Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Simulación de un Robot ABB IRB 120 para la detección por rastreo de piezas y posterior emplazamiento llevado a cabo por Robot ABB CRB 15000 GoFa. Implementación de protocolo OPC UA para la comunicación entre ambos y diseño de Interfaz gráfica desde MATLAB.

Autor:

Ruiz Luengo, Jorge

Tutor:

Herreros López, Alberto

Departamento de Ingeniería de Sistemas y Automática

Valladolid, julio, 2023

Resumen

El siguiente trabajo consiste en la programación de dos robots, utilizando el software de simulación RobotStudio. Uno de ellos tiene el objetivo de detectar la orientación de una serie de piezas con formas geométricas distintas, ubicadas sobre una mesa giratoria circular. El segundo robot empleado, coloca dichas piezas en los huecos correspondientes localizados en otra mesa similar.

Haciendo uso de una Interfaz Gráfica que se ha creado y desarrollado en MATLAB, será el usuario quien pueda llevar a cabo la simulación.

Para poder comunicar ambas aplicaciones entre sí y transferir datos, se implementa un modelo de comunicación del tipo Cliente-Servidor mediante el protocolo OPC UA.

Palabras clave

Robot IRB120, Robot CRB15000 GoFa, RobotStudio, Interfaz Gráfica, MATLAB, Cliente-Servidor, OPC UA.

Summary

The following project is based on two robots programming, by using the simulation software RobotStudio. The aim of one of them is to detect the orientation of a set of different geometric pieces, that are situated in a swivel table. The second robot place theese pieces into the appropiate hole located in another similar table.

Using a created and developed MATLAB graphic interface, the user can carry out the simulation process.

To communicate both softwares and transfer data, is going to be implement a Client-Server communication model, through the OPC UA protocol.

Key Words

Robot IRB120, Robot CRB15000 GoFa, RobotStudio, Interfaz Gráfica, MATLAB, Cliente-Servidor, OPC UA.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

Introducción y objetivos

1.1. Introducción

El siguiente proyecto alberga la creación de una estación virtual de trabajo, haciendo uso del software de ABB, RobotStudio, con el fin de simular la interacción entre dos robots y los distintos elementos creados, que se encuentran ubicados en el mismo entorno.

Ambos robots emplean controladores diferentes, así como diversos parámetros de configuración. Su correspondiente estudio y evaluación será uno de los objetivos principales.

Con el fin de dotar de una mayor versatilidad a nuestro trabajo, se ha diseñado una Interfaz Gráfica en MATLAB con la que el usuario puede realizar la simulación. Se hace uso de un protocolo de comunicación Cliente-Servidor que permite la lectura y escritura de las diferentes variables que utilizan ambos robots en el programa RAPID.

Cabe destacar la importancia de la coordinación y sincronización de ambos robots a la hora de ejecutar los distintos movimientos.

Así mismo, se intentan aplicar los conocimientos de distintas ramas de la ingeniería de una forma más creativa o entretenida.

1.2. Objetivos

Uno de los principales objetivos es aplicar y profundizar en los conocimientos de programación en el lenguaje RAPID. Se ha utilizado el software de simulación RobotStudio para crear un entorno virtual de trabajo donde los robots utilizados en el presente proyecto puedan actuar e interaccionar entre ellos. Véase Figura 1.

Seguidamente, para completar nuestra área de trabajo se han diseñado las herramientas, mecanismos y objetos necesarios para llevar a cabo dicha simulación.

Estudio y análisis de los controladores empleados para cada robot. Se citan las ventajas e inconvenientes del controlador empleado para el robot CRB 15000 GoFa. La universidad de Valladolid pretende comprar un ejemplar en los próximos años, siendo este proyecto un estudio y análisis previo.

Cabe mencionar la aplicación de diversos conocimientos adquiridos durante el aprendizaje y la formación en el grado de Electrónica, Industrial y Automática para la creación de un algoritmo que detecte la orientación de un tipo de pieza generada aleatoriamente que se encuentra situada sobre una mesa giratoria.

Otro de los objetivos a tener en cuenta consiste en la creación e implementación de una Interfaz Gráfica en MATLAB que permita al usuario ejecutar la simulación y visualizar los resultados. Para poder transferir información de una manera eficaz y sencilla entre dicha interfaz y el software de ABB, se ha utilizado un protocolo de comunicación OPC UA.

Para la realización de este proyecto se han empleado los robots de la compañía ABB Robotics que se muestran a continuación (Figura 1).

Figura 1. A la izquierda, Robot ABB IRB 120 [1]. A la derecha, Robot ABB CRB 15000 GoFa [2].

1.3. Estructura de la memoria

El contenido del proyecto se ha organizado en:

- Capítulo 1: Introducción y objetivos.

  Se muestran los objetivos e intereses de la realización de este proyecto.

- Capítulo 2: Marco teórico y estado del arte.

  Análisis breve y recapitulación de información relacionada con el mundo de la robótica.

  Conceptos generales y clasificación de los diversos robots empleados en la actualidad.

  También se muestran algunos proyectos elaborados por antiguos alumnos de esta universidad.

- Capítulo 3: Software empleado.

  Explicación y descripción de los distintos softwares empleados (RobotStudio, MATLAB, ABB IRC5 OPC).

- Capítulo 4: Desarrollo del proyecto.

  Estudio y explicación detallada de las distintas partes del proyecto (área de trabajo, herramientas, controladores, código RAPID, código MATLAB, Interfaz Gráfica).

- Capítulo 5: Funcionamiento y resultados.

  Se muestra el correcto funcionamiento de la aplicación desarrollada.

- Capítulo 6: Conclusiones y líneas futuras.

  Valoración final de los resultados obtenidos.

  Planteamiento de posibles líneas futuras del presente proyecto.

- Bibliografía.

  Referencias bibliográficas y fuentes utilizadas.

- Anexos.

  Códigos de programación empleados (Contenido aparte).

CAPÍTULO 2

Marco teórico y estado del arte

2.1. Marco teórico

A lo largo de la historia, el ser humano siempre ha sentido la necesidad de construir máquinas con el propósito de hacer su vida mucho más cómoda, satisfacer sus necesidades y a la vez ahorrar y facilitar el trabajo diario. En muchos otros casos, el afán de aprender o el simple entretenimiento han supuesto la creación de dispositivos que hoy en día conocemos como autómatas [3].

En el año 400 A.C. el matemático y filósofo italiano Arquitas de Tarento creó una paloma hueca de madera, suspendida de una cuerda, que almacenaba en su interior un depósito de agua. Esta se calentaba mediante una llama situada en la parte inferior del depósito y tenía el objetivo de hacer salir el vapor de agua por un orificio ubicado en la parte inferior y lograr así mover el pájaro. Se trata de la primera aplicación del principio físico de acción-reacción [4].

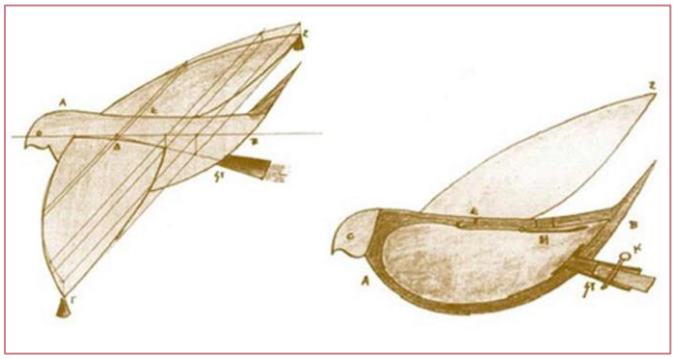

Figura 2. Arquitas y su paloma voladora [5].

Entrados ya en el nacimiento de Cristo, el matemático y científico griego Herón de Alejandría, construyó varios autómatas con formas de ave [4].

Haciendo un salto en el tiempo, en el siglo XIII, podemos destacar el gallo del reloj de la catedral de Estrasburgo. Este animal movía el pico y las alas al dar las horas. Se trata del autómata más antiguo que se conserva actualmente [4].

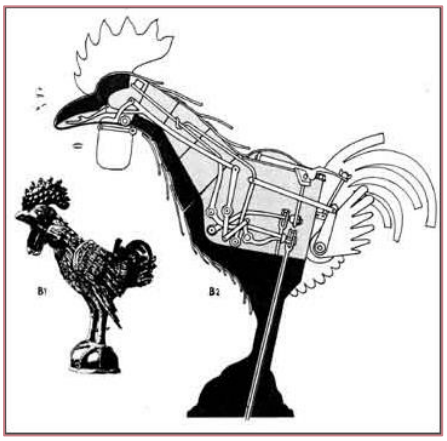

Figura 3. Gallo de Estrasburgo [6].

Otro ejemplo de autómata es el león mecánico de tamaño real, construido por Leonardo Da Vinci para el rey Luís XII de Francia, el cual se abría el pecho con su garra y enseñaba el escudo de armas del rey [4].

Figura 4. León mecánico de Leonardo Da Vinci restaurado [7].

En 1921, el escritor checo Karel Capek estrenó su obra Rossum's Universal Robot (R.U.R), usando así la palabra robot por primera vez. El origen etimológico de esta proviene a su vez de la palabra eslava 'Robota', que significa trabajo forzado. En la obra citada anteriormente, los robots eran máquinas androides que habían sido fabricadas por un científico llamado Rossum y a su vez servían para realizar los distintos trabajos físicos que debían desempeñar los jefes humanos [8].

Actualmente, todas las asociaciones o federaciones de robótica emplean su propia definición, aunque con evidentes analogías. La de la Federación Internacional de Robótica (IFR), es: " *Se entiende por robot industrial de manipulación a toda máquina de manipulación automática, reprogramable y multifuncional que conste de tres o más ejes que permitan orientar y/o posicionar piezas, herramientas o dispositivos con el fin de ejecutar trabajos diversos en las diversas etapas de la producción industrial* " [3].

En 1945, el mayor impulsor de la palabra robot, Isaac Asimov, publicó una novela en la revista Galaxy Science Fiction, donde enunciaba las "Tres Leyes de la Robótica". Su origen es ficticio, pero actualmente, se tienen en cuenta en este campo [8].

	TRES LEYES DE LA ROBÓTICA
PRIMERA	Un robot nunca ha de perjudicar a un ser humano ni permitir que este sufra daño con su inacción.
SEGUNDA	Un robot ha de cumplir órdenes del ser humano a excepción de aquellas que entren en conflicto con la primera ley.
TERCERA	Un robot ha de proteger siempre su existencia salvo que vaya en contra de la primera o la segunda ley.

Tabla 1. Leyes de la robótica según Isaac Asimov.

La robótica industrial que se conoce hoy en día se inspira en muchos de los autómatas descritos anteriormente. Fue en 1954 cuando George C. Devol diseña y crea el primer robot programable llamado 'Unimate', solicitando de esta forma una patente para lo que él denominó 'Dispositivos de transferencia

programada de artículos'. Dicho robot o manipulador constaba de interruptores de fin de carrera, levas y motores hidráulicos. Años más tarde, intentó vender su producto a numerosas industrias, pero no lo logró [4].

El estadounidense Joseph Engelberger, en 1960, adquiere la patente de Devol y forma su propia compañía llamada 'Unimation'. Tras mucho esfuerzo y sacrificio, consigue sobreponerse y llega a triunfar en el mercado industrial mediante la venta de estos robots. Es por ello que se le conoce como "El padre de la robótica". Seguidamente, se instala el primer ejemplar del robot 'Unimate' en la fábrica de automóviles estadounidense "General Motors" [4].

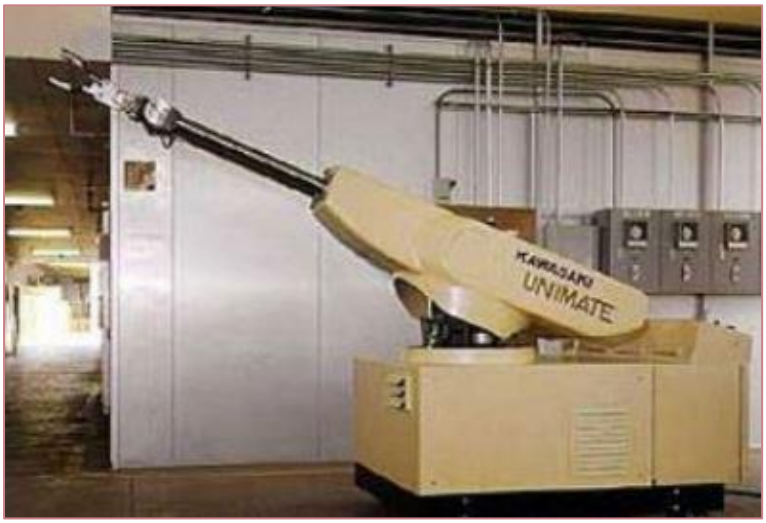

Figura 5. Robot 'Unimate' [4].

En la década de los 70, se percibe notablemente el gran crecimiento y auge de la robótica. Es por ello que se empiezan a crear las primeras asociaciones como son la denominada "JIRA" (Asociación de Robótica Industrial de Japón) o la "RIA" en América (Asociación de Robótica Industrial) [4].

Empresas de diversos lugares del planeta comienzan a desarrollar nuevos robots, implementando por primera vez el control por computador o haciendo uso de microprocesadores [4].

En 1980 se funda "IFR" (Federación Internacional de Robótica). Cinco años más tarde, se crea en España la "Asociación Española de Robótica" (AER). En este momento, se puede afirmar que las bases y los principios de la robótica han sido establecidos e integrados en el entorno industrial, dotándolo así de una versatilidad inmensa a la hora de poder realizar cualquier tipo de tarea, por muy compleja que resulte [4].

2.2. Conceptos generales

	CONCEPTOS GENERALES
TCP	Se define así el punto central de la herramienta del robot.
Área de trabajo	Hace referencia al volumen espacial al que puede acceder el robot. La herramienta no se debe tener en cuenta.
GDL	Todos y cada uno de los movimientos independientes que pueden realizar cada una de las articulaciones del robot respecto a la anterior.
Capacidad carga	Carga con la que puede operar y trabajar el robot. Se deben tener en cuenta ciertos parámetros como pueden ser el peso de la herramienta o los momentos de inercia entre muchos otros.
Precisión	Distancia real entre el punto alcanzado y el punto programado.
Repetibilidad	Programado y definido un punto según unas características específicas, radio de la esfera que contiene todos los puntos alcanzados tras realizar diversas simulaciones.
Resolución	Corresponde al mínimo incremento que es capaz de aceptar la unidad de control del robot.
Velocidad	Como su nombre indica, se trata de la velocidad de movimiento del robot. Destacar la velocidad por articulaciones o bien la velocidad media en el extremo.
Puntos singulares	Puntos del volumen espacial donde el robot no puede realizar una trayectoria rectilínea.

Tabla 2. Conceptos generales de un robot [4].

2.3. Clasificación y tipos de robots

Existen múltiples clasificaciones de los robots. A continuación, se indican algunas de ellas:

SEGÚN SU CRONOLOGÍA	
1ª Generación	Realizan y repiten una o varias tareas de manera programada y secuencial. No tienen en cuenta las alteraciones presentes en su entorno. Funcionan en Lazo Abierto, sin retroalimentación.
2ª Generación	Realizan tareas una vez aprendidos los movimientos que ejecutan previamente los humanos. Son los primeros en incluir retroalimentación. Utilizan sistemas de control en Lazo Cerrado para percibir el entorno.
3ª Generación	Emplean lenguajes de programación. Hacen uso de computadoras y sensores artificiales de visión y tacto.
4ª Generación	Pueden participar en diversos procesos gracias al crecimiento exponencial de la informática y la electrónica. A diferencia de la generación anterior, pueden tomar decisiones más complejas e inmediatas y realizar movimientos libremente.
5ª Generación	Incluye las máquinas más equipadas con inteligencia artificial, capaces de imitar comportamientos y pensamientos de los seres humanos.

Tabla 3. Clasificación de los robots según su cronología [9], [10] y [11].

	SEGÚN SU FUNCIÓN
Industriales	Intervienen en las distintas fases de la producción industrial.
De Servicios	Se utilizan en entornos no controlados, incluso hostiles.
Militares	Asisten a los ejércitos en intervenciones o acciones específicas.
Médicos	Ayudan a los cirujanos en sus operaciones con movimientos de alta precisión.
Educativos	Su principal objetivo es el de transmitir conocimientos al usuario.

Tabla 4. Clasificación de los robots según su función [11].

	SEGÚN SU ESTRUCTURA
Antropomórficos	Disponen de dos articulaciones. Una de ellas para el eje vertical y otra para elevar la articulación. Pueden incorporar una articulación extra en la muñeca del robot.
Poliarticulados	Existen diversas formas y configuraciones. Sin embargo, ofrecen desplazamientos limitados y se suelen mantener fijos.
Móviles	Presentan grandes ventajas a la hora de realizar los desplazamientos. Suelen incorporar diferentes plataformas con un sistema locomotor.
Androides	Intentan modelar de forma total o parcial el comportamiento de los seres humanos.
Zoomórficos	Basan su sistema locomotor en características y aspectos de determinados seres vivos.

Híbridos	Cualquier otro tipo o modelo de robot que no coincida con las especificaciones indicadas anteriormente.

Tabla 5. Clasificación de los robots según su morfología [12].

2.4. Estado del arte

Se muestran algunos de los proyectos de robótica realizados por antiguos alumnos del grado de Electrónica, Industrial y Automática en la Universidad de Valladolid.

En 2015, Juan Antonio Ávila Herrero diseñó para su trabajo de fin de grado una célula robótica, tanto real como simulada, con fines educativos en la asignatura de Sistemas Robotizados. Estaba compuesta por un robot ABB y otros elementos adicionales como son una botonera que muestra las señales de entrada-salida, una caja de rotuladores, dos mesas diferentes sobre las que dibujar o una herramienta de trabajo como la pinza para mover diferentes objetos [13].

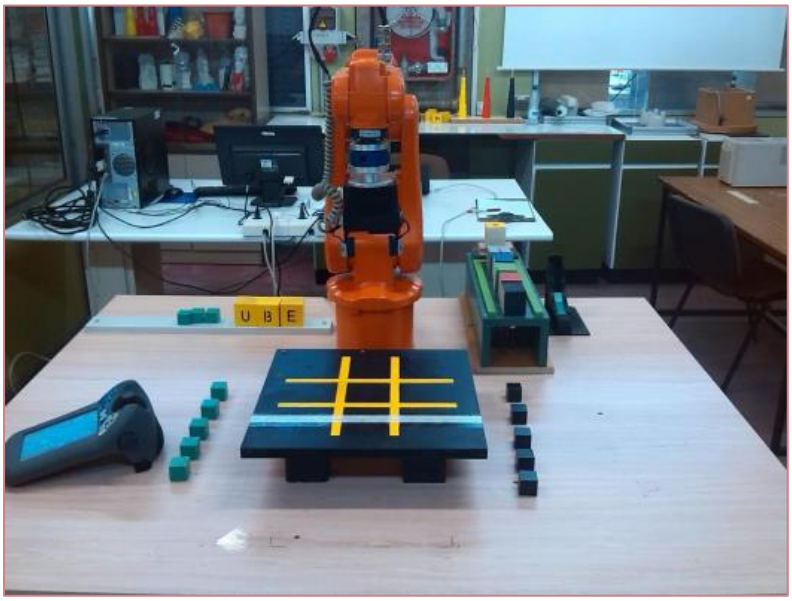

Figura 6. Célula de trabajo. TFG de Juan Antonio Ávila Herrero [13].

El trabajo de fin de grado de Raúl Vidal del Cura, en el año 2022, constaba de una célula robótica basada en una cadena de montaje con cuatro robots ABB sincronizados entre ellos con el fin de crear una luminaria [14].

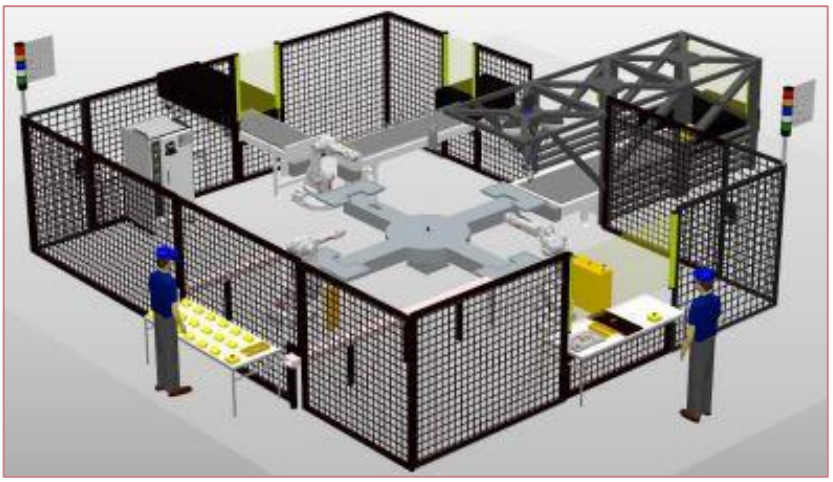

Figura 7. Cadena de montaje. TFG de Raúl Vidal del Cura [14].

