MODULE M_Lectura_sensores

!   Declaración de variables
   VAR Num Sensor1;

   PROC main_Lectura_sensores()
      TPWrite "Inicio Lectura de sensores";
      WHILE true DO

!        Leemos la señal analogía Sen_1
         Sensor1 :=  AOutput(Sen_1);

!         Mostramos por el FlexPendant el valor del sensor
         TPWrite "La distancia es: "\Num:=Sensor1;
         WaitTime 0.5;
      ENDWHILE
   ENDPROC

ENDMODULE
```

En la siguiente imagen podemos observar como en la ventana del FlexPendant se muestra la distancia en milimetros del objeto detectado por el sensor de ultrasonidos en milimetros.

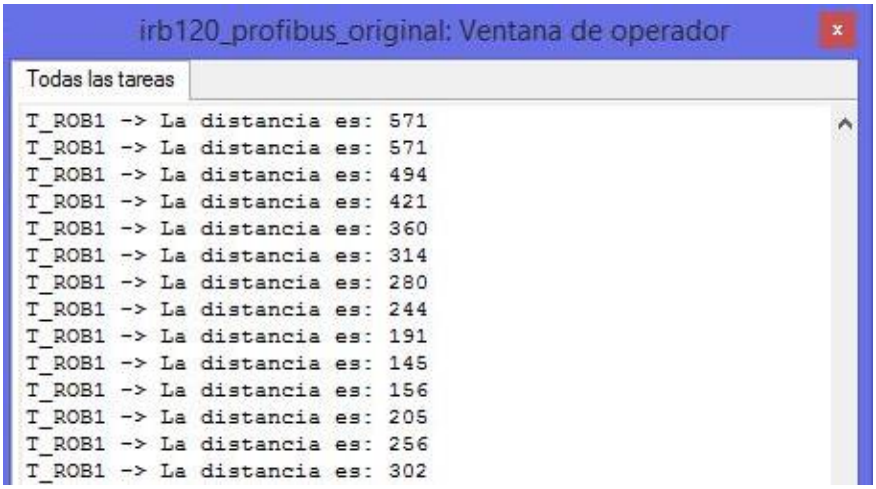


Figura 5.16: Pantalla del FlexPendant con los resultados de la lectura del sensor

5.5 RECONOCIMIENTO DE FORMAS MEDIANTE LA UTILIZACIÓN DE UN SENSOR DE ULTRASONIDOS

El objetivo de esta aplicación es poner en práctica el trabajo realizado previamente sobre la sensorización de la célula robótica añadiendo sensores externos comunicados con micro-procesador Arduino y la librería OPC de Matlab. En este ejemplo colocaremos un objeto en la parte frontal del Robot y el objetivo será el de obtener de una forma gráfica en Matlab la forma de este objeto.

Para ello contaremos con el sensor de ultrasonidos hc-sr04 que ira acopado a un cubo de madera para poder ser cogido con la pinza del robot de tal forma que el sensor se situará en el TCP del robot. El robot cogerá este cubo que contiene el sensor y realizará un barrido de la superficie donde se encuentra el objeto. Justo cuando el sensor de ultrasonidos detecte el objeto, almacenaremos en una matriz las coordenadas "X" e "Y" del TCP y la distancia que nos proporciona el sensor de ultrasonidos. Con estos datos, en Matlab trataremos de reconstruir la forma del objeto.

Figura 5.17: Alojamiento del sensor de ultrasonidos

En este ejemplo la información sobre las coordenadas "X" e "Y" del TCP no se obtiene de sensores externos si no del controlador IRC5 mediante las instrucciones de RAPID, por lo que el valor de las coordenadas se actualizan en el controlador y tendrá que ser leídas desde Matlab mediante OPC, esto constituye

una importante diferencia respecto del ejemplo anterior donde se realizaba el proceso inverso, desde Matlab modificábamos el valor de una variable mediante OPC y la leíamos a través de RAPID.

Por tanto el código en RAPID tendrá dos apartados diferenciados, por un lado la función para que el Robot realice un barrido de toda la superficie y por otro lado, crearemos una rutina TRAP en la que cada 0.1 segundos se actualicen las variables que contienen el valor de las coordenadas "X" e "Y" del TCP.

En RAPID las rutinas TRAP proporcionan una forma de responder a las interrupciones. Las rutinas TRAP pueden asociarse con una interrupción determinada. Cuando posteriormente se produce una interrupción concreta, se ejecuta la rutina TRAP correspondiente. Las rutinas TRAP no pueden ejecutarse explícitamente desde el programa. En nuestro caso activaremos la rutina TRAP de nuestro programa de forma periódica con un periodo de muestreo determinado.

La rutina TRAP que actualiza periódicamente el valor de las coordenadas "X" e "Y" del TCP es el que se muestra a continuación. El código RAPID completo se encuentra en los anexos.