En el mismo año, Elena Pozas Mata implementó en su trabajo de fin de grado la programación de un robot colaborativo multitarea ABB con el propósito de tocar un xilófono en base a las notas solicitadas por el usuario mediante una interfaz gráfica de usuario [15].

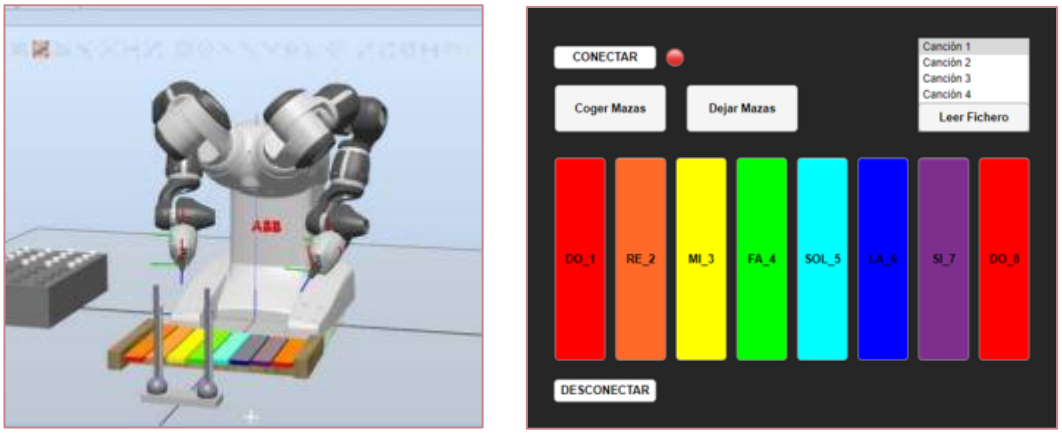

Figura 8. Célula de trabajo e Interfaz. TFG de Elena Pozas Mata [15] [16].

CAPÍTULO 3

Software empleado

Una de las finalidades de la realización de este proyecto era implementar los conocimientos de distintas ramas de la ingeniería. En nuestro caso, combinamos tres softwares, mostrados en la Figura 9, para lograr los objetivos y resultados propuestos.

Desde el software de RobotStudio, se va a realizar la programación y simulación de la célula robótica, compuesta por distintos robots y a su vez por diversos elementos de interacción creados.

MATLAB se utiliza para desarrollar e implementar la Interfaz Gráfica con la que el usuario puede configurar la estación de trabajo y ordenar la ejecución de los movimientos de los robots empleados.

Para poder comunicar ambos softwares, se hace uso de ABB IRC5 OPC, el cual utiliza una conexión de tipo OPC UA bajo un protocolo de comunicación cliente-servidor. El servidor en este caso es el propio ABB IRC5 OPC y existen dos clientes, RobotStudio Y MATLAB.

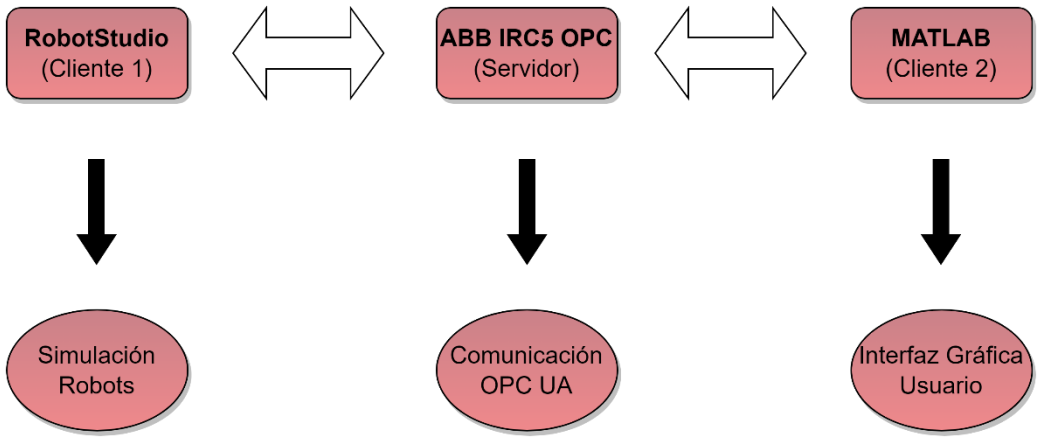

Figura 9. Esquema general y funcional del Software.

3.1. RobotStudio

Es una de las herramientas más utilizadas en el mundo de la robótica industrial hoy en día que permite configurar y programar los robots ABB, tanto a nivel físico como a nivel virtual [16].

Para poder realizar cualquier tipo de movimiento con un robot, es necesario el uso de un controlador (modo online para controladores de robots físicos y modo offline para controladores virtuales) [16].

Este software ofrece un entorno de trabajo muy cercano al de cualquier proceso o actividad llevada a cabo en el mundo real sin afectar a la producción en curso [17].

En el presente trabajo, haciendo uso de este software, se ha podido crear el área de trabajo correspondiente, así como configurar y establecer todos los parámetros necesarios para su posterior simulación.

Es importante mencionar el lenguaje de programación de alto nivel empleado para este software. Se denomina RAPID y es donde se ha llevado a cabo la programación de todos los movimientos e interrupciones de los robots.

3.2. MATLAB

Millones de ingenieros y científicos de todo el planeta utilizan esta plataforma de cálculo numérico y programación para realizar el análisis de datos, desarrollar algoritmos o bien crear modelos [18].

En nuestro caso, el uso de este software nos ha permitido realizar diversos algoritmos. Uno de ellos para leer datos de un fichero y almacenarlos de una manera concreta en una matriz. Otro para reconocer el objeto a identificar mediante el cálculo del área de una de sus superficies o podemos destacar también el algoritmo principal diseñado para detectar una orientación específica de un objeto en un volumen espacial determinado.

MATLAB incorpora una serie de herramientas muy útiles llamadas 'toolbox' que dotan al software de un mayor rango de aplicación. Se ha utilizado 'OPC Toolbox' para poder realizar el intercambio de variables a tiempo real (lectura y escritura) entre RobotStudio y este.

La Interfaz Gráfica ha sido diseñada haciendo uso de otra aplicación conocida como 'AppDesigner'. Esta herramienta genera automáticamente el código una vez el usuario ha seleccionado el elemento a incluir en la interfaz.

3.3. ABB IRC5 OPC

Antes de explicar y analizar la aplicación empleada, hay que aclarar algunos conceptos básicos de las comunicaciones industriales.

OPC es un protocolo de comunicación abierto que permite la transmisión de datos entre distintos sistemas de control industrial. Al tratarse de un estándar abierto, supone menores costes para los fabricantes y mayores opciones para los usuarios [19].

El intercambio de datos puede llegar a ser un desafío entre dispositivos de distintos proveedores o que poseen características diferentes, ocasionando posibles fallos de vulnerabilidad. Sin embargo, se hace uso de OPC UA. Las siglas UA hacen referencia a una arquitectura unificada basada en la tecnología de comunicación TCP/IP, logrando de esta manera una comunicación más fluida y fiable propia de la Industria 4.0 [19] y [20].

Mencionar de nuevo el modelo cliente-servidor utilizado, comprendido por dos clientes en nuestro caso, que envían peticiones al servidor y este último les devuelve una respuesta. Esto permite la lectura y escritura de variables a tiempo real. La separación de responsabilidades o la gestión de información de forma centralizada son algunas de las características más comunes de esta arquitectura, haciendo que sea una de las más aplicadas en la actualidad.

Centrándonos en la propia aplicación, el software ABB IRC5 OPC ofrece la posibilidad de crear y coordinar alias para los controladores ABB IRC5 [20].

Un alias representa un interfaz de comunicaciones con un controlador ABB IRC5. Se ha de crear un alias por cada controlador empleado y estos accederán al servidor de datos ABB IRC5 OPC UA, permitiendo la transferencia de datos de máquina a máquina, de robots a máquinas o a sensores, al sistema de monitorización e incluso a la nube [20].

CAPÍTULO 4

Desarrollo del proyecto

El siguiente capítulo recoge todas las explicaciones del trabajo realizado.

4.1. RobotStudio

Este apartado comprende distintas secciones donde se indican los pasos llevados a cabo para modelar la estación, así como la configuración adoptada para cada uno de los controladores. También se muestra la realización de las distintas herramientas y mecanismos inteligentes y los puntos que forman las diversas trayectorias establecidas para ambos robots.

4.1.1. Modelado de la estación

Lo primero que se debe hacer es crear una estación vacía donde añadiremos los robots empleados para la realización del proyecto, así como los correspondientes controladores para el correcto funcionamiento de estos o los distintos componentes inteligentes con los que van a interactuar dichos robots.

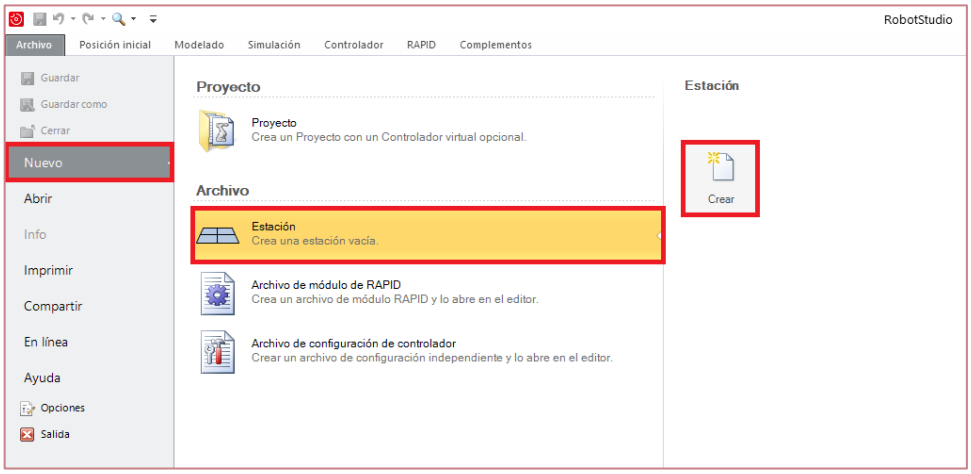

Figura 10. Creación de la estación vacía.

Seguidamente, importamos los robots desde la biblioteca ABB. En nuestro caso hay que seleccionar el robot ABB IRB 120 y el robot ABB CRB 15000 GoFa. Véase en la Figura 11 y Figura 12.

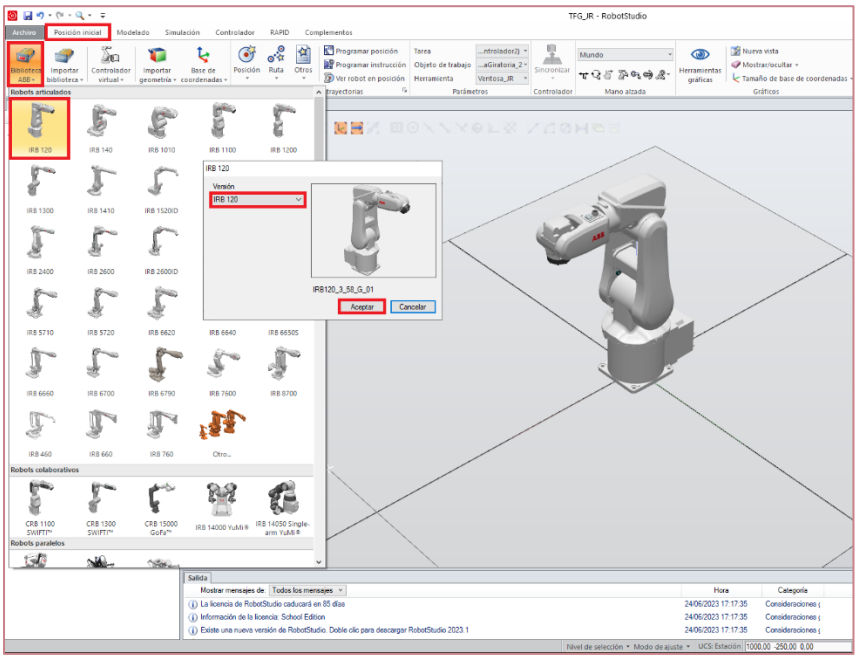

Figura 11. Importación robot ABB IRB 120.

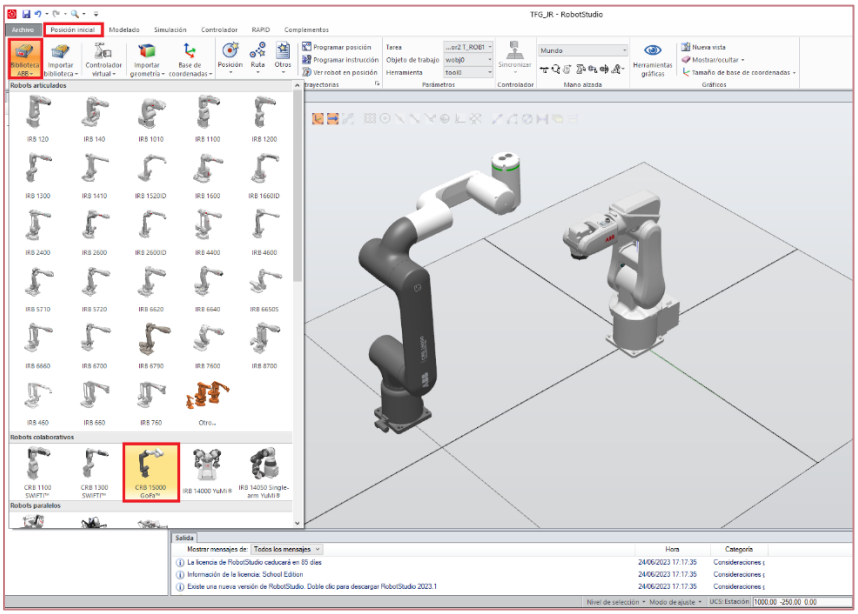

Figura 12. Importación robot ABB CRB 15000 GoFa.

Para fijar la posición de cualquier elemento en la estación de RobotStudio, en este caso la posición de ambos robots, debemos pulsar en “Posición” y “Fijar Posición”.

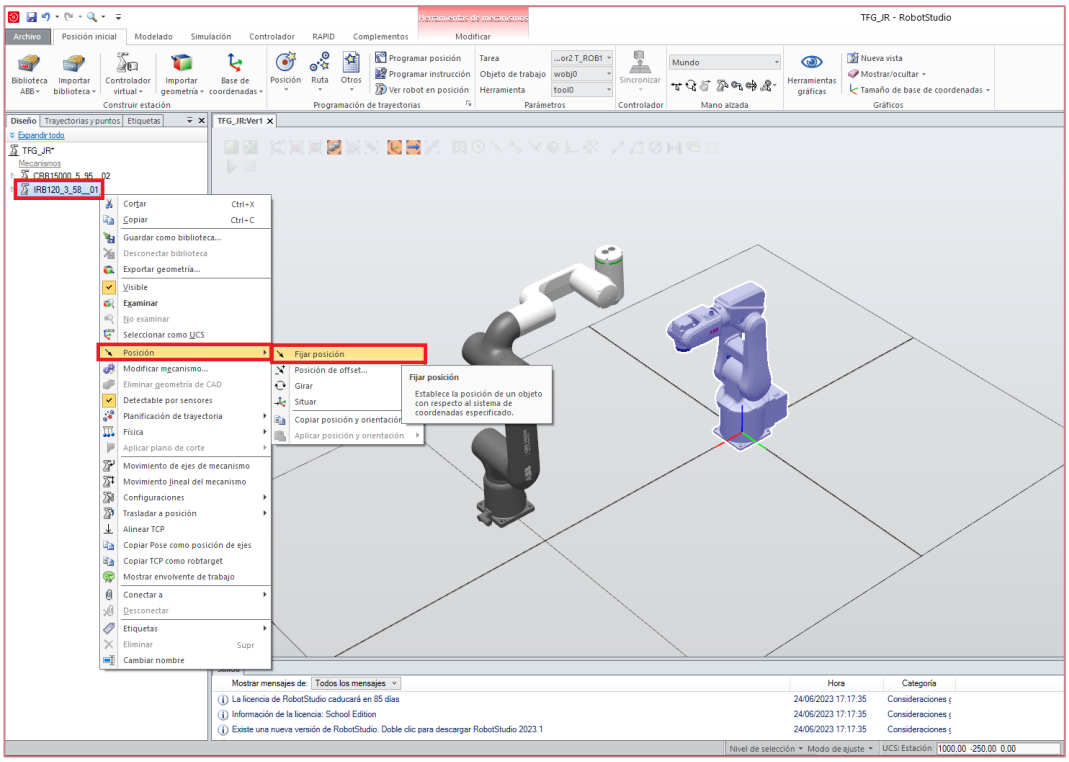

Figura 13. Fijar posición del robot en la estación.

A continuación, se muestran las coordenadas de emplazamiento de ambos robots respecto al sistema de referencia mundo, situado en el (0,0,0).

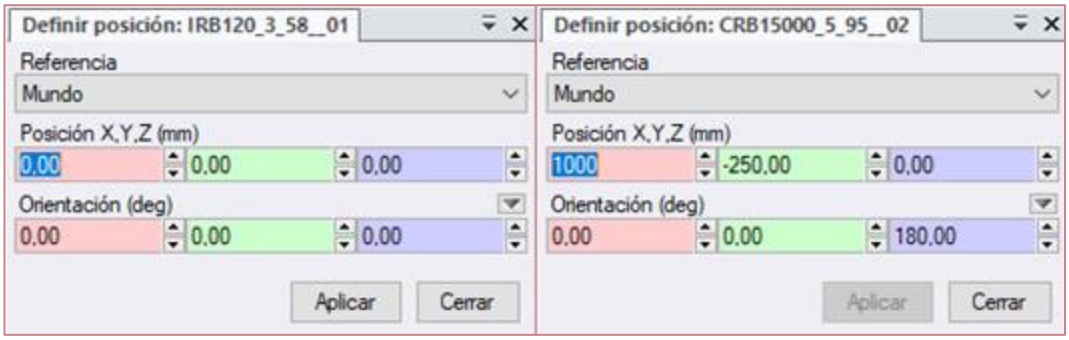

Figura 14. Posiciones de los dos robots.

4.1.2. Configuración de los controladores

El siguiente paso que se debe realizar es la creación de los controladores correspondientes. Dado que nuestro proyecto consta de dos robots diferentes, debemos asignar uno por cada robot.

Para crear un controlador, se debe pulsar en "Controlador virtual" y seguidamente en la pestaña de "Desde diseño". Véase a continuación:

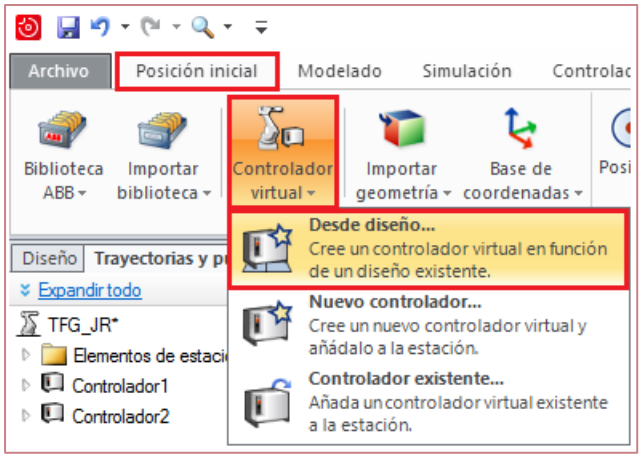

Figura 15. Creación de controlador virtual desde diseño.

A continuación, se citan las opciones de configuración adoptadas para cada controlador.

CONFIGURACIÓN CONTROLADOR 1
Robot IRB 120
Sistema Operativo RobotWare 5.16
Advanced Shape Tuning (602-1)
World Zones (608-1)
Independent Axis (610-1)
Path Recovery (611-1)
Path Offset (612-1)
SoftMove (885-1)
Collision Detection (613-1)

FTP and NFS Client (614-1)
PC Interface (616-1)
Flexpendant Interface (617-1)
Multitasking (623-1)
Sensor Interface (628-1)
Robot Reference Interface (897-1)
Production Manager (812-1)

Tabla 6. Configuración Controlador IRB 120.

CONFIGURACIÓN CONTROLADOR 2
Robot CRB 15000 GoFa
Sistema Operativo RobotWare 7.18.1
World Zones (3106-1)
Independent Axis (3111-1)
Path Recovery (3113-1)
SoftMove (3108-1)
Collision Detection (3107-1)
FTP and NFS Client (3116-1)
NFS Client (3117-1)
Multitasking (3114-1)
Flexpendant Program Package (3151-1)
Flexpendant Essential App Package (3120-2)
Flexpendant Limited App Package (3120-1)

Tabla 7. Configuración Controlador CRB 15000 GoFa.