```
TRAP Write_OPC_TCP
      VAR pos tcp;
      tcp := CPos(\Tool:=Pinza_clase \WObj:=wobj0);
       SetAO Tcp_x, tcp.x;
       SetAO Tcp_y, tcp.y;
  ENDTRAP
```

Esta rutina TRAP se ejecuta periodicamente cada 0.1 segundos, en la variable "tcp" de tipo pos almacenamos el TCP de la Pinza, que es justamente donde va alojado el sensor de ultrasonidos, posteriormente mediante la función SetAO almacenamos en las variables virtuales del controlado (Tcp_x, Tcp_y) el valor de las coordenadas del TCP. Hay que diferenciar entre las variables del programa y las señales virtuales creadas en el controlador. Por OPC solo podemos leer o escribir señales del controlador.

Las señales Tcp_x y Tcp_y son señales virtuales del controlador que se han creado para esta práctica, son salidas analógicas a las que podremos acceder desde Matlab mediante OPC.

El esquema de conexiones del sensor de ultrasonidos y arduinos es el mostrado en el ejemplo anterior, el codigo de Arduino tampoco cambia.

Desde Matlab recibimos la información del sensor de ultrasonidos por el puerto serial cada 0.1 segundos, cuando el sensor de ultrasonidos detecte un objeto, desde Matlab leeremos por OPC las varialbes Tcp_x y Tcp_y almacenandolas en una matriz, tambien almacenaremos la distancia al objeto que nos proporciona el sensor de ultrasonidos de tal forma que al final tendremos una nube de puntos para poder reconstruir la forma del objeto en Matlab.

Como se ha visto anteriormente, cada 0.1 segundos almacenamos en una matriz las coordenadas (x y z) del objeto cuando este es detectado por el sensor de ultrasonido, hemos visto que las coordenadas x e y las obtenemos del TCP de la pinza que es donde va alojado el sensor, y la coordenada z es la distancia desde la mesa hasta el sensor de ultrasonidos restandole el valor de la distancia del sensor hasta el objeto. En Matlab, realizamos una triangulación 3D de los puntos obtenidos anteriormente y posteriormente mediante la triangulación 3-D de Delaunay representamos los tetraedros obteniendo una representación del objeto.

Si colocamos una piramide de base pentagonal y ejecutamos el programa obtenemos el siguiente resultado:

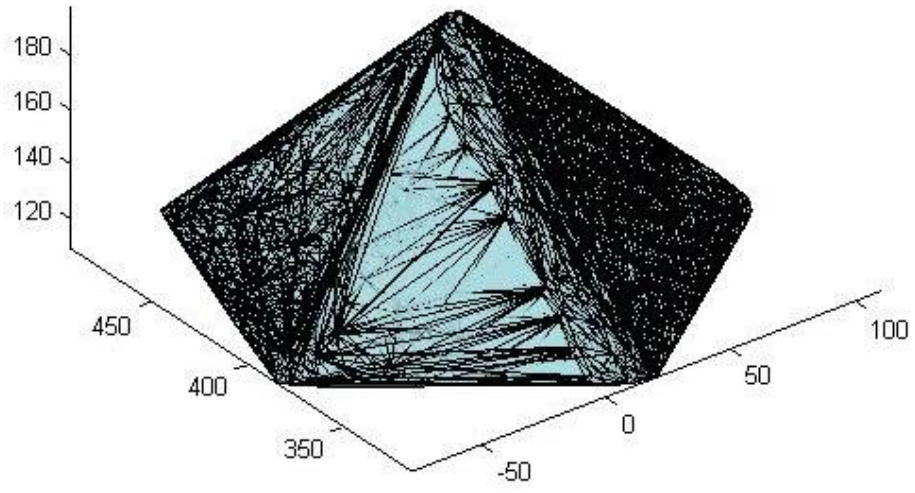


Figura 5.18: Representación gráfica 3D en Matlab del objeto

Si realizamos una triangulación en 2D, solo con las componente (x, y) y las representamos gráficamente obtenemos el contorno del objeto.

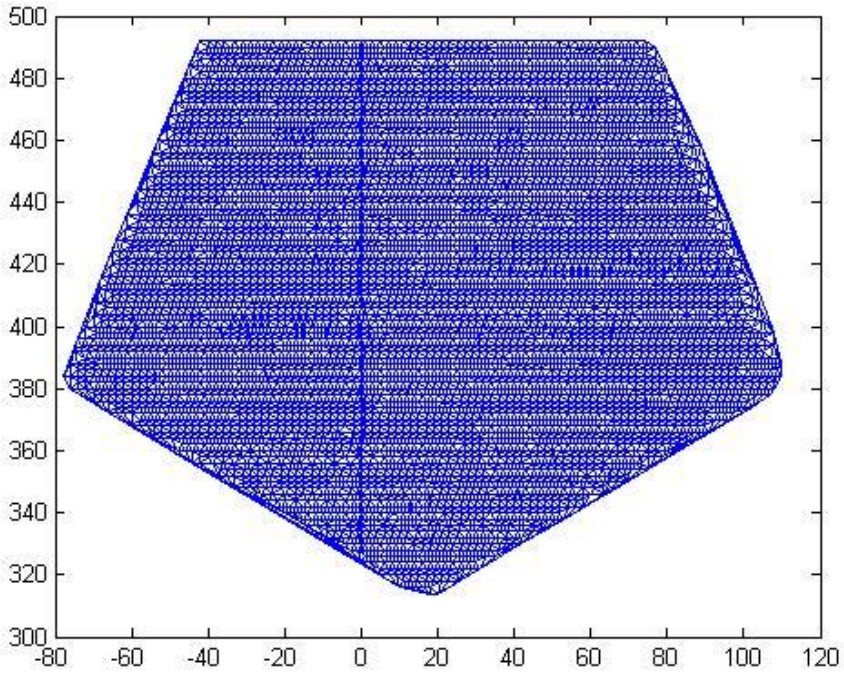


Figura 5.19: Representación gráfica 2D en Matlab del objeto

6 CONCLUSIONES Y LÍNEAS FUTURAS

Las conclusiones que podemos hemos obtenido al concluir este proyecto son:

- La importancia de construir los modelos de una planta robotizada para su simulación ya que permiten incrementar la rentabilidad del sistema robotizado mediante tareas como formación, programación y optimización, sin afectar a la producción, lo que proporciona una reducción de riesgos y un incremento de la productividad.

- El software RobotStudio de ABB permite realizar simulaciones muy cercanas a la realidad, es uno de los programas más utilizados en entornos reales para realizar la programación "offline" de células robóticas industriales.