El siguiente paso que se debe realizar es crear las entradas y salidas digitales necesarias para el desarrollo del proyecto.

Para ello se pulsa en “Configuración” e “I/O”. Por otro lado, se debe seleccionar la opción “Signal” y de esta forma aparece el cuadro donde introduciremos el nombre de la señal a crear, así como el tipo de señal (entrada o salida digital en nuestro caso). Véase la Figura 16.

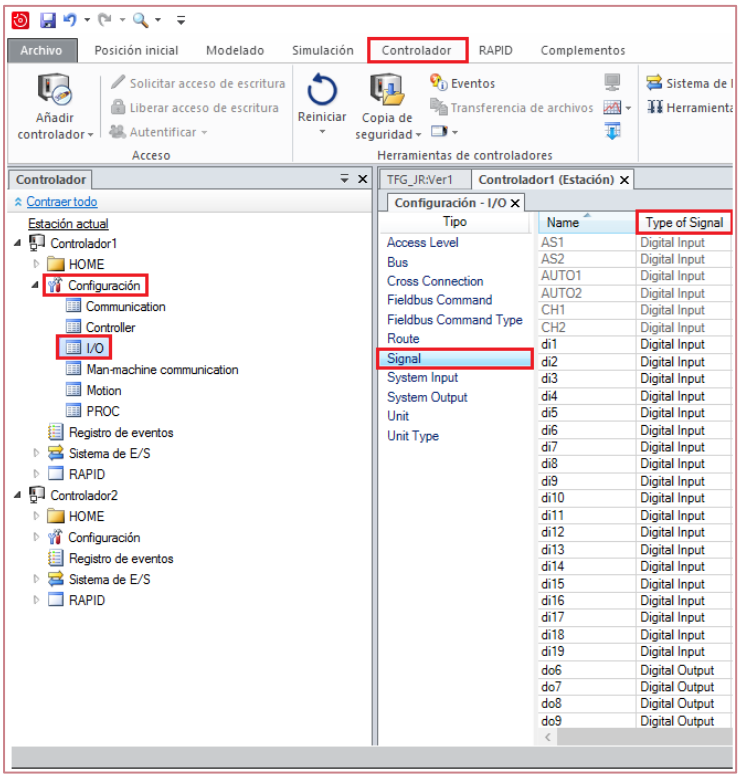

Figura 16. Creación de entradas y salidas digitales.

Llegados a este punto, se procede a explicar y analizar la Flexpendant empleada para el robot CRB 15000 GoFa (“OmniCore”).

Mencionar que dicha Flexpendant permite al usuario operar con el robot en modo manual. Sin embargo, también pueden programarse instrucciones de movimiento en el dispositivo citado para que sean ejecutadas.

La apariencia de OmniCore es muy similar a la de la Flexpendant del robot IRB 120 (“IRC5”). La disposición de todos los botones prácticamente se mantiene igual, siendo muy intuitiva y facilitando la operativa en todo momento.

El joystick de control sigue estando en el lado derecho. Las opciones que se pueden visualizar en la pestaña inicial son similares. Se pueden programar movimientos, crear señales de entrada y/o salida, así como calibrar los ejes del robot o visualizar mediante un puntero la parte del código donde se encuentra el robot ejecutándose, entre otras opciones.

La única diferencia a destacar reside en que la Flexpendant OmniCore utiliza una pantalla táctil de mayor tamaño y resolución, aportando comodidad en todo momento. Véase en la siguiente ilustración.

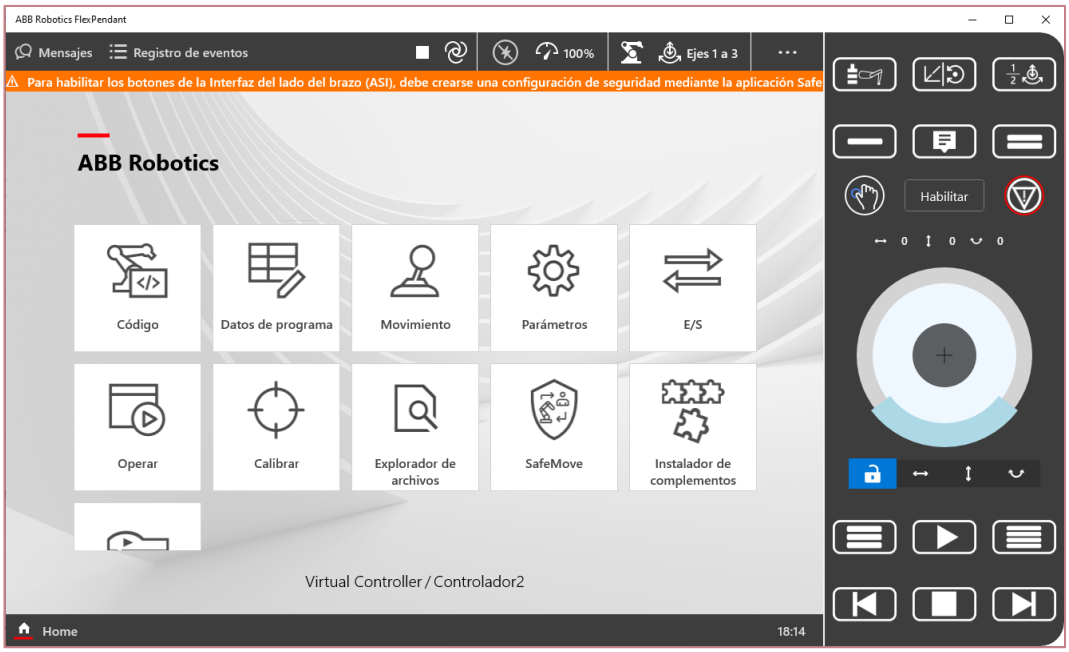

Figura 17. Flexpendant OmniCore robot CRB 15000 GoFa.

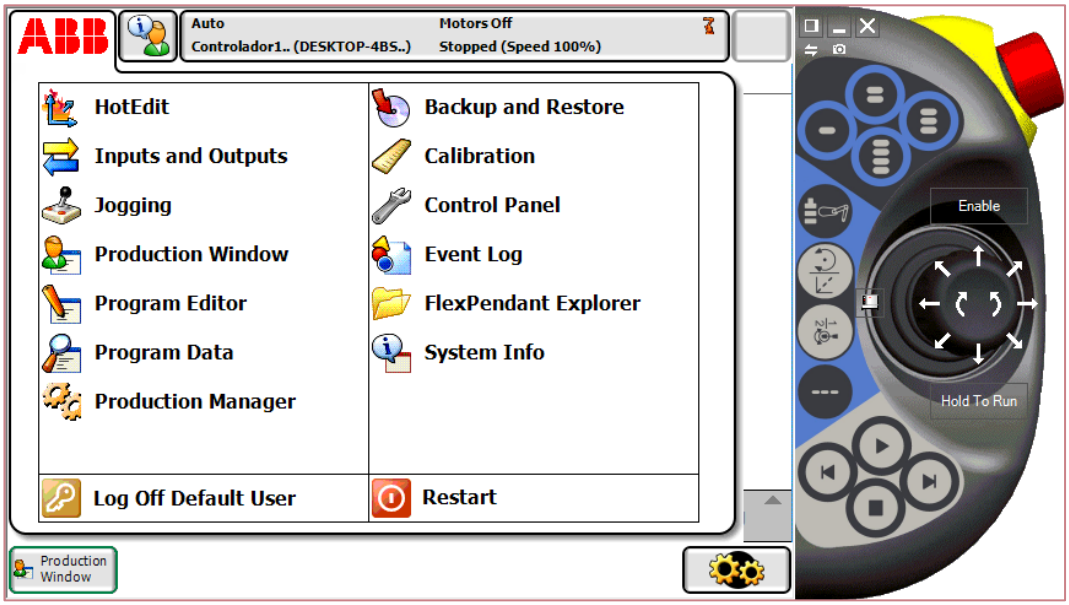

Figura 18. Flexpendant IRC5 robot IRB 120.

Otro factor que debemos mencionar es el amplio abanico de protocolos de comunicación industrial que pueden establecerse. Algunos de ellos son

Ethernet IP, Profibus, Profinet, Devicenet, o el usado por nosotros mismos, OPC UA entre otros.

Algunas características y ventajas que ofrece este nuevo controlador y su correspondiente Flexpendant son una mayor productividad debido a una mejora en el control del movimiento, un ahorro energético que reduce el consumo en hasta un 20%, así como mayor seguridad en la colaboración con seres humanos o una mayor conectividad con otros dispositivos. [21]

Un inconveniente a destacar es el parámetro de configuración "MultiMove". Este no es compatible entre ambos controladores empleados, por lo que no se pueden sincronizar los movimientos de ambos robots de esta forma tan sencilla. Para solventar el problema, se han configurado y sincronizado ambos controladores con señales de entrada y salida digitales, dotándole así a nuestro proyecto de una mayor complejidad.

4.1.3. Componentes Inteligentes

Nuestro trabajo consta de diversos elementos con los que los robots interaccionan. A continuación, se procede a explicar todos y cada uno de ellos.

Piezas

Como bien se ha explicado anteriormente, nuestro robot IRB 120 debe detectar una serie de piezas de distintas formas geométricas creadas. En el presente proyecto vamos a trabajar con tres tipos de piezas: Prisma Rectangular, Prisma Triangular y Prisma Pentagonal.

Las piezas mostradas en la Figura 20 han sido creadas pulsando en la pestaña "Curva" y seleccionando la opción "Polígono".

Se introducen los parámetros de diseño necesarios y se obtiene el contorno de la pieza a representar. Véase el ejemplo del contorno del pentágono en la Figura 19.

Seguidamente, se selecciona la opción de "Extrudir superficie", se indica la altura de la pieza, marcamos la pestaña de "Sólido" y finalmente pulsamos la pestaña "Crear".

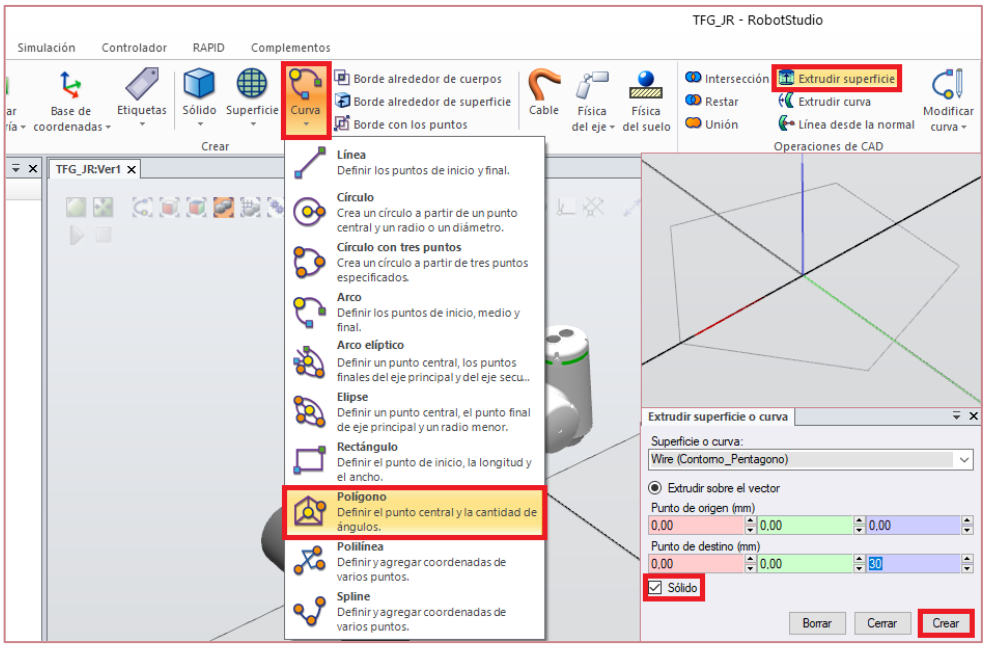

Figura 19. Pasos a seguir para el modelado de las piezas.

En la siguiente tabla, se recogen algunos de los parámetros de diseño de las piezas.

PARÁMETROS DE LAS PIEZAS	
P. Rectangular	Lado Mayor = 150 mm Lado Menor = 50 mm Altura = 30 mm
P. Triangular	Lado = 173,2 mm Mediana = 150 mm Altura = 30 mm
P. Pentagonal	Lado = 100 mm Apotema = 68,819 mm Altura = 30 mm

Tabla 8. Parámetros de diseño de las piezas.

El resultado final de los distintos prismas geométricos creados se muestra a continuación:

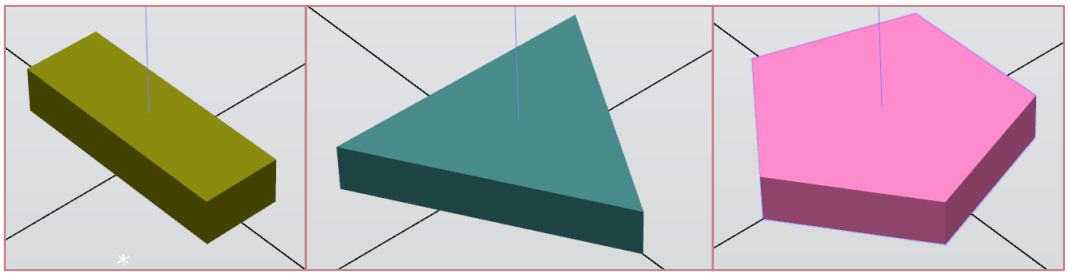

Figura 20. Piezas elaboradas.

Generador Aleatorio

Para poder conseguir crear las piezas de forma aleatoria, se ha creado un generador basado en los siguientes componentes:

- **Random** (Figura 21):

  Este bloque permite generar un número aleatorio ("Value") comprendido entre un mínimo y un máximo establecido ("Min" y "Max" respectivamente).

  Nuestro trabajo consta de tres piezas geométricas diferentes por lo que se van a generar números comprendidos entre el uno y el cuatro.

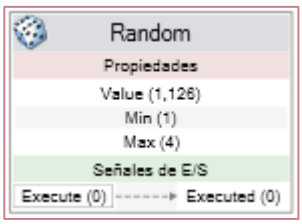

Figura 21. Componente Random Generador Aleatorio.

- **Comparer** (Figura 22):

  A continuación, vamos a clasificar el número aleatorio generado en función de la parte entera. Si esta es "1", corresponde al Prisma Rectangular. Si se trata de un número con parte entera igual a "2",

corresponde al Prisma Triangular y el caso restante hace referencia al Prisma Pentagonal.

En el apartado “ValueA” se muestra el número aleatorio generado. “ValueB” contiene el número a comparar. Haciendo uso de los operadores (“Operator”) podemos comparar ambos valores. En caso de que se cumpla la condición impuesta en el operador, se activará la señal “Output”.

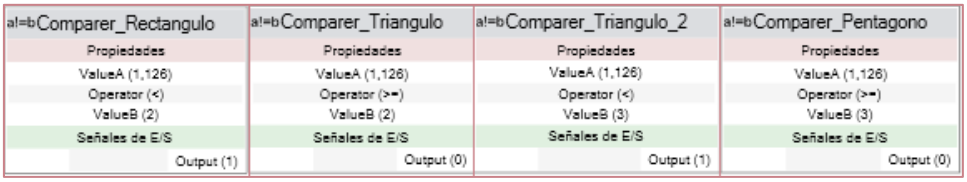

Figura 22. Componente Comparer Generador Aleatorio.

Cabe destacar el uso de dos comparadores en el caso del triángulo. Esto es debido a que necesitamos un número que sea mayor o igual a dos, pero estrictamente menor que tres. Cumpliéndose ambas condiciones, siempre vamos a tener un número aleatorio con parte entera de valor “2”.

Para que se cumplan las condiciones citadas anteriormente, debemos hacer uso de otro componente que se explica a continuación.

- **LogicGate [AND]** (Figura 23):

  Se trata de una puerta lógica que solo activa la señal de salida en caso de que se encuentren activas las dos entradas. (“InputA” e “InputB”).

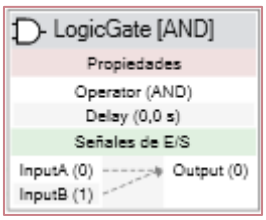

Figura 23. Componente LogicGate [AND] Generador Aleatorio.

- **Source** (Figura 24):

  Este componente se emplea para crear una copia de un componente gráfico siempre que la señal de entrada se encuentre activa. En nuestro caso se trata de los prismas creados con anterioridad.

  En "Source" se indica el tipo de pieza que se quiere generar. Seguidamente, se indican las coordenadas de emplazamiento de dicha pieza ("Position"). En el presente proyecto, todas las piezas se van a situar sobre el centro de la mesa.

  El resto de parámetros a configurar son los que vienen establecidos por defecto en RobotStudio.

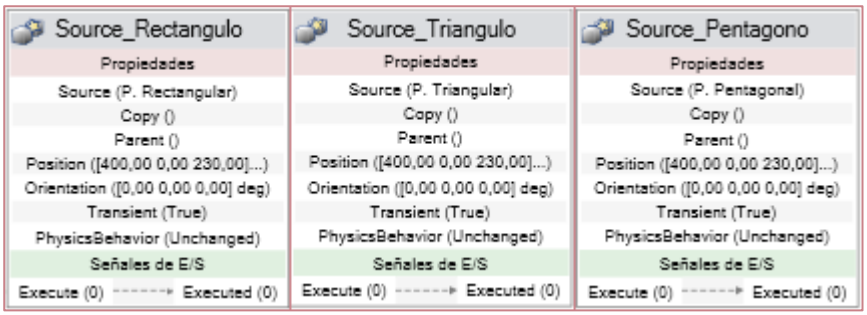

Figura 24. Componente Source Generador Aleatorio.

- **Sink** (Figura 25):

  Permite eliminar la copia del componente gráfico creado con el bloque Source. Esto permite eliminar las piezas al finalizar la simulación y de esta forma evitar que se acumulen en futuras simulaciones.

Figura 25. Componente Sink Generador Aleatorio.

Finalmente, en la figura 26, se muestran todas las conexiones realizadas para el correcto funcionamiento del generador de números.

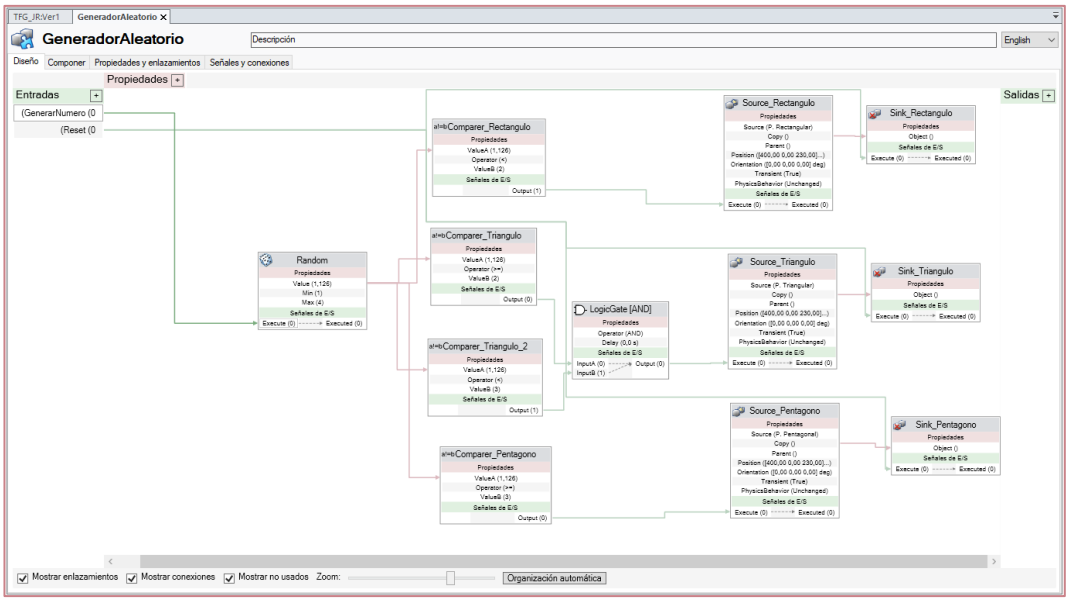

Figura 26. Conexiones del Generador Aleatorio.

Mesa Giratoria

El siguiente elemento del que se dispone es una mesa que hemos diseñado. Consta de dos piezas principales, el soporte o base y el tablero donde finalmente se colocarán las piezas.

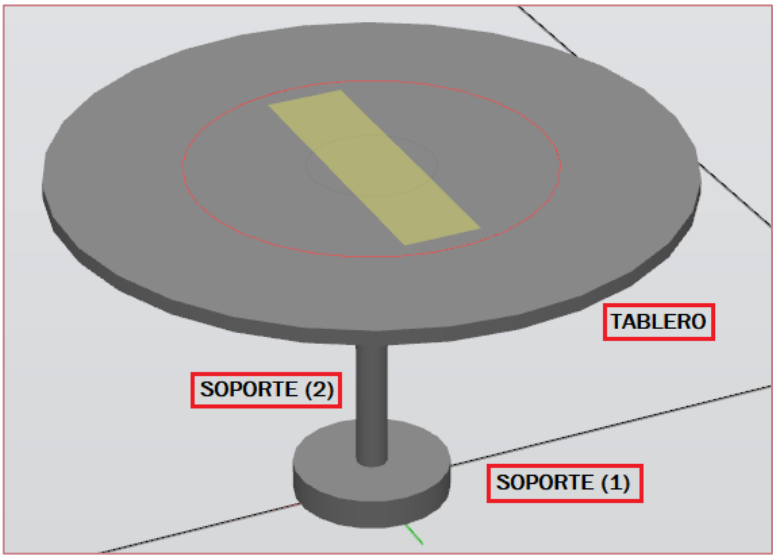

Figura 27. Mesa giratoria.

Para modelar los distintos cuerpos cilíndricos que componen la mesa, debemos seleccionar el tipo de sólido que se quiere crear. Para ello se debe hacer click en la pestaña de “Modelado” y “Sólido”.

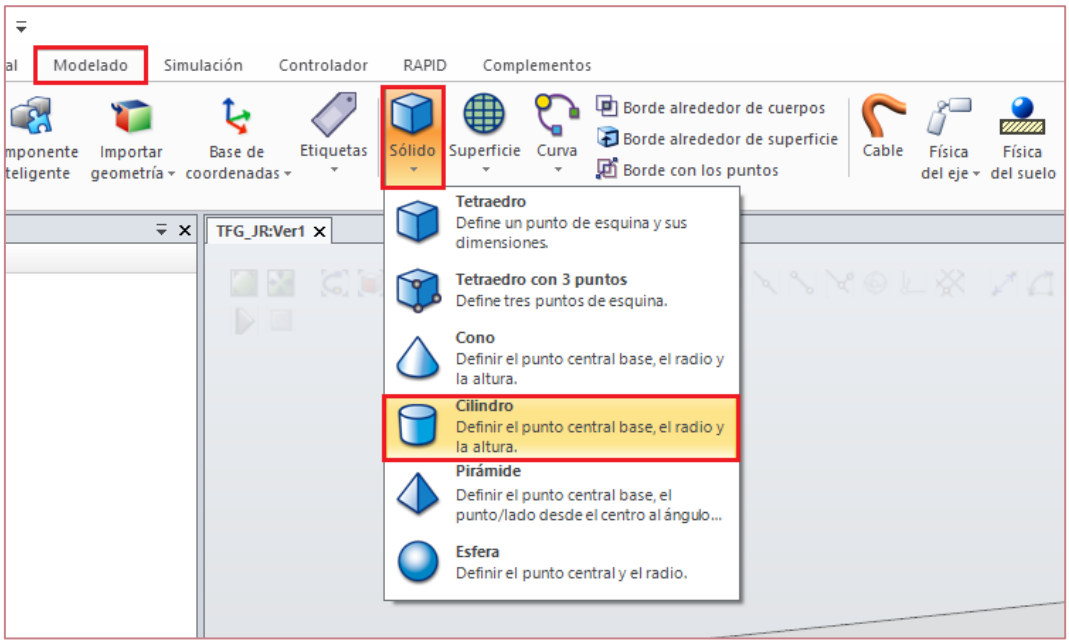

Figura 28. Modelado de un sólido en RobotStudio.

Véanse las medidas y características de diseño en la siguiente tabla:

PARÁMETROS DE LA MESA	
Soporte (1)	Radio = 50 mm Altura = 20 mm
Soporte (2)	Radio = 10 mm Altura = 200 mm
Tablero	Radio = 200 mm Altura = 10 mm

Tabla 9. Parámetros de diseño de la mesa giratoria.

Para lograr que la mesa gire, se debe crear en RobotStudio un mecanismo. A continuación, se detallan los pasos seguidos.

En primer lugar, seleccionamos la pestaña “Crear mecanismo”. Se introduce el nombre que se le quiere dar al mecanismo, en nuestro caso es una mesa giratoria; por último, se selecciona la pestaña “Dispositivo”.

El siguiente paso a realizar es configurar los eslabones que forman nuestra mesa; es decir, debemos añadir la base y el tablero diseñados y creados previamente. Se debe pulsar la pestaña “Añadir Eslabón…”.

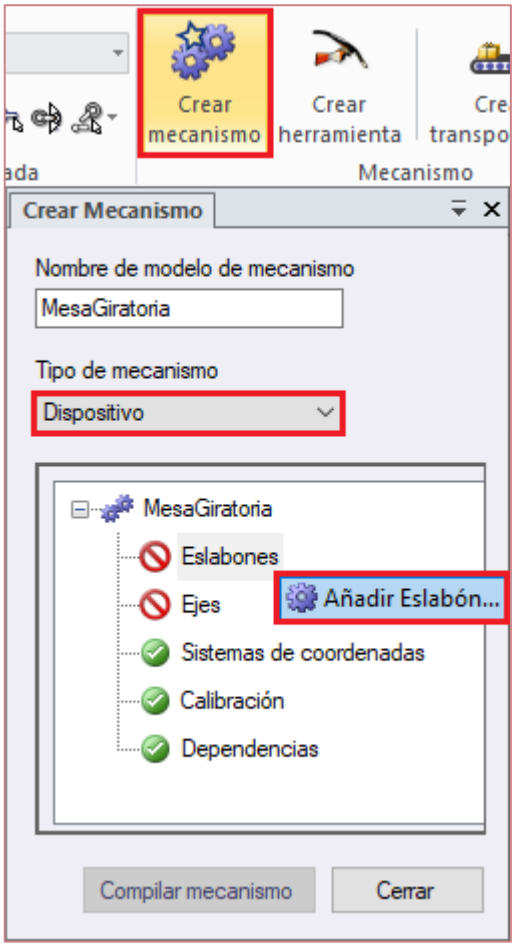

Figura 29. Añadir eslabón a un mecanismo.

En el lateral izquierdo de la Figura 30 se muestra la introducción de los eslabones. Para ello se debe pulsar sobre el “triángulo verde”.

Una vez se hayan seleccionado todos los componentes, estos aparecerán en el recuadro de “Componentes añadidos” como se indica en el lateral derecho de la Figura 30.

Finalmente, se debe pulsar la pestaña “Aplicar” y de esta forma estarían creados los dos eslabones de nuestro mecanismo.

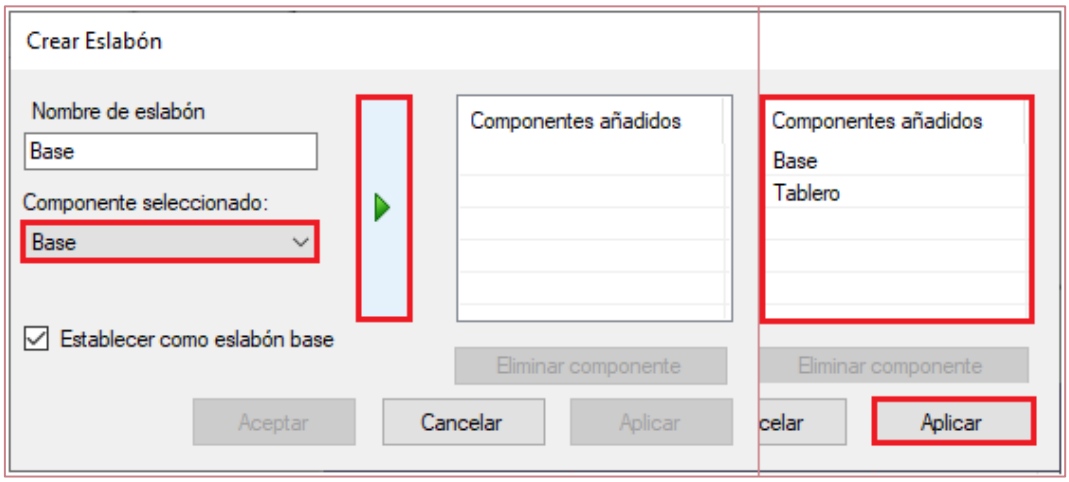

Figura 30. Creación de eslabones en RobotStudio.

El segundo paso a realizar es configurar el movimiento de rotación de la mesa. Pulsamos en “Añadir Eje…”. Véase la Figura 31.

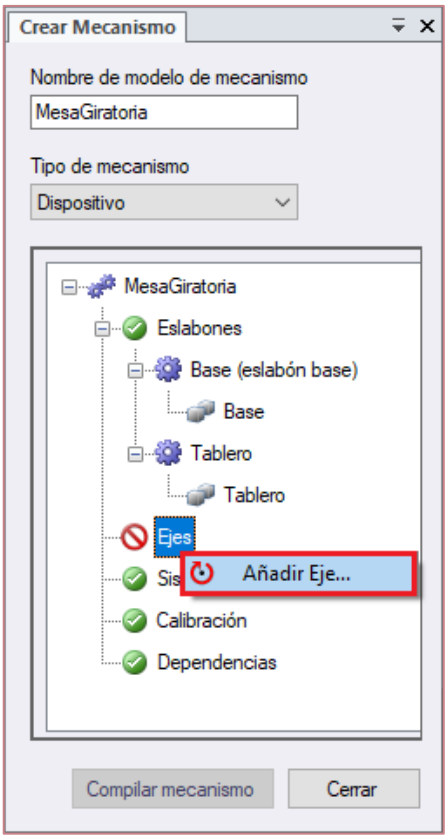

Figura 31. Añadir eje a un mecanismo.

La rotación de la mesa se debe realizar sobre el eje Z, poniendo un "1" en el apartado "Axis Direction". También se deben indicar las coordenadas de emplazamiento de dicho eje. En nuestro caso, el eje Z coincide con el centro de la mesa, situada a 400 mm (eje X) respecto del sistema de referencia mundo. Cabe destacar el ángulo de giro, siendo este de 360º. De esta forma la mesa puede completar una vuelta completa en sentido horario y otra en sentido antihorario.

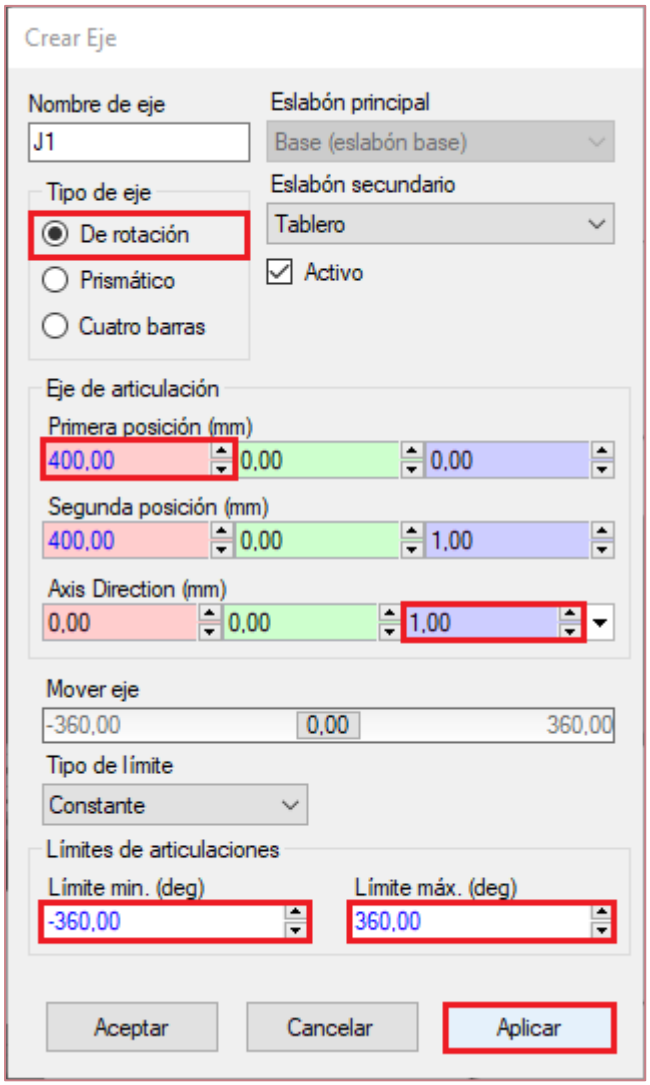

Figura 32. Configuración del eje Z del mecanismo.

En la Figura 33, se puede apreciar en color verde el eje de rotación creado para la mesa.

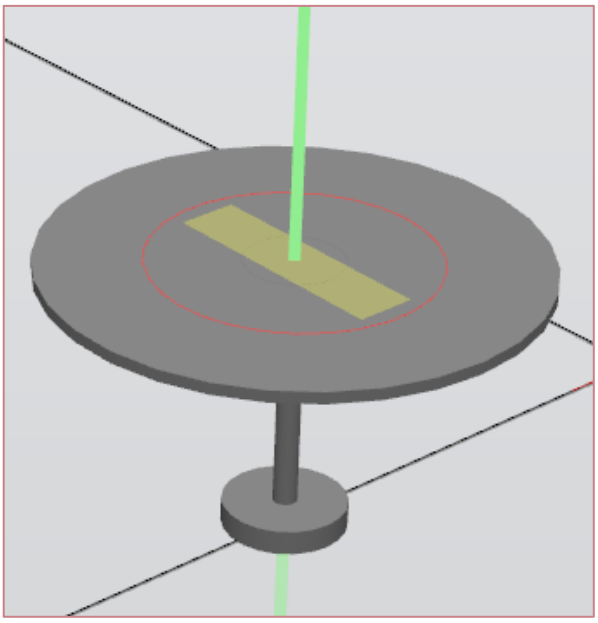

Figura 33. Eje de rotación del mecanismo.

Llegados a este punto, el modelado de la mesa giratoria ya estaría completado. Ahora se deben realizar las conexiones lógicas pertinentes para el correcto funcionamiento de la misma. Los componentes empleados son los siguientes:

- **PlaneSensor** (Figura 34):

  Se trata de un sensor plano definido por cuatro parámetros. El primero de ellos, "Origin", especifica el origen de creación de dicho plano. Los dos siguientes, "Axis1" y "Axis2" delimitan la longitud y la anchura del sensor que se va a crear. El apartado "SensedPart" permite detectar cualquier objeto que entre en contacto con el plano creado, poniendo a uno la señal de salida "SensorOut". Para ello, la señal "Active" debe estar previamente activada.

  En nuestro caso, es imprescindible el uso de este elemento para que la mesa pueda realizar el movimiento rotacional, dado que necesita que haya previamente una pieza colocada sobre este. En caso contrario, la mesa no podrá girar.

  Véase en la Figura 27 el sensor plano de color amarillo situado en la posición central de la mesa.

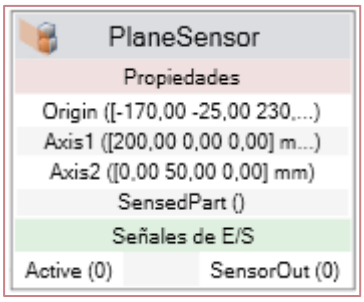

Figura 34. Componente PlaneSensor Mesa Giratoria.

- **Queue** (Figura 35):

  Este componente permite crear una "cola" para que la pieza detectada por el sensor plano permanezca a la espera hasta que el usuario decida hacer girar la mesa. Es por ello que "Enqueue" se activará cuando la señal "SensorOut" citada anteriormente se ponga a uno, logrando así poner la pieza creada en la cola.

  Para nuestro proyecto, el número máximo de piezas que puede haber situadas en la mesa; es decir, en la cola, es de uno. Se indica en el apartado "NumberOfObjects".

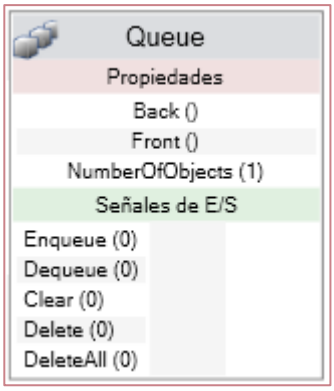

Figura 35. Componente Queue Mesa Giratoria.

- **MoveAlongCurve** (Figura 36):

  Permite realizar el movimiento de rotación de la mesa. Se debe indicar en el apartado "WirePart" la curva a lo largo de la cual queremos que se

mueva la pieza. En nuestro caso, hemos creado un círculo que puede apreciarse en la Figura 27.

En el apartado “Object” conectamos la pieza que se encuentra en la cola de espera.

Finalmente, para comenzar la ejecución del giro, “Execute” debe estar activada. Por el contrario, si el usuario desea parar la mesa, “Pause” deberá estar activada e inmediatamente la señal anterior se pondrá a cero.

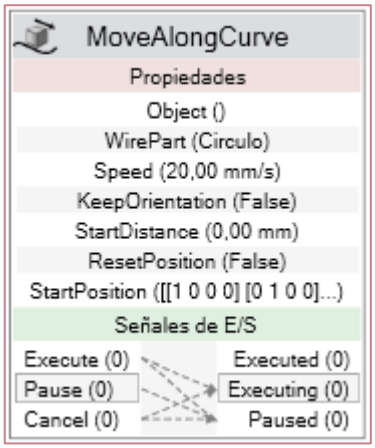

Figura 36. Componente MoveAlongCurve Mesa Giratoria.

En la Figura 37 se muestran todas las conexiones realizadas entre los diversos componentes, así como las entradas creadas.

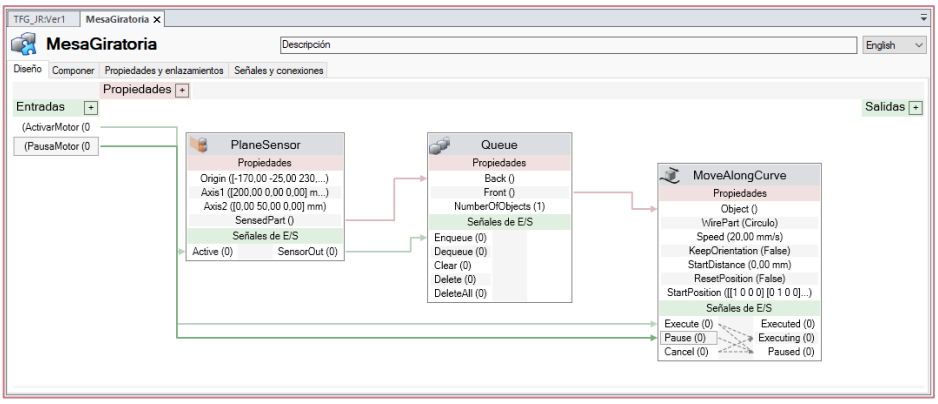

Figura 37. Conexiones de la Mesa Giratoria.

Mesa Figuras

Esta mesa es una copia de la mesa giratoria creada previamente, dado que tiene las mismas medidas. La única salvedad es que se han eliminado todos los componentes lógicos empleados.

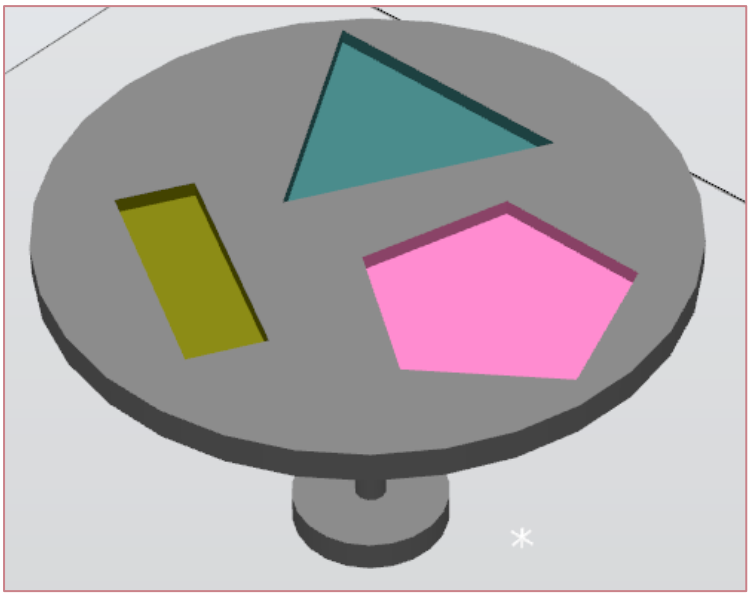

Figura 38. Mesa Figuras.

Para crear los huecos correspondientes a cada pieza se han seguido los siguientes pasos:

En primer lugar, se han incrustado 10 mm de la pieza en la mesa. Véase la parte izquierda de la Figura 39.

En la pestaña “Modelado”, se debe pulsar la opción de “Restar”. Seguidamente, se seleccionan las dos partes que se quieren restar como se muestra en la zona derecha de la ilustración.

Mencionar que este proceso se ha repetido dos veces más, dando lugar a todos los huecos que conforman la mesa que aparece en la Figura 38.