- La importancia de OPC en la industria permitiéndonos establecer una comunicación estandarizada entre clientes y servidores OPC solucionando el clásico problema de los drivers propietarios, esto permite intercambiar libre y fácilmente información entre dispositivos y aplicaciones de cualquier tipo, como por ejemplo soluciones de HMI (Human Machine Interface), planillas de cálculo, motores de base de datos, ERPs, entre otras.

- Arduino puede ser una buena solución de bajo coste a la hora de elegir un microcontrolador, tiene una gran popularidad y en la red se puede encontrar una gran cantidad de información.

Se proponen las siguientes líneas futuras de trabajo para aquellos interesados en la robótica:

- La comunicación entre Matlab y el robot se puede realizar mediante socket y compararla con la comunicación OPC.

- Se puede comunicar directamente Arduino con el robot sin necesidad de usar Matlab.

7 BIBLIOGRAFÍA

[1] Fundamentos de Robótica, Antonio Barrientos; McGraw Hill.

[2] **ROBOTSTUDIO:** Manual del operador RobotStudio 6.01. ID de documento: 3HAC032104-005 Revisión: P.

[3] **IRB 120**: Manual del producto: IRB120-3/0.6, IRB 120T-3/0.6. Doc. ID: 3HAC035728-005.

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

[5] **RAPID:** Manual de referencia técnica Instrucciones, funciones y tipos de datos de RAPID: ID de documento: 3HAC16581-5 Revisión: J.

[6] Manual del operador IRC5 con FlexPendant. Doc. ID: 3HAC16590-5.

[7] **OPC SERVER ABB**: Application manual IRC5 OPC Server help Document ID: 3HAC023113-001 Revision: 8

[8] **OPC Toolbox**. Software Mathworks. http://es.mathworks.com/products/opc

[9] Manual de referencia técnica Parámetros del sistema. ID de documento: 3HAC17076-5 Revisión: K.

[10] Computational Geometry: Software Mathworks. http://es.mathworks.com/help/matlab/computational-geometry.html

[11] **ARDUINO**. Guía de usuario de Arduino. Rafael Enriquez Herrador.

[12] **ARDUINO.** Language Reference. https://www.arduino.cc/en/Reference/HomePage

[13] 8x8 Pixel ROM Font Editor. https://www.min.at/prinz/o/software/pixelfont/

8 Anexos

ANEXO A: DATASHEET IRB 120

IRB 120

Specification

Variants	Reach	Payload	Armload
IRB 120-3/0.6	580 mm	3 kg (4kg)*	0.3 kg

Features

Integrated signal supply	10 signals on wrist
Integrated air supply	4 air on wrist (5 bar)
Position repeatability	0.01 mm
Robot mounting	Any angle
Degree of protection	IP30
Controllers	IRC5 Compact / IRC5 Single cabinet

Movement

Axis movements	Working range	Maximum speed	
		IRB 120	IRB 120T
Axis 1 Rotation	+165° to -165°	250 °/s	250 °/s
Axis 2 Arm	+110° to -110°	250 °/s	250 °/s
Axis 3 Arm	+70° to -110°	250 °/s	250 °/s
Axis 4 Wrist	+160° to -160°	320 °/s	420 °/s
Axis 5 Bend	+120° to -120°	320 °/s	590 °/s
Axis 6 Turn	+400° to -400°	420 °/s	600 °/s

Performance

	IRB 120	IRB 120T
1 kg picking cycle		
25 x 300 x 25 mm	0.58 s	0.52 s
25 x 300 x 25 with 180° axis 6 reorientation	0.92 s	0.69 s
Acceleration time 0-1 m/s	0.07 s	0.07 s

Electrical connections

Supply voltage	200–600 V, 50/60 Hz
Rated power Transformer rating	3.0 kVA
Power consumption	0.25 kW

Physical

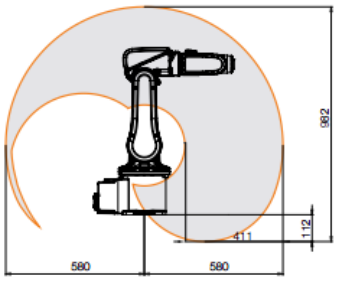
Dimension robot base	180 x 180 mm
Dimension robot height	700 mm
Weight	25 kg

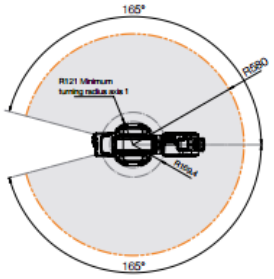
Environment

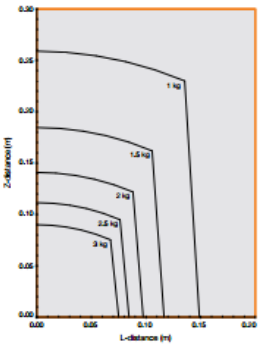
Ambient temperature for Robot manipulator:	