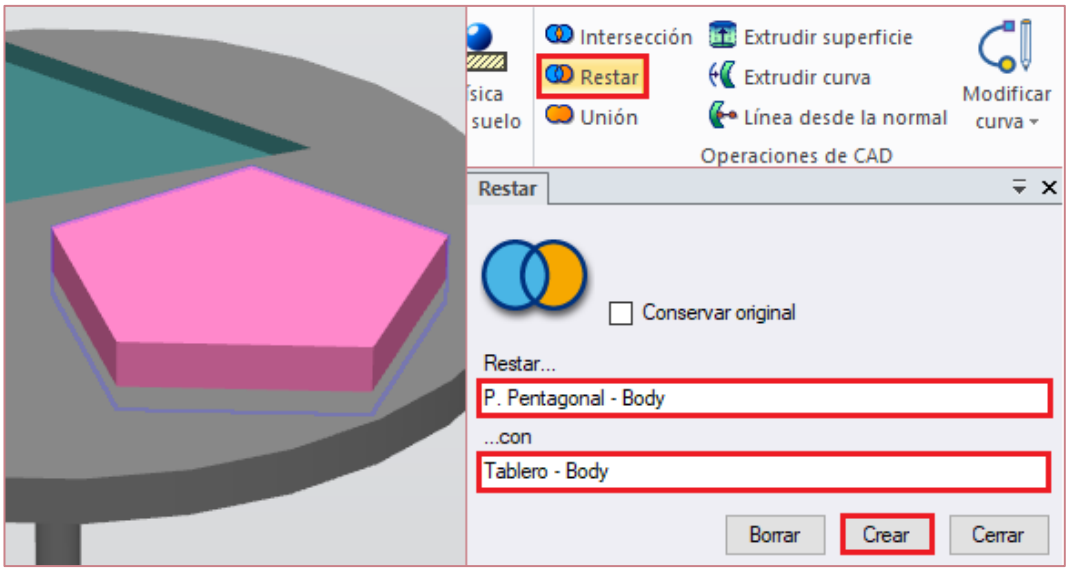

Figura 39. Creación de los huecos de la mesa.

Detector de piezas

Se trata de una herramienta que se ha diseñado y creado exclusivamente para el robot IRB 120. Esta consta de 24 sensores lineales que se activan (“1”) o se desactivan (“0”) en función de la posible detección de una pieza situada sobre la mesa giratoria durante el transcurso de la simulación.

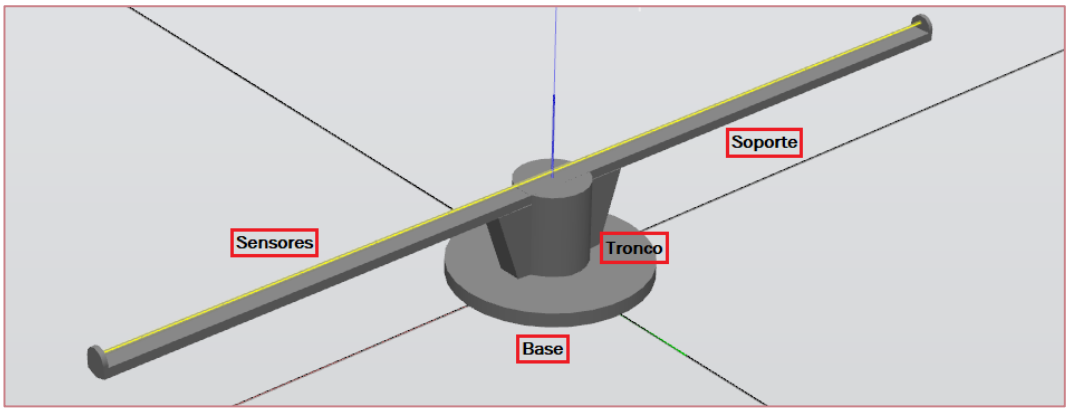

Figura 40. Detector de Piezas.

Este detector consta de tres componentes. El primero de ellos se trata de una base circular que se conecta a la muñeca del robot. La segunda pieza denominada tronco consta de un cilindro central reforzado por dos prismas trapezoidales simétricos. Finalmente, el soporte permite emplazar los veinticuatro sensores utilizados. Para modelar todos estos sólidos, se siguen los pasos indicados en la Figura 28.

A continuación, se detallan los parámetros de diseño de la herramienta.

PARÁMETROS DEL DETECTOR	
Base	Radio = 50 mm Altura = 6 mm
Tronco	Radio = 15 mm Altura = 34 mm
Soporte	Largo = 400 mm Ancho = 10 mm
Sensores	Largo = 16,24 mm Radio = 0,98 mm

Tabla 10. Parámetros de diseño del Detector.

Haciendo referencia a los sensores, es importante destacar que cada sensor está separado del siguiente, de manera equidistante, por una distancia de 0.25 milímetros.

Por otro lado, para que RobotStudio reconozca el detector como una herramienta y el robot IRB 120 pueda trabajar con ella, se deben realizar los pasos mostrados a continuación:

Lo primero es seleccionar la pestaña “Crear herramienta” como se puede observar en la Figura 41.

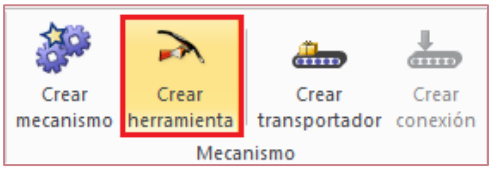

Figura 41. Pestaña Crear Herramienta.

Seguidamente, aparece un recuadro donde se debe indicar en nombre de la herramienta. A continuación, debemos seleccionar el componente que se quiere transformar en herramienta. En nuestro caso es el detector creado previamente.

Importante destacar que el proyecto se lleva a cabo en un entorno de simulación por lo que no se considera necesario introducir datos en los parámetros correspondientes a la masa de la propia herramienta así como los momentos de inercia.

Por último, se debe indicar la posión del punto central de la herramienta (TCP), ubicándose este a 40 mm sobre el eje Z.

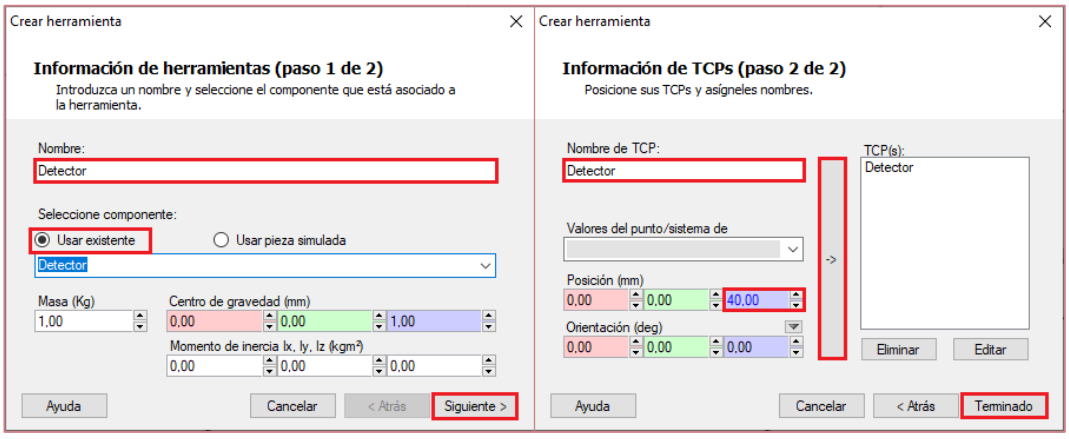

Figura 42. Pasos a seguir para la creación de una herramienta en RobotStudio.

Llegados a este punto, la herramienta ya estaría creada. Lo único que falta es colocarla en el extremo del brazo del robot IRB 120 (“Link 6”). Para ello, hacemos click derecho sobre la pestaña detector y seleccionamos la opción de “Conectar a”.

Figura 43. Pasos seguidos para la colocación de la herramienta en el extremo del robot.

El aspecto del robot ABB IRB 120 con la herramienta de detección de piezas sería el mostrado en la Figura 44.

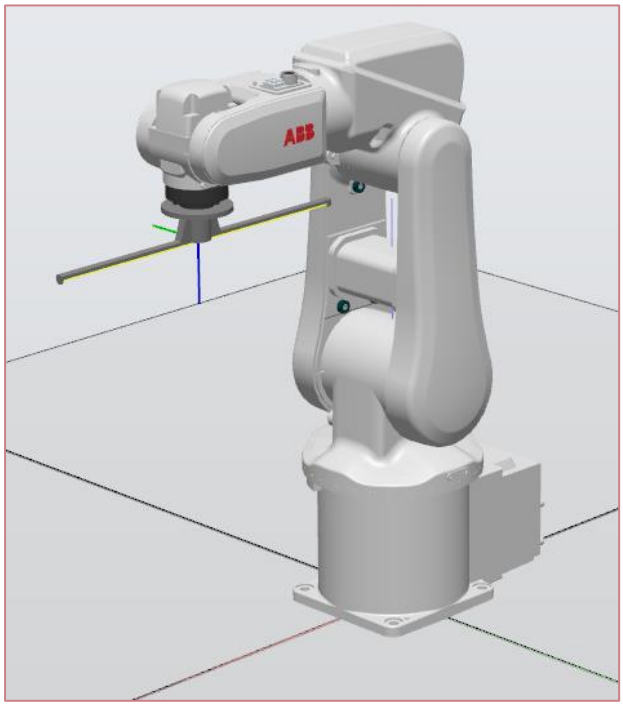

Figura 44. Robot ABB IRB 120 con la herramienta Detector.

Las conexiones lógicas que se realizan para esta herramienta son bien sencillas. Solamente consta de un componente lógico.

- LineSensor (Figura 45):

  En el apartado "Start" se indica la posición que ocupa en el espacio de trabajo el extremo inicial del sensor. El segundo apartado, "End", muestra la posición final del otro extremo. "Radius" especifica el radio del sensor a crear.

  Destacar que todos los sensores tienen que estar activados para que siempre puedan detectar cualquier objeto en su camino. "Active" estará siempre a "1".

  Cuando el sensor detecta una pieza, "SensorOut" se activa inmediatamente.

Por último, se muestran todos los sensores creados.

Figura 45. Componentes LineSensor Detector Pieza.

Ventosa

La segunda herramienta se ha creado para que el robot colaborativo CRB 15000 GoFa pueda succionar las piezas que se encuentran en la mesa giratoria y depositarlas sobre los huecos correspondientes de la otra mesa.

La ventosa consta de cuatro componentes físicos. Una base formada por un cilindro y una semiesfera, un tronco cilíndrico y un prisma cónico hueco en el extremo final de la herramienta que alberga un sensor lineal en su interior. Los pasos a seguir para el diseño y modelado de los sólidos que comprenden esta herramienta se muestran en la Figura 28.

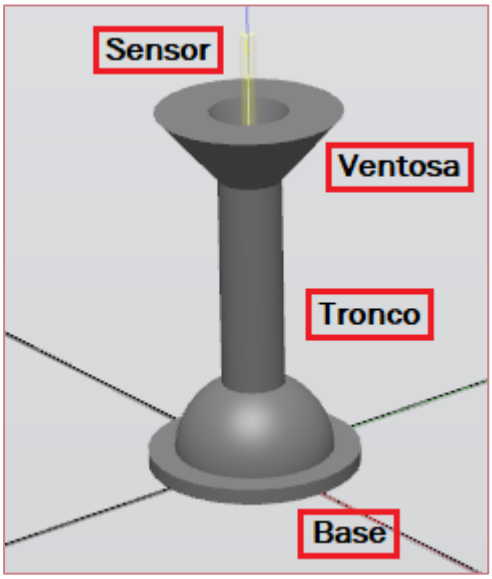

Figura 46. Ventosa.

A continuación, se mencionan los parámetros de modelado o diseño de dicha herramienta:

PARÁMETROS DE LA VENTOSA	
Base (Cilindro)	Radio = 40 mm Altura = 6 mm
Base (Semiesfera)	Radio = 30 mm
Tronco	Radio = 12.5 mm

	Altura = 80 mm
Ventosa	Radio Exterior = 35 mm Radio Interior = 15 mm Altura = 35 mm
Sensor	Largo = 30 mm Radio = 3 mm

Tabla 11. Parámetros de diseño de la Ventosa.

Del mismo modo que para el detector, debemos indicar que este componente creado se trata de una herramienta. De esta forma, el robot colaborativo podrá trabajar con ella. Para ello se siguen los mismos pasos que los indicados en las Figuras 41 y 42.

Para colocar la herramienta en el extremo del brazo del robot, véase la Figura 43.

Finalmente, se debe explicar la lógica de dicho componente y las conexiones empleadas para su funcionamiento.

- **LineSensor** (Figura 47):

  Se trata de un sensor lineal de 3 milímetros de radio que activará su señal de salida “SensorOut” cuando “SensedPart” detecte una pieza y la señal de entrada “Active” se encuentre activada.

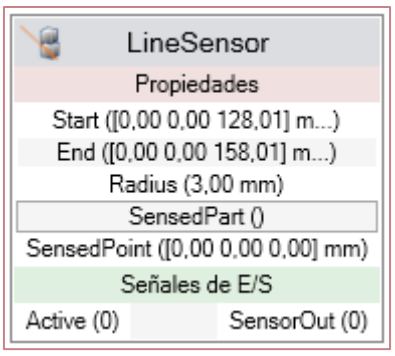

Figura 47. Componente LineSensor Ventosa.

- **Attacher** (Figura 48):

  El siguiente bloque permite conectar un objeto en RobotStudio. En nuestro caso, va a permitir conectar a la herramienta ventosa (“Parent”) la pieza detectada por el sensor lineal (“Child”). Como en todos los casos anteriores, la señal “Execute” debe estar a “1”.

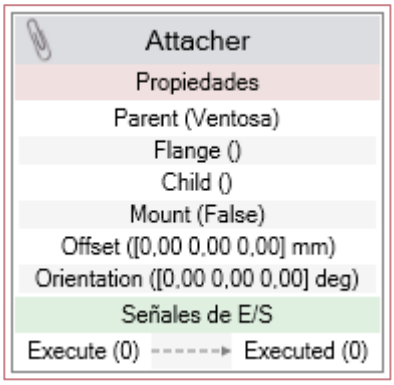

Figura 48. Componente Attacher Ventosa.

- **Dettacher** (Figura 49):

  El bloque Dettacher realiza la operación contraria al bloque explicado anteriormente. Es decir, permite desconectar un objeto que previamente ha sido conectado. De igual modo, el objeto a desconectar será la pieza detectada por el sensor (“Child”) siempre que la entrada esté activa.

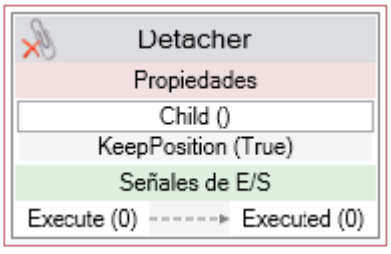

Figura 49. Componente Dettacher Ventosa.

Por último, se muestran todas las conexiones lógicas realizadas entre los distintos bloques.

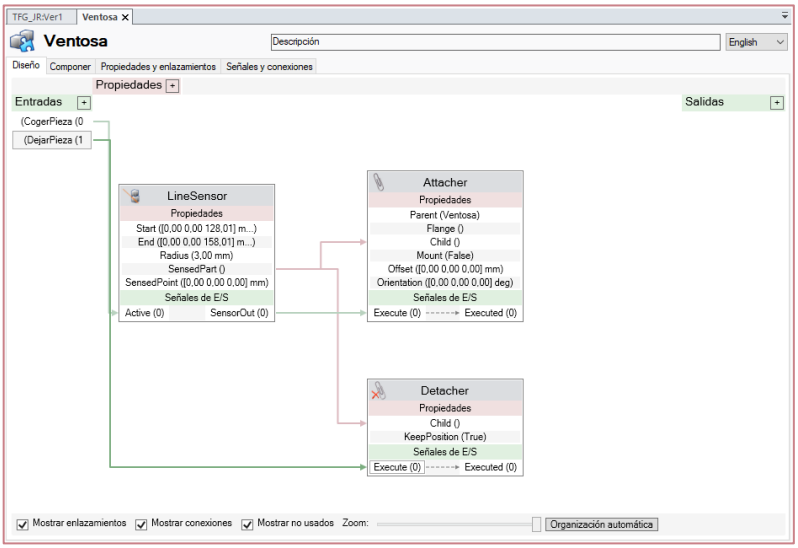

Figura 50. Conexiones Ventosa.

Véase el emplazamiento de todos los componentes creados, así como los dos robots utilizados en la estación de trabajo.

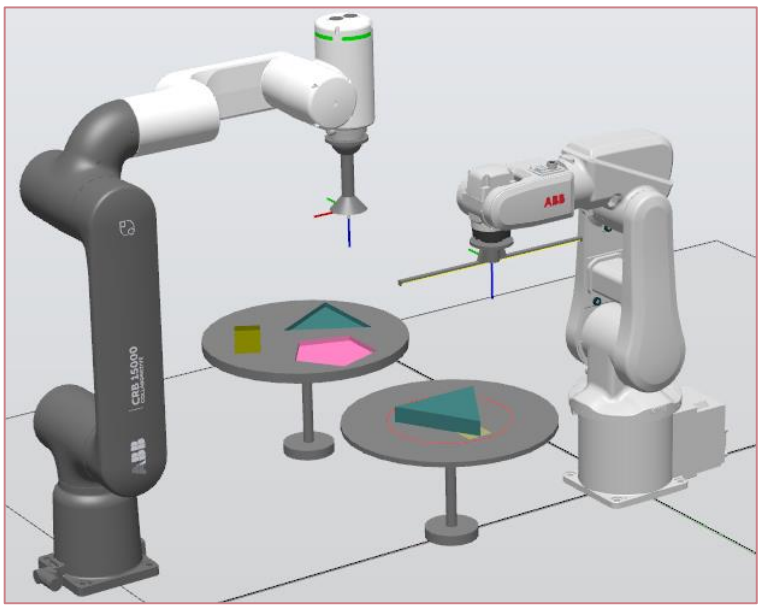

Figura 51. Estación de trabajo.

4.1.4. Trayectorias y puntos

En este apartado se explica cómo se han creado los puntos necesarios para que el robot pueda realizar las diversas trayectorias, así como los objetos de trabajos empleados o los datos de herramientas utilizados.

Tooldata

Hace referencia a los datos o características de las herramientas empleadas. En nuestro proyecto solo se trabajan con dos herramientas, el detector de piezas y la ventosa.

En primer lugar, en la pestaña “Posición inicial” se debe seleccionar la opción “Trayectorias y puntos”. Pulsando en “Datos de herramienta”, se despliega la opción de “Crear dato de herramienta” e inmediatamente aparece el recuadro que se muestra en el lateral de la Figura 52.

Seguidamente, se indica el nombre y las coordenadas de la herramienta. Para ambas herramientas diseñadas, las coordenadas introducidas coinciden con la posición del TCP como se indica en la Figura 42.

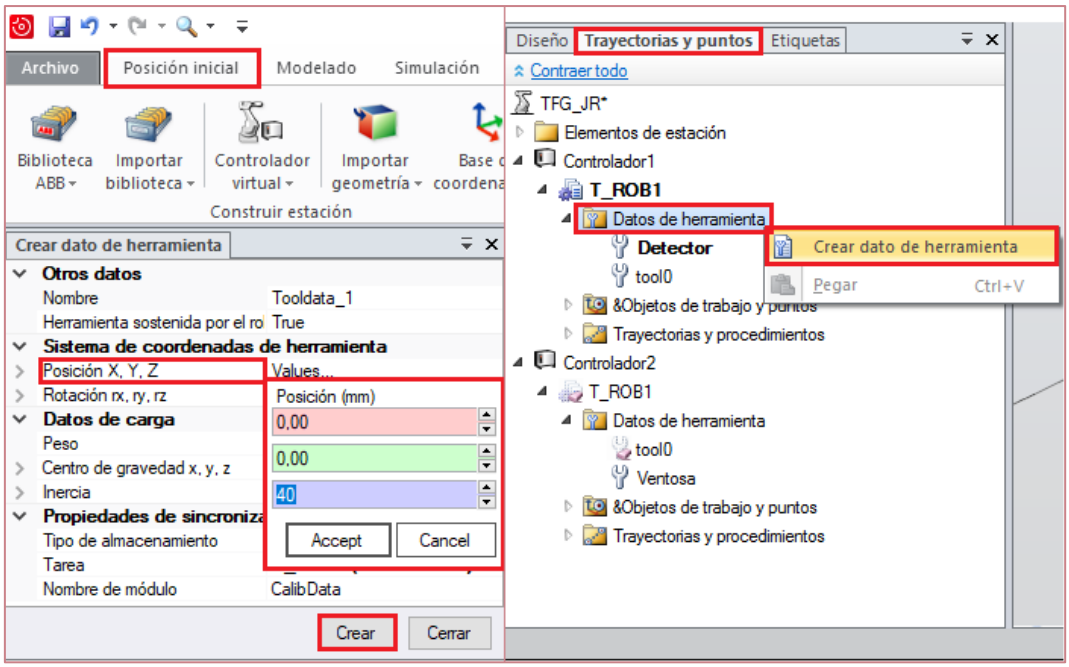

Figura 52. Creación dato herramienta (Tooldata).