During operation	+5°C (41°F) to +45°C (122°F)
Relative transportation and storage	-25°C (-13°F) to +55°C (131°F)
For short periods	up to +70°C (158°F)
Relative humidity	Max 95%
Options	Clean Room ISO class 5 (certified by IPA)**
Noise level	Max 70 dB (A)
Safety	Safety and emergency stops 2-channel safety circuits supervision 3-position enabling device
Emission	EMC/EMI-shielded

* With vertical wrist
** ISO class 4 can be reached under certain conditions
Data and dimensions may be changed without notice

Working range at wrist center & load diagram







ANEXO B: DATASHEET IRC5 COMPACT CONTROLLER

The IRC5 Compact offers the capabilities of the extremely powerful IRC5 controller in a truly compact format. In addition, the IRC5 Compact delivers space saving benefits and easy commissioning through one phase power input, external connectors for all signals and a built in expandable 16 in, 16 out I/O system. Utilising many of the well-known features of the IRC5 controller, the compact version offers familiar programming and operation, ensuring no additional training is required. The IRC5 Compact controller is available for the lower end robots of the IRB range.

Specification	
Controller hardware:	Multi-processor system
	PCI bus
	Flash disk mass memory
	Energy back-up power failure handling
	USB memory interface
Control software:	Well proven real-time OS
	High-level RAPID programming language
	PC-DOS file format Preloaded software, available on DVD
	Extensive functionality set, see separate RobotWare data sheet
Electrical Connections	
Supply voltage:	Single phase 220/230 V, 50-60 Hz
Physical	

Size HxWxD	Weight
258 x 450 x 580	28.5 kg

Environment	
Ambient temperature:	+ 0° C (32°F) - +45°C (113°F)
Relative humidity	Max. 95%
Level of protection	IP20
Fulfilment of regulations	Machine directive 98/37/EC regulation
	Annex II B
	EN 60204-1:2006
	ISO 10218-1:2006
	ANSI/RIA R 15.06 -1999
User Interfaces	
Control panel	On cabinet or remote
FlexPendant	Weight 1 kg
	Graphical color touch screen
	Joystick
	Emergency stop
	Support for right- and left hand operators
	USB Memory support
Maintenance	Diagnostic software
	Recovery procedure
	Logging with time stamp
	Remote Service enabled
Supported robots	IRB 120
	IRB 140
	IRB 260
	IRB 360
	IRB 1410
	IRB 1600

Safety	
Basic:	Safety and emergency stops
	2-channel safety circuits supervision
	3-position enabling device
Machine Interfaces	
Inputs/outputs:	Standard 16/16 (up to 2200)
Digital:	24V DC or relay signals
Analogue:	2 x 0-10V, 3x ±10V, 1x4-20mA
Serial channel:	1 x RS 232 (RS422 with adaptor)
Network:	Ethernet (10/100 Mbits per second)
Two channels:	Service and LAN
Fieldbus Master:	DeviceNet™
	PROFIBUS DP
	Ethernet/IP™

ANEXO C: DATASHEET MÓDULO DE ULTRASONIDO HC-SR04

Product features:

Ultrasonic ranging module HC - SR04 provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The basic principle of work:

(1) Using IO trigger for at least 10us high level signal,

(2) The Module automatically sends eight 40 kHz and detect whether there is a pulse signal back.

(3) IF the signal back, through high level , time of high output IO duration is the time from sending ultrasonic to returning. Test distance = (high level time×velocity of sound (340M/S) / 2.