Finalmente, se debe mencionar el dato de herramienta “tool0” que viene por defecto en RobotStudio y se corresponde con la muñeca del robot, no siendo necesario en ningún momento para la realización del proyecto.

Los datos de herramientas creados se pueden ver definidos a continuación:

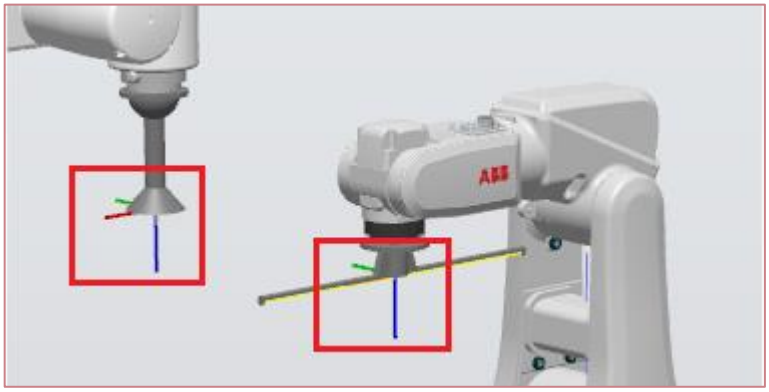

Figura 53. Visualización de Tooldatas.

Wobjdata

Este tipo de dato sirve para describir el objeto de trabajo.

Un objeto de trabajo es un sistema de referencia que se crea y se asocia a los componentes necesarios (en nuestro caso las dos mesas) de tal forma que las instrucciones llevadas a cabo por los robots se basarán en movimientos relativos a ese nuevo sistema de referencia creado.

Para ello, se debe pulsar en la pestaña “Objetos de trabajos y puntos” y seleccionar la opción “Crear objeto de trabajo”.

El siguiente paso consiste en introducir el nombre y las coordenadas de emplazamiento de dicho objeto de trabajo como se muestra en la Figura 54.

Cabe destacar “Wobj0” que hace referencia al objeto de trabajo que RobotStudio crea por defecto para cada robot.

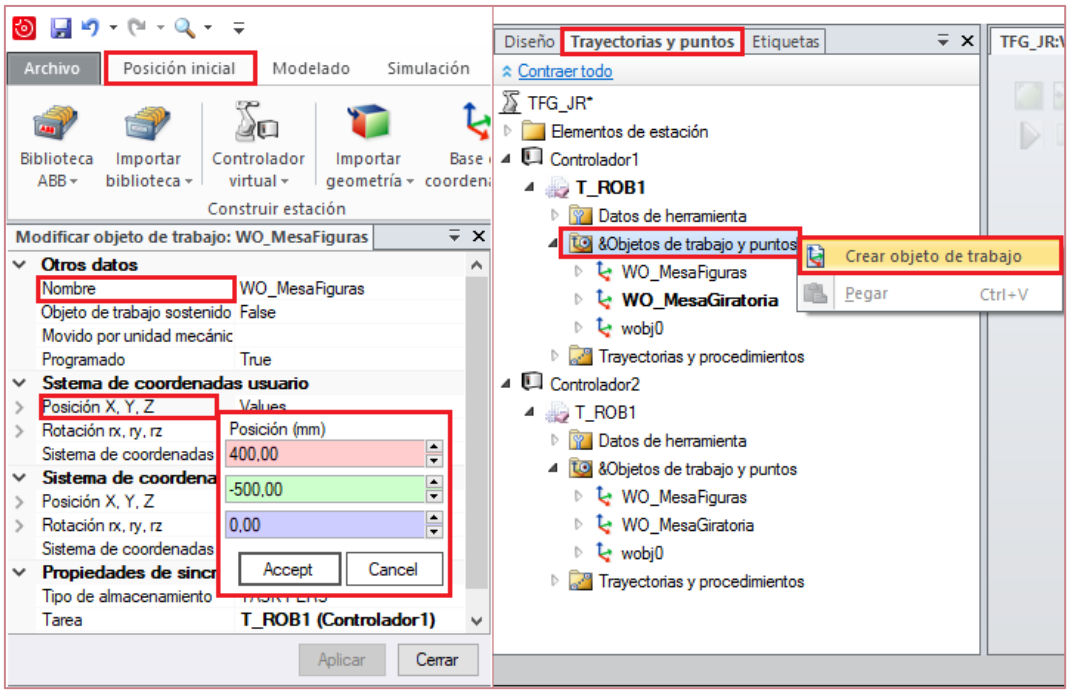

Figura 54. Creación objeto de trabajo (Wobjdata).

En la siguiente ilustración se pueden visualizar los Objetos de trabajo creados.

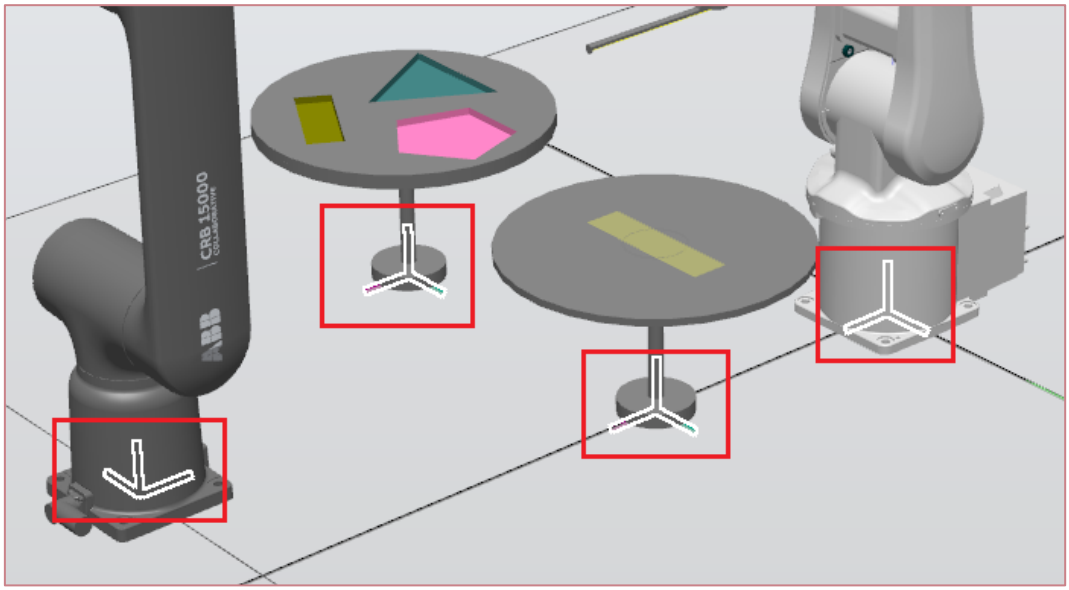

Figura 55. Visualización de Wobjdatas.

Robtarget

Este tipo de dato se emplea para definir un punto geométrico del espacio de trabajo.

Estos puntos creados son empleados en las instrucciones de movimiento de los robots para que estos puedan mover sus ejes hacia la posición especificada.

Para crear un Robtarget, se debe seleccionar el objeto de trabajo al que queremos referir las coordenadas de dicho punto y pulsar “Crear punto”.

En segundo lugar, se debe indicar la posición y orientación del punto a crear.

Finalmente, ponemos un nombre identificativo y nos aseguramos de que el objeto de trabajo seleccionado es el indicado. Véase la Figura 56.

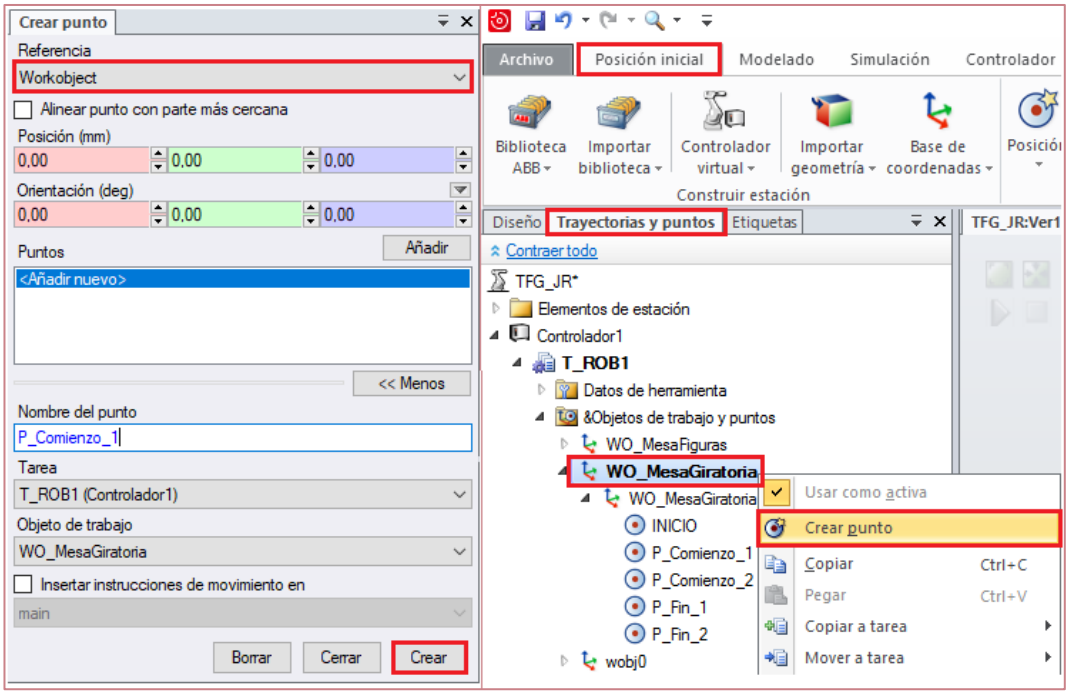

Figura 56. Creación de un punto (Robtarget).

En la Figura 57 se visualizan todos los puntos creados en sus respectivos objetos de trabajo, así como las trayectorias de movimiento realizadas por ambos robots.

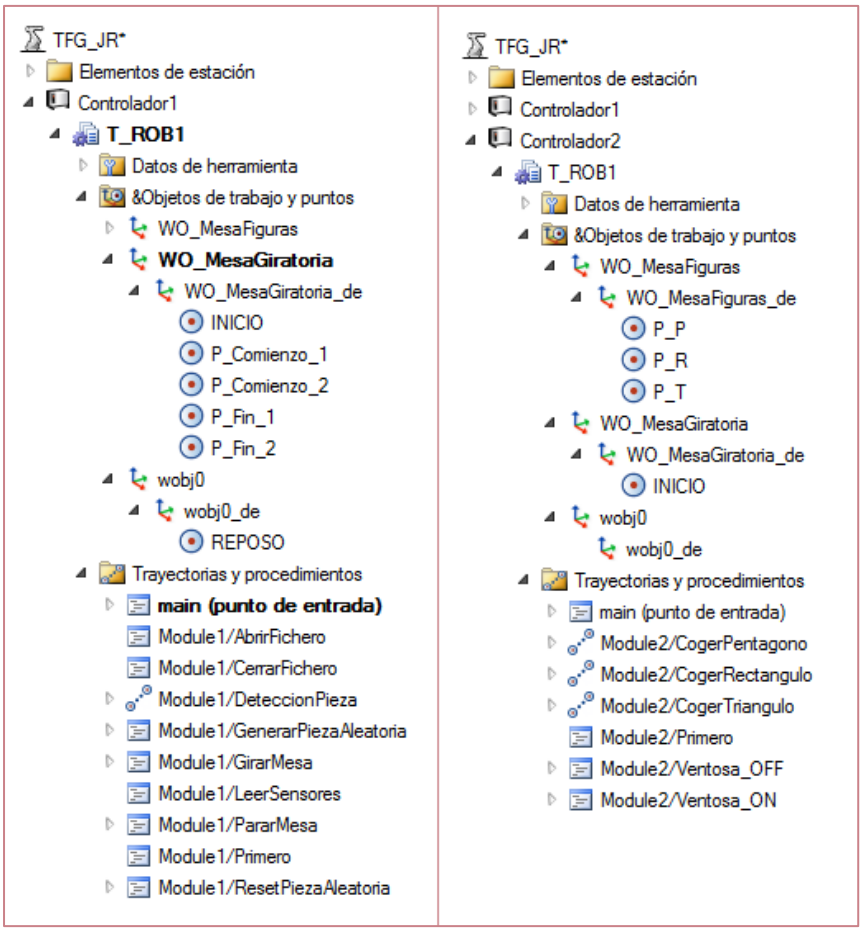

Figura 57. Visualización de puntos y trayectorias.

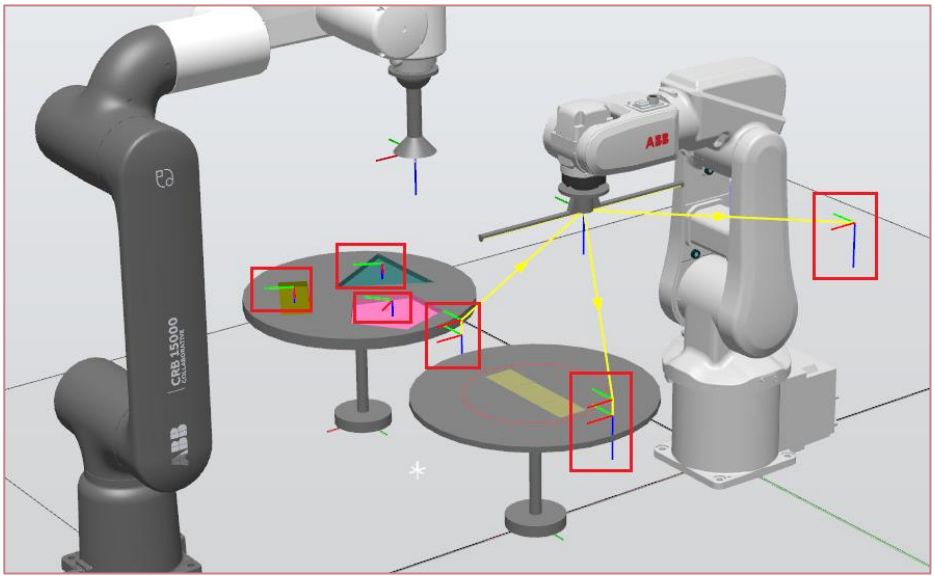

Figura 58. Puntos creados en la estación de trabajo.

Jointtarget

Haciendo uso de este tipo de dato, se logra establecer una configuración específica de todos los ejes que forman el robot.

En nuestro caso, queremos que el robot IRB 120 adopte siempre una configuración de 90º de giro en el eje número 5. Véase la Figura 64.

4.1.5. Lógica de la estación

Para poder acceder a las conexiones y lógica cableada del proyecto se debe pulsar en la pestaña “Simulación” y “Lógica de estación”.

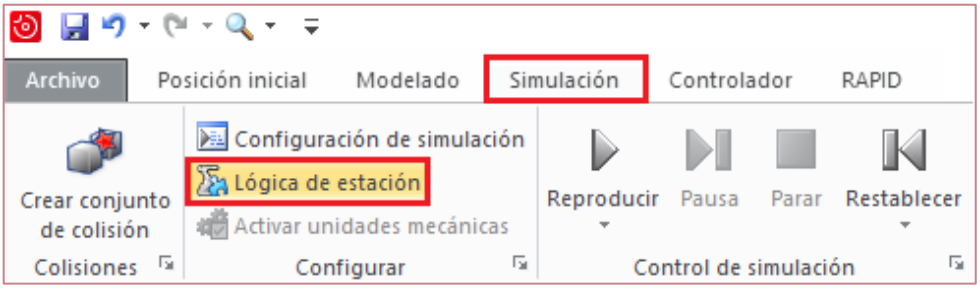

Figura 59. Pestaña Lógica de estación.

En la Figura 60 se puede observar la lógica de la estación. Consta de un bloque por cada componente inteligente creado.

El detector consta de 24 salidas digitales numeradas que se conectan a las primeras veinticuatro entradas digitales del Controlador 1 (Robot ABB IRB 120).

La señal de salida “Sensor_1” se corresponde con la entrada digital “di1” y así sucesivamente.

El Controlador 1 alberga 4 salidas digitales, siendo las dos primeras (“do12” y “do13”) empleadas para activar o desactivar respectivamente el generador de piezas aleatorias.

“do14” se utiliza para establecer el giro de la mesa. Por el contrario, “do15” permite pararla.

El controlador 2 (Robot ABB CRB 15000 GoFa) consta de dos salidas digitales conectadas al bloque de la ventosa, permitiendo a esta coger o no la pieza.

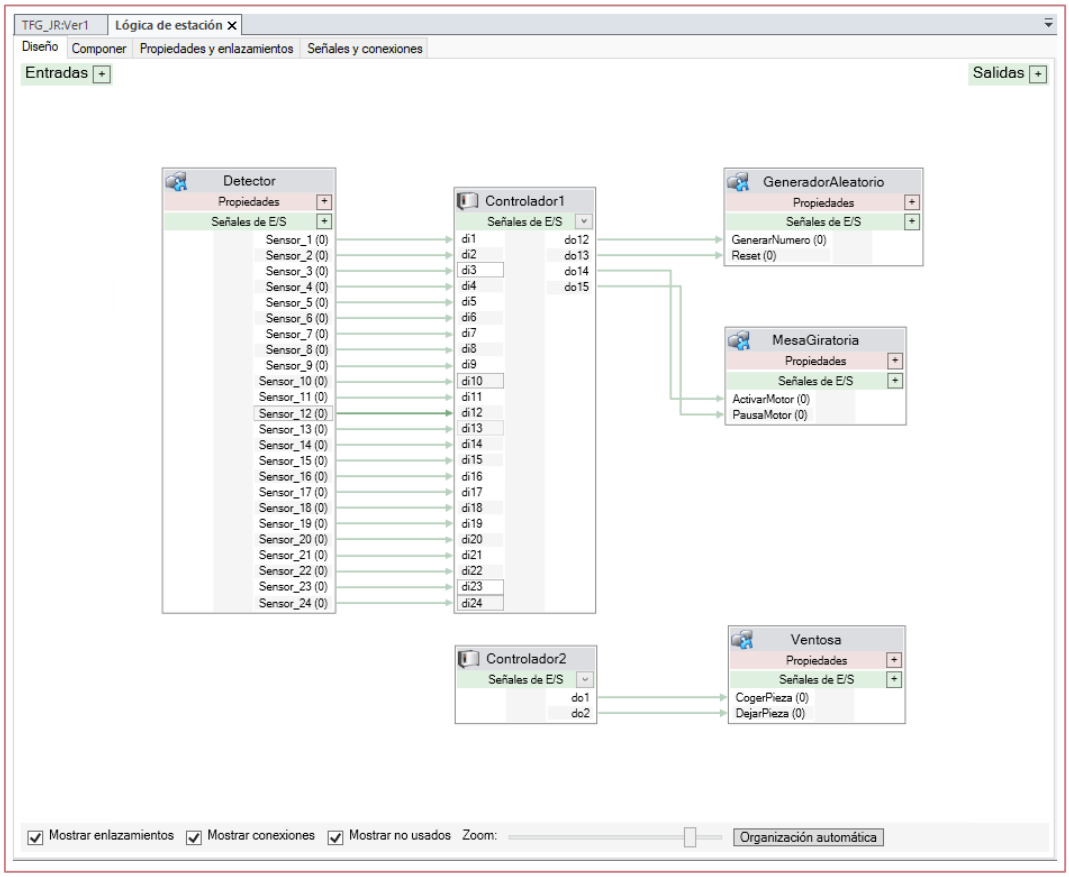

Figura 60. Lógica de la estación.

4.2. RAPID

En el presente apartado se explica y se analiza el código RAPID empleado para el funcionamiento de ambos robots. Está formado por variables, procedimientos, rutinas de interrupción, etc.

Dado que nuestro proyecto consta de dos robots diferentes con un controlador distinto para cada uno de ellos, debemos diferenciar dos códigos, denominados “Module1” y “Module2”.

4.2.1. Module 1

Este módulo ha sido programado para el Robot ABB IRB 120.

A continuación, se muestra un diagrama que alberga todos los procedimientos y funciones empleadas.

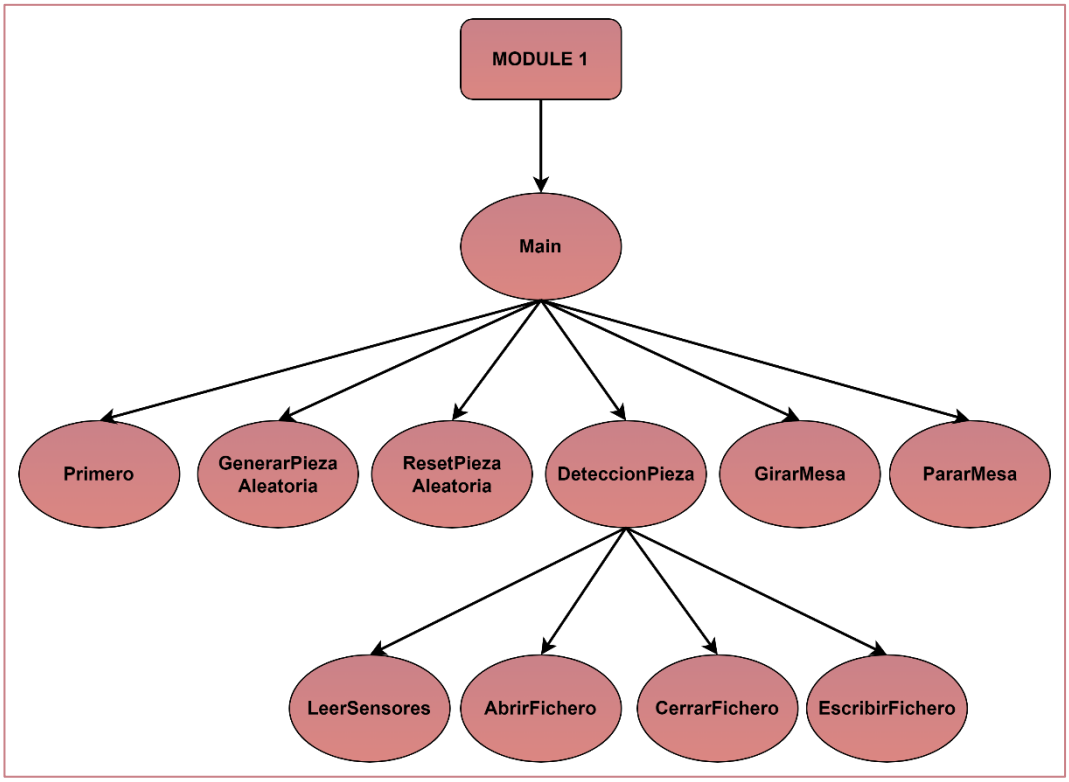

Figura 61. Diagrama de funciones y procedimientos Module1.

Variables locales

Lo primero que se debe hacer antes de empezar a programar el módulo es declarar todas las variables necesarias para el desarrollo de este.

En primer lugar, se procede a sincronizar los puntos creados en la estación (Figura 58) con RAPID. Para ello, se pulsa sobre la pestaña “Posición inicial” y se selecciona la opción “Sincronizar con RAPID”.

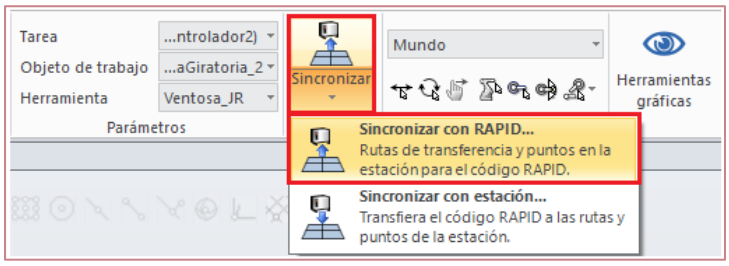

Figura 62. Sincronización con RAPID.

Seguidamente se seleccionan todos los puntos, datos de herramienta y objetos de trabajo creados que se desean visualizar en RAPID y se pulsa “Aceptar”. Véase la Figura 63.

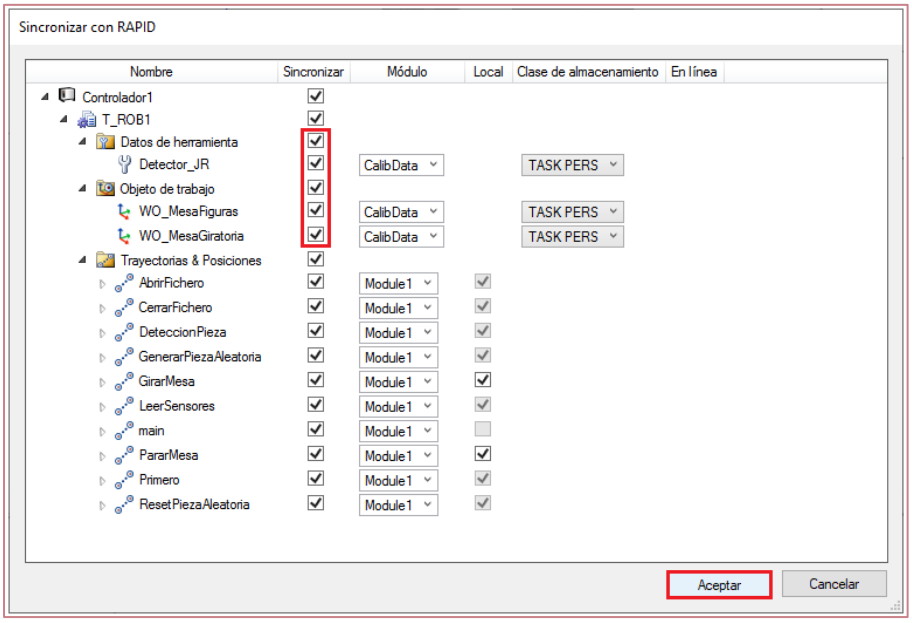

Figura 63. Sincronización de Tooldata y Wobjdata.

Aclarar que todos los puntos (robtarget) han sido creados respecto un objeto de trabajo creado previamente. Es por ello que al sincronizar con RAPID dichos objetos de trabajo, se sincronizarán del mismo modo los puntos.

```
MODULE Module1
    CONST robtarget INICIO:=[[-97.489,-0.028,518],[0,1,0,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P_Comienzo_1:=[[0,198.98989898,290],[0,1,0,0],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P_Comienzo_2:=[[0,198.98989898,263.5],[0,1,0,0],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P_Fin_1:=[[0,-198.98989898,263.5],[0,1,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget P_Fin_2:=[[0,-198.98989898,290],[0,1,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget REPOSO:=[[2.511,299.972,518],[0,1,0,0],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST jointtarget Jfase:=[[0,0,0,0,90,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 64. Puntos de la estación de trabajo (Module1).

En segundo lugar, incluimos el resto de variables locales utilizadas en este módulo.

En la Figura 65 puede apreciarse como la mayoría de variables creadas son de tipo “num”. Destacar la variable de tipo “iodev” empleada para poder leer y escribir en ficheros desde RobotStudio. “intnum” son las denominadas variables de interrupción del programa. Finalmente, mencionar la variable de tipo “clock”, tratándose esta de un reloj.

Por último y no menos importante debe hacerse mención a las variables locales persistentes (“PERS”), empleadas en el módulo para que puedan ser sincronizadas con MATLAB y de esta forma establecer la comunicación entre ambos softwares. Estas van a ser leídas en todo momento desde MATLAB y cambiarán de valor en función de la propia interacción del usuario.

```
LOCAL VAR num contador:=1;
LOCAL VAR num avance:=0;
LOCAL PERS num ejecucion_rastreo:=0;
LOCAL PERS num rastreo_ok:=0;

LOCAL VAR num Matriz{200,24};

LOCAL VAR iodev Fichero_Sensores;

LOCAL VAR intnum Idi25;
LOCAL VAR intnum Idi26;
LOCAL VAR intnum Idi27;
LOCAL VAR intnum Idi28;
LOCAL VAR intnum Idi29;

LOCAL VAR clock Clock1;
LOCAL PERS num Tiempo:=0;

LOCAL PERS num Cont_Piezas_C1:=0;
```

```
LOCAL VAR num Sensor1;
LOCAL VAR num Sensor2;
LOCAL VAR num Sensor3;
LOCAL VAR num Sensor4;
LOCAL VAR num Sensor5;
LOCAL VAR num Sensor6;
LOCAL VAR num Sensor7;
LOCAL VAR num Sensor8;
LOCAL VAR num Sensor9;
LOCAL VAR num Sensor10;
LOCAL VAR num Sensor11;
LOCAL VAR num Sensor12;
LOCAL VAR num Sensor13;
LOCAL VAR num Sensor14;
LOCAL VAR num Sensor15;
LOCAL VAR num Sensor16;
LOCAL VAR num Sensor17;
LOCAL VAR num Sensor18;
LOCAL VAR num Sensor19;
LOCAL VAR num Sensor20;
LOCAL VAR num Sensor21;
LOCAL VAR num Sensor22;
LOCAL VAR num Sensor23;
LOCAL VAR num Sensor24;
```

Figura 65. Variables locales Module1.

Una vez definidas todas las variables, se procede a explicar las funciones y procedimientos empleados en el trabajo.

Primero

Como bien indica su nombre, es un procedimiento que debe ejecutarse siempre al principio del programa.

El primer paso es cancelar las posibles interrupciones que se hubieran creado o activado antes de comenzar la simulación.

Seguidamente, asociamos las variables de interrupción definidas previamente a las rutinas “Trap”.

El último paso consiste en asignar una señal de entrada digital del controlador correspondiente con la variable de interrupción deseada.

```
LOCAL PROC Primero()

    !Cancelamos las interrupciones
    IDelete Idi25;
    IDelete Idi26;
    IDelete Idi27;
    IDelete Idi28;
    IDelete Idi29;

    !Conectamos las variables con las interrupciones
    CONNECT Idi25 WITH Trap_di25;
    CONNECT Idi26 WITH Trap_di26;
    CONNECT Idi27 WITH Trap_di27;
    CONNECT Idi28 WITH Trap_di28;
    CONNECT Idi29 WITH Trap_di29;

    !Interrupciones
    ISignalDI di25, 1, Idi25;
    ISignalDI di26, 1, Idi26;
    ISignalDI di27, 1, Idi27;
    ISignalDI di28, 1, Idi28;
    ISignalDI di29, 1, Idi29;

ENDPROC
```

Figura 66. Procedimiento Primero.

DeteccionPieza

Este procedimiento creado va a permitir realizar el proceso de rastreo de la pieza aleatoria generada que se encuentra ubicada sobre la mesa.

Para ello, se realiza un bucle “while” que va a permitir a la herramienta de detección ir recorriendo la mesa (haciendo uso de las instrucciones de tipo “MoveL”) e ir guardando en una matriz (declarada previamente como variable local) los “0” y “1” originados por los veinticuatro sensores de los que consta la herramienta, en caso de que estos detecten o no una pieza en su camino.

Por último, se escriben en un fichero todos y cada uno de esos valores almacenados en la matriz. Posteriormente MATLAB procederá a trabajar con él y haciendo uso de un algoritmo creado por nosotros mismos, seremos capaces de identificar de qué tipo de pieza se trata, así como su correspondiente orientación en la mesa.

```
LOCAL PROC DeteccionPieza()

    ConfJ\Off;
    ConfL\Off;

    MoveAbsJ Jfase,v200,fine,Detector_JR;
    MoveL P_Comienzo_1,v200,fine,Detector_JR\WObj:=WO_MesaGiratoria_1;
    MoveL P_Comienzo_2,v100,fine,Detector_JR\WObj:=WO_MesaGiratoria_1;

    CerrarFichero;
    AbrirFichero;

    WHILE avance <= 400 DO

        MoveL Offs(P_Comienzo_2,0,-avance,0),v100,fine,Detector_JR
        \WObj:=WO_MesaGiratoria_1;
        LeerSensores;

        Matriz{contador,1}:=Sensor1;
        Matriz{contador,2}:=Sensor2;
        Matriz{contador,3}:=Sensor3;
        Matriz{contador,4}:=Sensor4;
        Matriz{contador,5}:=Sensor5;
        Matriz{contador,6}:=Sensor6;
        Matriz{contador,7}:=Sensor7;
        Matriz{contador,8}:=Sensor8;
        Matriz{contador,9}:=Sensor9;
        Matriz{contador,10}:=Sensor10;
        Matriz{contador,11}:=Sensor11;
        Matriz{contador,12}:=Sensor12;
        Matriz{contador,13}:=Sensor13;
        Matriz{contador,14}:=Sensor14;
        Matriz{contador,15}:=Sensor15;
        Matriz{contador,16}:=Sensor16;
        Matriz{contador,17}:=Sensor17;
        Matriz{contador,18}:=Sensor18;
        Matriz{contador,19}:=Sensor19;
        Matriz{contador,20}:=Sensor20;
        Matriz{contador,21}:=Sensor21;
        Matriz{contador,22}:=Sensor22;
        Matriz{contador,23}:=Sensor23;
        Matriz{contador,24}:=Sensor24;


        contador:=EscribirFicheros(contador);

        WaitTime 0.1;
        contador:=contador+1;
        avance:=avance+2.02020202;

    ENDWHILE

    MoveL P_Fin_1,v100,fine,Detector_JR\WObj:=WO_MesaGiratoria_1;
    MoveL P_Fin_2,v100,fine,Detector_JR\WObj:=WO_MesaGiratoria_1;
    MoveL INICIO,v200,fine,Detector_JR\WObj:=WO_MesaGiratoria_1;
    MoveL REPOSO,v200,fine,Detector_JR\WObj:=wobj0;

    contador:=1;
    avance:=0;

ENDPROC
```

Figura 67. Procedimiento DeteccionPieza.

CerrarFichero

Antes de poder escribir en el fichero creado los valores almacenados en la matriz, se debe cerrar el fichero previamente. Se hace uso del comando "Close".

```
LOCAL PROC CerrarFichero()
    Close Fichero_Sensores;
ENDPROC
```

Figura 68. Procedimiento CerrarFichero.

AbrirFichero

En este procedimiento se indica la ruta a seguir por el software para almacenar los datos dentro del fichero. Se deben utilizar los comandos "Open" para abrir previamente el fichero y "Write" para poder escribir en él.

```
LOCAL PROC AbrirFichero()
    Open "Home:" \File:="Fichero_Sensores.txt",Fichero_Sensores\Write;
ENDPROC
```

Figura 69. Procedimiento AbrirFichero.

LeerSensores

Se asocian las veinticuatro primeras entradas digitales del controlador("di") a las variables "Sensor" creadas previamente. De esta forma, cada variable puede adoptar valor "1" en caso de que el sensor haya entrado en contacto con la pieza o valor "0" en caso contrario.

```
LOCAL PROC LeerSensores()

    Sensor1:=di1;
    Sensor2:=di2;
    Sensor3:=di3;
    Sensor4:=di4;
    Sensor5:=di5;
    Sensor6:=di6;
    Sensor7:=di7;
    Sensor8:=di8;
    Sensor9:=di9;
    Sensor10:=di10;
    Sensor11:=di11;
    Sensor12:=di12;
    Sensor13:=di13;
    Sensor14:=di14;
    Sensor15:=di15;
    Sensor16:=di16;
    Sensor17:=di17;
    Sensor18:=di18;
    Sensor19:=di19;
    Sensor20:=di20;
    Sensor21:=di21;
    Sensor22:=di22;
    Sensor23:=di23;
    Sensor24:=di24;

ENDPROC
```

Figura 70. Procedimiento LeerSensores.

EscribirFicheros

A diferencia del resto, “EscribirFicheros” es una función y no un procedimiento. La única diferencia radica en la posibilidad de poder retornar o devolver un valor (“Return”). En nuestro caso es necesario emplear esta función para poder devolver la variable “contador” y que pueda ser incrementada en cada iteración del bucle creado para el procedimiento “DeteccionPieza”. De esta forma se pueden distinguir las distintas posiciones de avance de la herramienta detector.

La primera instrucción de la Figura 71 almacena en el fichero dicha posición de avance.

El resto de instrucciones escriben en el fichero el valor adoptado por cada sensor en la posición específica de avance del detector a lo largo de la mesa.

```
LOCAL FUNC num EscribirFicheros(num contador)

    Write Fichero_Sensores,""\Num:=contador;
    Write Fichero_Sensores,""\Num:=Matriz{contador,1};
    Write Fichero_Sensores,""\Num:=Matriz{contador,2};
    Write Fichero_Sensores,""\Num:=Matriz{contador,3};
    Write Fichero_Sensores,""\Num:=Matriz{contador,4};
    Write Fichero_Sensores,""\Num:=Matriz{contador,5};
    Write Fichero_Sensores,""\Num:=Matriz{contador,6};
    Write Fichero_Sensores,""\Num:=Matriz{contador,7};
    Write Fichero_Sensores,""\Num:=Matriz{contador,8};
    Write Fichero_Sensores,""\Num:=Matriz{contador,9};
    Write Fichero_Sensores,""\Num:=Matriz{contador,10};
    Write Fichero_Sensores,""\Num:=Matriz{contador,11};
    Write Fichero_Sensores,""\Num:=Matriz{contador,12};
    Write Fichero_Sensores,""\Num:=Matriz{contador,13};
    Write Fichero_Sensores,""\Num:=Matriz{contador,14};
    Write Fichero_Sensores,""\Num:=Matriz{contador,15};
    Write Fichero_Sensores,""\Num:=Matriz{contador,16};
    Write Fichero_Sensores,""\Num:=Matriz{contador,17};
    Write Fichero_Sensores,""\Num:=Matriz{contador,18};
    Write Fichero_Sensores,""\Num:=Matriz{contador,19};
    Write Fichero_Sensores,""\Num:=Matriz{contador,20};
    Write Fichero_Sensores,""\Num:=Matriz{contador,21};
    Write Fichero_Sensores,""\Num:=Matriz{contador,22};
    Write Fichero_Sensores,""\Num:=Matriz{contador,23};
    Write Fichero_Sensores,""\Num:=Matriz{contador,24};

    RETURN contador;

ENDFUNC
```

Figura 71. Función EscribirFicheros.

GenerarPiezaAleatoria

El siguiente procedimiento está asociado a una rutina de interrupción. Véase el lateral izquierdo de la Figura 72.

El objetivo de este procedimiento es activar (“Set”) la señal digital de salida “do12” del controlador que está conectada a la entrada del componente inteligente generador, permitiendo así crear una pieza. Véase la Figura 60.

Previamente nos aseguramos de que no haya ninguna pieza creada haciendo “Reset” a la señal “do13” que se encuentra conectada del mismo modo al generador, en este caso a la entrada de reseteo de piezas.

```
LOCAL TRAP Trap_di25
    GenerarPiezaAleatoria;
ENDTRAP
```

```
LOCAL PROC GenerarPiezaAleatoria()
    Reset do13;
    Set do12;
ENDPROC
```

Figura 72. Procedimiento GenerarPiezaAleatoria.

ResetPiezaAleatoria

La función del presente procedimiento es opuesta a la del procedimiento descrito anteriormente.

Permite eliminar una pieza creada previamente. Para ello, "do13" debe estar activada.

```
LOCAL TRAP Trap_di26
    ResetPiezaAleatoria;
ENDTRAP
```

```
LOCAL PROC ResetPiezaAleatoria()
    Reset do12;
    Set do13;
ENDPROC
```

Figura 73. Procedimiento ResetPiezaAleatoria.

GirarMesa

Puede apreciarse en el lateral izquierdo de la ilustración cómo dicho procedimiento está asociado a una rutina de interrupción.

El objetivo es activar ("Set") la señal digital de salida del controlador "do14" que está conectada a la entrada del componente MesaGiratoria. De esta forma conseguimos que la mesa comience a girar. Véase la Figura 60.

Previamente, debemos asegurarnos de que la mesa no se encuentra realizando ningún movimiento de rotación. Es por ello que se hace "Reset" a la señal "do15", conectada del mismo modo al componente inteligente.

También debemos mencionar la activación del reloj, que comienza a medir el tiempo de giro de la mesa durante la simulación. Esta variable será útil para el algoritmo creado en MATLAB, permitiendo identificar el ángulo de giro de las piezas ubicadas en la mesa.

```
LOCAL TRAP Trap_di27
    GirarMesa;
ENDTRAP
```

```
LOCAL PROC GirarMesa()
    ClkStart Clock1;
    Reset do15;
    Set do14;
ENDPROC
```

Figura 74. Procedimiento GirarMesa.

ParарMesa

Realiza la función inversa a la citada en el procedimiento anterior.

Mediante la activación de “do15” se puede parar el giro de la mesa.

En este momento, el reloj se detiene.