Wire connecting direct as following:

- 5V Supply.
- Trigger Pulse Input.
- Echo Pulse Output.
- 0V Ground.

Electric Parameter:

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Dimension	45*20*15mm

ANEXO D: DATASHEET ARDUINO MEGA ADK

The Arduino ADK R3 is a microcontroller board based on the ATmega2560 (datasheet). It has a USB host interface to connect with Android based phones.

It is compatible with Android's Accessory Development Kit examples. It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button.

The ADK is based on the Mega 2560. Plus it has an USB Host circuit that enable this board to communicate with USB Devices, and give them power supply.

Additional features coming with the R3 version are:

- ATmega16U2 instead 8U2 as USB-to-Serial converter.

- 1.0 pinout: added SDA and SCL pins for TWI communication placed near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board and the second one is a not connected pin, that is reserved for future purposes.

- stronger RESET circuit.

Technical Specifications

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	9V
Input Voltage (limits)	7-18V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

Power

The Arduino Mega can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less tan 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega differs from all preceding boards in that it does not use the FTDI USB-toserial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

● **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, Access it through this pin.

● **5V.** The regulated power supply used to power the microcontroller and other

components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.

● **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

● **GND.** Ground pins.

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode() , digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

● **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

● **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.

● **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.

● **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI

communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

● **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

● **I2C: 20 (SDA) and 21 (SCL).** Support I2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I2C pins on the Duemilanove or Diecimila.

The Mega has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

● **AREF.** Reference voltage for the analog inputs. Used with analogReference().

● **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega has a number of facilities for communicating with a computer,another Arduino, or other microcontrollers. The ATMega provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

The ATMega also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. For SPI communication, use the SPI library.

Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the ArduinoMega is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATMega via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload. This setup has other implications. When the Mega is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

ANEXO E: CÓDIGO DE ARDUINO PARA LA LECTURA DEL SENSOR DE ULTRASONIDOS

```
/*****************************************************************
 Sensor de Ultrasonidos HC-SR04 con Arduino
*****************************************************************/

/*******************
Conexiones:
 - TRIG al pin D12
 - ECHO al pin D13
 - VCC to Arduino 5v
 - GND to Arduino GND
*******************/
#include <stdlib.h> //Liberia para el manejo de cadenas de
caracteres
String readString;

// Definimos los pines
#define PIN_TRIG 12
#define PIN_ECO  13

void setup() {
  // Inicialización de la comunicación serial
  Serial.begin (9600);

  // Inicializacion de pines digitales
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECO, INPUT);
}

void loop() {
  long duracion, distancia;  // Variables
  char bandera;


  if (Serial.available()) {
       bandera = Serial.read();
  }


  if(bandera == 'A') {

    /* Hacer el disparo */
    digitalWrite(PIN_TRIG, LOW);
    delayMicroseconds(2);
    digitalWrite(PIN_TRIG, HIGH);  // Flanco ascendente
    delayMicroseconds(10);         // Duracion del pulso
    digitalWrite(PIN_TRIG, LOW);   // Flanco descendente
```

```
    /* Función para medir la longitud del pulso entrante. Mide el
tiempo que transcurrido en microsegundos entre el envío
del pulso ultrasónico y cuando el sensor recibe el rebote, es deci
r: desde que el pin 12 empieza a recibir el rebote, HIGH, hasta qu
e   deja de hacerlo, LOW, la longitud del pulso entrante*/
    duracion = pulseIn(PIN_ECO, HIGH);

    /* Calculo de la distancia efectiva en mm*/
    distancia = (duracion*0.17);

  /* Imprimir resultados a la terminal serial */
    if (distancia >= 2000 || distancia < 0){
      Serial.println("0");
    }
    else {
      Serial.println(distancia);
    }

   }

   if(bandera == 'B') {
     // Cálculos del sensor B
   }

   if(bandera == 'C') {
     // Cálculos del sensor C
   }
}
```

ANEXO F: CÓDIGO DE LA APLICACIÓN 5.5 RECONOCIMIENTO DE FORMAS MEDIANTE LA UTILIZACIÓN DE UN SENSOR DE ULTRASONIDOS

Anexo F.1: Código RAPID

```
MODULE M_ReconocimientoDeFormas

   ! Definición de variables
   LOCAL PERS wobjdata
mesanegra:=[FALSE,TRUE,"",[[179.554,155.837,126.215],[0.999897,0.00977107,
0.00883153,0.00564256]],[[-0.000223517,0.000342727,-
0.000037253],[1,0.000003308,- 0.000000036,0.000002491]]];
   LOCAL PERS wobjdata
mesablanca:=[FALSE,TRUE,"",[[381.998,57.0371,68.2967],[0.456518,0.0541315,0.09986
23,0.882433]],[[0.000655651,-0.00110269,-0.000679167],[1,0.000001664,-
0.000001998,0.000007345]]];
   LOCAL PERS wobjdata
SoportesLapiceros:=[FALSE,TRUE,"",[[415.831523847,220.379710604,0],[1,0,0,0]],[[0,0,
0],[1,0,0,0]]];
   LOCAL PERS wobjdata
mesablanca_of:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[382.890935064,51.632809543,53.07
3888311],[0.46027159,0.062416665,-0.113418048,-0.878288432]]];
   LOCAL PERS tooldata
Pinza_clase:=[TRUE,[[0,0,206],[0.965926,0,0,0.258819]],[1.8,[2.7,0,84.4],[1,0,0,0],0.033,
0.034,0.007]];

   VAR intnum IWriteOPC;

   PROC main()

      F_Write_OPC_TCP;
      Escanear;

   endproc

   PROC Escanear()
!     Con el Robot realizamos un barrido
      VAR num i;
      VAR num resolucion;
      resolucion:=2;
```

```
      MoveJ [[500.87,-
80.45,240.17],[0.0117498,0.692804,0.721009,0.00547898],[-1,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v30,fine,Pinza_clase;
      MoveL
[[500.87,80.80,240.17],[0.0117497,0.692804,0.721009,0.00547891],[0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,fine,Pinza_clase;

      FOR i FROM 1 TO 100 DO
        MoveL
[[500.87-resolucion,80,240.17],[0.0117497,0.692804,0.721009,0.00547891],[0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,fine,Pinza_clase;
        MoveL
[[500.87-resolucion,-80,240.17],[0.0117497,0.692804,0.721009,0.00547891],[0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,fine,Pinza_clase;
        resolucion:=resolucion+2;
        MoveL
[[500.87-resolucion,-80,240.17],[0.0117497,0.692804,0.721009,0.00547891],[0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,fine,Pinza_clase;
        MoveL
[[500.87-resolucion,80,240.17],[0.0117497,0.692804,0.721009,0.00547891],[0,-1,0,0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]],v10,fine,Pinza_clase;
        resolucion:=resolucion+2;
      ENDFOR
    ENDPROC

    PROC F_Write_OPC_TCP()
      TPWrite "Inicio F_Write_OPC_Joint";

!       Eliminamos una interrupción
      IDelete IWriteOPC;
!       Conectamos la interrupción con la rutina TRAP "rite_OPC_TCP.
      CONNECT IWriteOPC WITH Write_OPC_TCP;
!       solicitar y activamos la interrupción cada 0.1 segundos
      ITimer 0.1,IWriteOPC;
    ENDPROC

    TRAP Write_OPC_TCP
      VAR pos tcp;

!       Amacenamos en la variable "tcp" de tipo pos el TCP de la pinza
```

```
        tcp:=CPos(\Tool:=Pinza_clase\WObj:=wobj0);

!       Actualizamos las variables del controlador "Tcp_x" y "Tcp_y"
        SetAO Tcp_x,tcp.x;
        SetAO Tcp_y,tcp.y;
    ENDTRAP

ENDMODULE
```

Anexo F.2: Código Matlab

```
%% Borrar y limpiar
% Borramos todas las variables y limpiamos la pantalla
clear all
close all
clc


%% Conectarse a OPC

%N° de sensores
N=1;
% Las variables que queremos leer o escribir, deben estar
%creadas en el controlador y tener el mismo nombre.


names= {'Tcp_x', 'Tcp_y'};

opcreset;
j=1;

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

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



%% Abrir puerto serie
delete(instrfind({'Port'},{'COM3'}));
puerto_serial=serial('COM3');
puerto_serial.BaudRate=9600;
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');

fopen(puerto_serial);
```

```
%% Leer OPC
 while true,
%      Leemos las dos variables (Tcp_x, Tcp_y) mediante OPC y las
%      almacenamos en la variable de Matlab q
     for i= 1:N,
          w_item= read(SenItem{i});
         q(i)= w_item.Value;
     end

%     Si el sensor de ultrasonidos detecta un objeto, almacenamos
los datos de la coordenada x,y,z en una matriz

fwrite(puerto_serial,'A','uchar');
distancia = fscanf(puerto_serial);
distancia_double=str2num(distancia);


    if distancia_double<150
      P(j,1)=q(3);
      P(j,2)=q(2);
      P(j,3)=150-fscanf(puerto_serial);

      j=j+1;
   end
   pause(0.1);
 end


  %% Dibujar 3D con mayado triangular

    P=double(P);

%   Creamos una triangulación 3D de los puntos obtenidos
anteriormente
    DT = delaunayTriangulation(P);

%   Utiliza la triangulación 3-D de Delaunay para representar los
tetraedros
    faceColor  = [0.6875 0.8750 0.8984];
    figure
    tetramesh(DT,'FaceColor',faceColor,'FaceAlpha',0.3);



  %% Cerrar puerto Serial y eliminar las variables
```

```
fclose(puerto_serial);
delete(puerto_serial);
clear all;
```