```
LOCAL TRAP Trap_di28
    ParарMesa;
ENDTRAP
```

```
LOCAL PROC ParарMesa()
    ClkStop Clock1;
    Reset do14;
    Set do15;
ENDPROC
```

Figura 75. Procedimiento ParарMesa.

4.2.2. Module 2

Este módulo ha sido programado para el Robot ABB CRB 15000 GoFa.

A continuación, se muestra un diagrama que alberga todos los procedimientos y funciones empleadas.

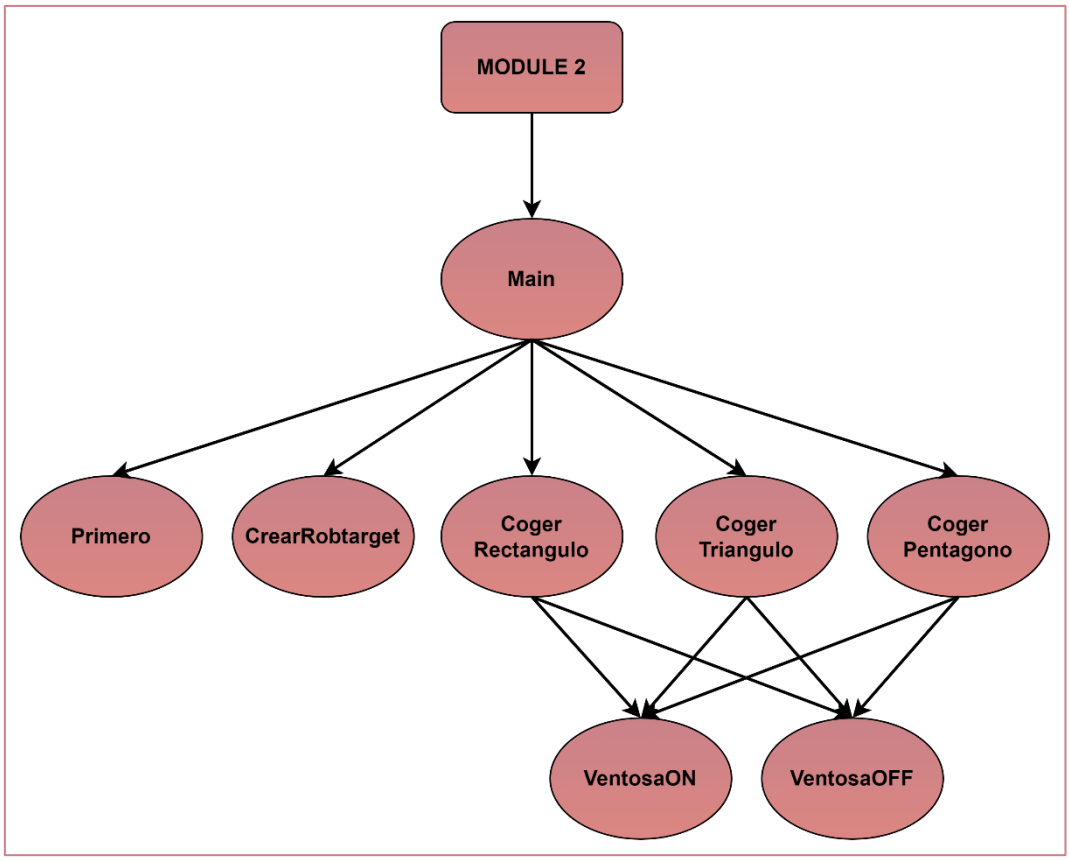

Figura 76. Diagrama de funciones y procedimientos Module 2.

Variables locales

En primer lugar, se deben sincronizar todos los puntos creados en la estación. Los pasos a seguir se muestran en la Figura 62.

```
CONST robtarget INICIO:=[[49.985,-250,578],[0,0.999999999,0,-0.000043633],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P_P:=[[0.000178541,100.000147974,260],[0,0.951056516,0.309016994,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P_R:=[[99.999999873,-49.999905211,260],[0,0.866025404,0.499999999,0],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget P_T:=[[-80.000000009,-50.001270184,260],[0,0.866025404,0.5,0],[0,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 77. Puntos de la estación de trabajo (Module2)

A continuación, se muestran el resto de variables empleadas para el Módulo 2.

```
LOCAL VAR intnum Idi1;

LOCAL PERS num Q_1:=0;
LOCAL PERS num Q_2:=0;
LOCAL PERS num Q_3:=0;
LOCAL PERS num Q_4:=0;

LOCAL PERS string Opcion_Figura:="";
LOCAL PERS num Inicio_CogerPieza:=0;
LOCAL PERS num Fin_CogerPieza:=0;

LOCAL PERS num Cont_Piezas_C2:=0;
```

Figura 78. Variables locales Module2.

En el presente módulo se puede apreciar una única variable de interrupción que permitirá al robot CRB 15000 GoFa succionar la pieza y depositarla en el hueco correspondiente de la otra mesa diseñada cuando esta se active.

```
LOCAL TRAP Trap_di1
   Inicio_CogerPieza:= 1;
ENDTRAP
```

Figura 79. Interrupción Inicio_CogerPieza.

Las variables de tipo “PERS” se encuentran inicializadas a cero. Sin embargo, van a ser modificadas desde MATLAB, permitiendo llevar a cabo el correcto funcionamiento del programa.

Primero

Como bien se ha explicado anteriormente, este módulo solo consta de una variable de interrupción.

En primer lugar, se eliminan posibles interrupciones activadas antes de la simulación. Seguidamente, se conecta la variable a la rutina de interrupción

“Trap” y finalmente se asigna dicha variable de interrupción con una entrada digital del controlador 2.

```
LOCAL PROC Primero()

    !Cancelamos las interrupciones
    IDelete Idi1;

    !Conectamos las variables con las interrupciones
    CONNECT Idi1 WITH Trap_di1;

    !Interrupciones
    ISignalDI di1, 1, Idi1;

ENDPROC
```

Figura 80. Procedimiento Primero

Ventosa_ON

En la Figura 81 se puede apreciar cómo se activa la señal digital del controlador “do2” que se encuentra conectada con la entrada de la herramienta ventosa y permite iniciar la succión de la pieza.

```
LOCAL PROC Ventosa_ON()
    SetDO do1,1;
    Reset do2;
ENDPROC
```

Figura 81. Procedimiento Ventosa_ON.

Ventosa_OFF

Este procedimiento sirve para dejar de succionar la pieza que ha sido cogida previamente por el robot una vez que se encuentra ubicada en el hueco correspondiente.

```
LOCAL PROC Ventosa_OFF()
    SetDO do2,1;
    Reset do1;
ENDPROC
```

Figura 82. Procedimiento Ventosa_OFF.

CrearRobtarget

La siguiente función es la que permite al Robot CRB 15000 GoFa saber la posición y orientación exacta con la que debe succionar la pieza que se encuentra en la mesa giratoria para su posterior emplazamiento. Para ello debe leer el robtarget que se devuelve en esta misma función.

En el primer paréntesis de la instrucción, se indica que la pieza se va a encontrar siempre en el centro de la mesa, a una altura de 260 milímetros respecto del suelo.

El segundo paréntesis hace referencia a la orientación de los ejes con la que la herramienta debe succionar la pieza. Esta orientación será proporcionada por MATLAB después de haberse ejecutado el algoritmo diseñado.

```
LOCAL FUNC robtarget CrearRobtarget()

    VAR robtarget ROBT;

    ROBT:=[[0,0,260],[Q_1,Q_2,Q_3,Q_4],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    RETURN ROBT;

ENDFUNC
```

Figura 83. Función CrearRobtarget.

CogerRectangulo

Este procedimiento se ejecuta cuando la variable persistente “Opcion_Figura” de la ilustración 78 adquiere el valor “Rectangulo”.

Seguidamente, llama a la función explicada anteriormente para saber la posición y orientación adecuada de la pieza que debe emplazar (en este caso se trata de un prisma rectangular).

Haciendo uso de instrucciones del tipo “MoveL” desplaza la pieza, previamente succionada (“Ventosa_ON”), hasta el hueco correspondiente situado en mesa y la deposita en el mismo (“Ventosa_OFF”).

Finalmente, el robot vuelve a su posición inicial.

```
LOCAL PROC CogerRectangulo()

    VAR robtarget punto;

    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    punto:=CrearRobtarget();
    MoveL Offs (punto,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    MoveL Offs (punto,0,0,0),v50,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    Ventosa_ON;
    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    MoveL Offs (P_R,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    MoveL Offs (P_R,0,0,0),v50,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    Ventosa_OFF;
    MoveL Offs (P_R,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;

ENDPROC
```

Figura 84. Procedimiento CogerRectangulo.

CogerTriangulo

El funcionamiento en este caso es igual que para el prisma rectangular con la salvedad de que se trata en este caso de un prisma triangular (“Opcion_Figura = Triangulo”).

```
LOCAL PROC CogerTriangulo()

    VAR robtarget punto;

    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    punto:=CrearRobtarget();
    MoveL Offs (punto,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    MoveL Offs (punto,0,0,0),v50,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    Ventosa_ON;
    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    MoveL Offs (P_T,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    MoveL Offs (P_T,0,0,0),v50,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    Ventosa_OFF;
    MoveL Offs (P_T,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;

ENDPROC
```

Figura 85. Procedimiento CogerTriangulo.

CogerPentagono

En este caso la variable “Opcion_Figura” adquiere el valor “Pentagono”.

Las instrucciones llevadas a cabo en la Figura 86 son iguales que para los dos casos anteriores, habiendo sido explicadas en el procedimiento “CogerRectangulo”.

```
LOCAL PROC CogerPentagono()

    VAR robtarget punto;

    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    punto:=CrearRobtarget();
    MoveL Offs (punto,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    MoveL Offs (punto,0,0,0),v50,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    Ventosa_ON;
    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;
    MoveL Offs (P_P,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    MoveL Offs (P_P,0,0,0),v50,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    Ventosa_OFF;
    MoveL Offs (P_P,0,0,15),v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_2;
    MoveL INICIO,v200,fine,Ventosa_JR\WObj:=WO_MesaGiratoria_1;

ENDPROC
```

Figura 86. Procedimiento CogerPentagono.

4.3. OPC UA

El protocolo de comunicaciones empleado en el trabajo basa su funcionamiento en una arquitectura unificada con un modelo cliente-servidor. Esto nos va a permitir intercambiar los valores de las variables programadas entre RobotStudio y MATLAB.

La comunicación se estable entre dos clientes, RobotStudio y MATLAB; y un servidor de datos, alojado en la aplicación ABB IRC5 OPC.

Haciendo uso de esta aplicación, se seleccionan los controladores que van a conectarse al servidor. Para ello se debe crear un alias por cada controlador. Se debe pulsar la pestaña “Scan” para que la aplicación sea capaz de detectar los controladores activos en ese momento. Seguidamente, se seleccionan los controladores a utilizar y se pulsa “Create”.

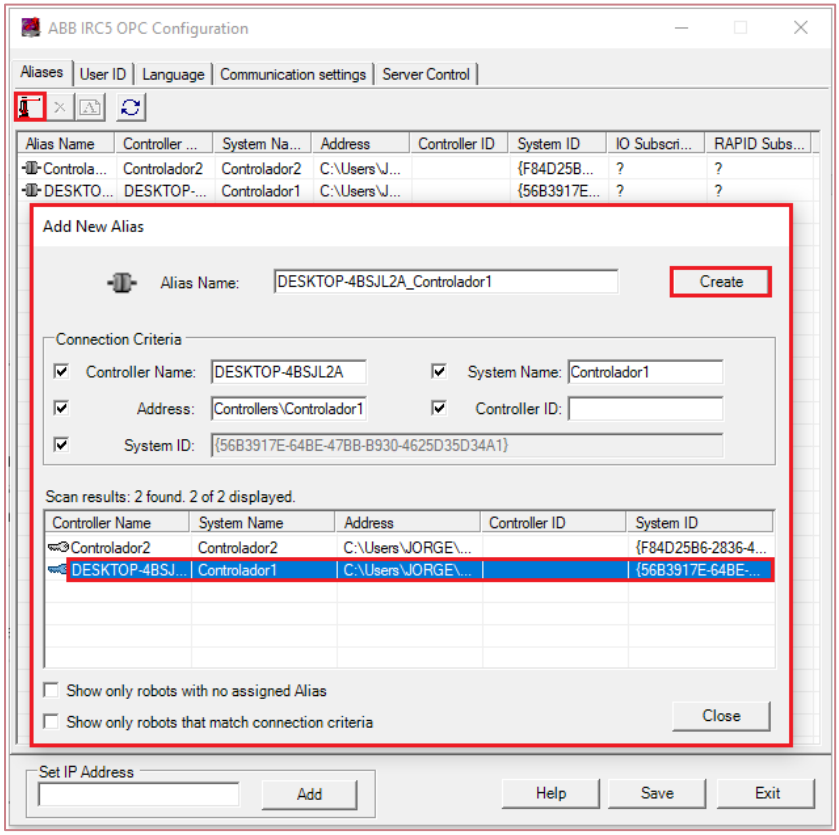

Figura 87. Añadir Controlador en ABB IRC5 OPC.

Una vez ya se han creado y reconocido los alias para cada controlador, el siguiente paso consiste en arrancar el servidor. Pulsaremos sobre la opción “Server Control” y “Start”. Véase en la siguiente ilustración.

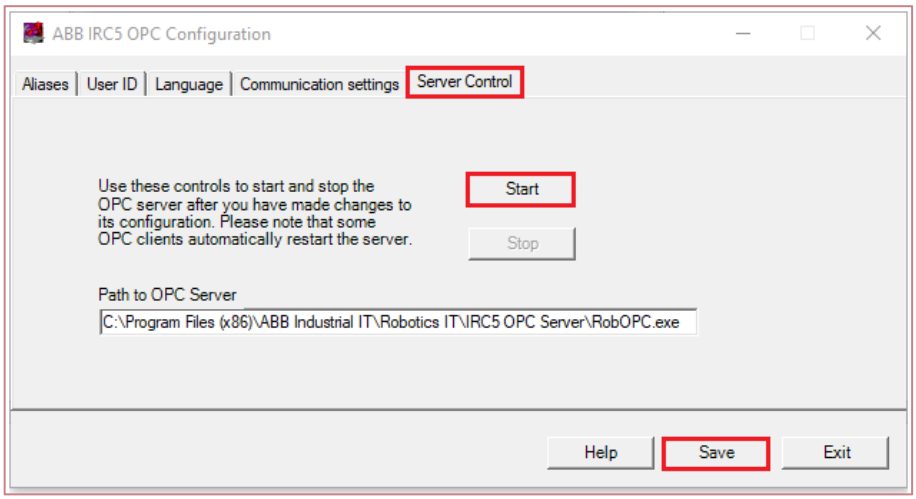

Figura 88. Arranque del servidor en ABB IRC5 OPC.

En este momento, ya se puede establecer la comunicación entre RobotStudio y el propio servidor.

El siguiente paso es dotar a MATLAB de la misma posibilidad. Se debe introducir el comando “opcDataAccessExplorer”.

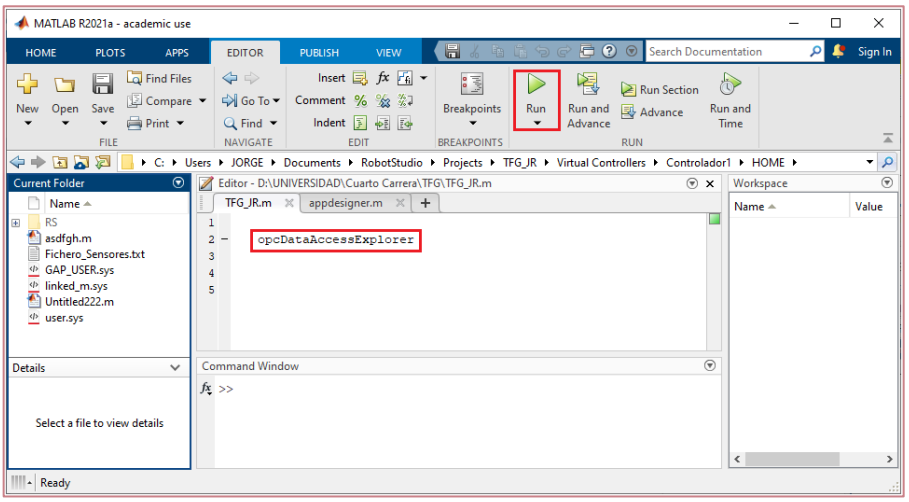

Figura 89. “opcDataAccessExplorer”.

Al ejecutar el comando mencionado anteriormente, aparece la herramienta con la que vamos a trabajar.

Lo primero que se debe hacer es crear un “Host” que corresponde a la dirección IP de la máquina que alberga el servidor OPC y se crea un “Cliente” como puede verse en el recuadro rojo de la Figura 90. Finalmente nos conectamos al servidor y ya estaría establecida la comunicación.

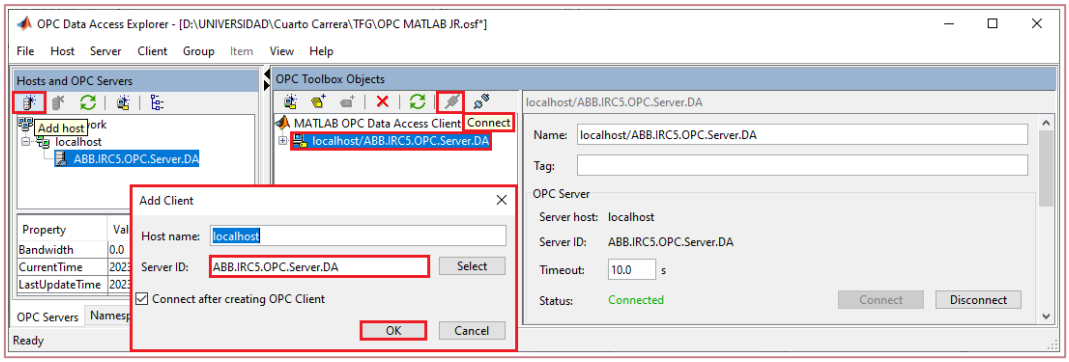

Figura 90. Cliente MATLAB.

En segundo lugar, se procede a añadir todos los elementos que van a ocupar el espacio de nombres del servidor. Esto va a permitir llevar un control o monitorización en todo momento de las variables seleccionadas, pudiendo ser estas modificas y/o actualizadas.

En nuestro caso, hemos creado previamente cinco grupos donde almacenamos de forma ordenadas las variables con las que queremos interaccionar. Véase la Figura 91.

Estas variables son las mencionadas en las Figuras 65 y 78, de tipo “PERS”.

Finalmente, se realiza la programación en MATLAB que se explica en el siguiente apartado.

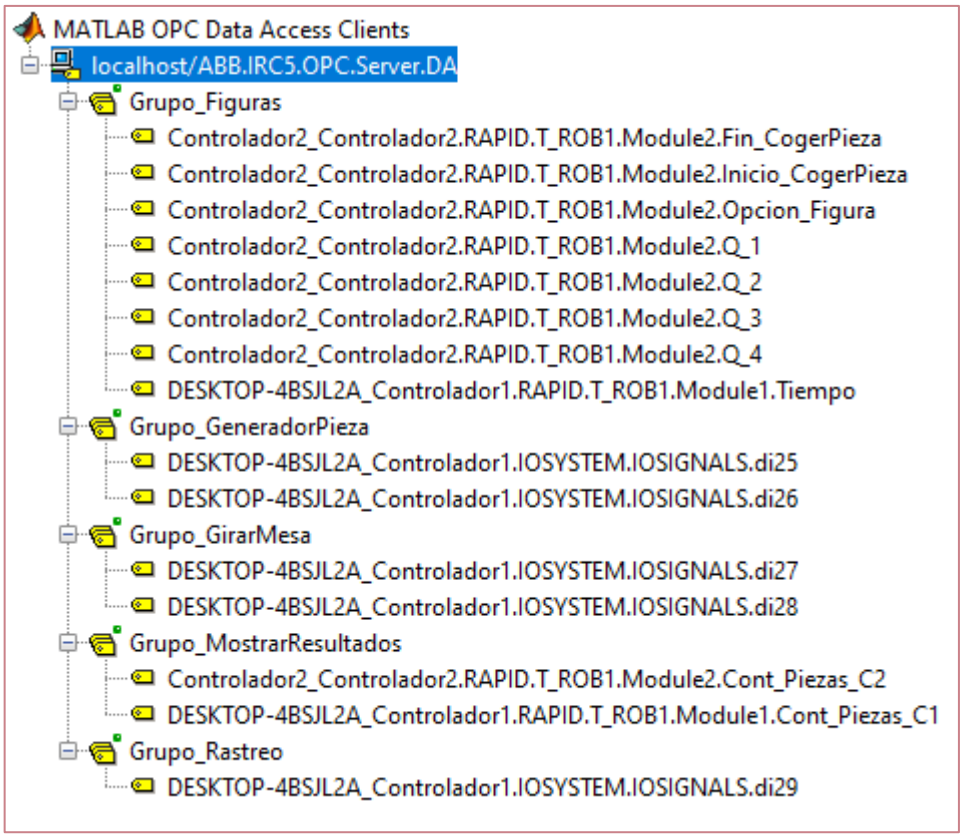

Figura 91. Variables en el Servidor OPC.

4.4. MATLAB

Con el fin de dotar a nuestro proyecto de una mayor versatilidad, así como facilitar al usuario la propia interacción, se ha creado una interfaz gráfica desde MATLAB, haciendo uso de la aplicación “AppDesigner”.

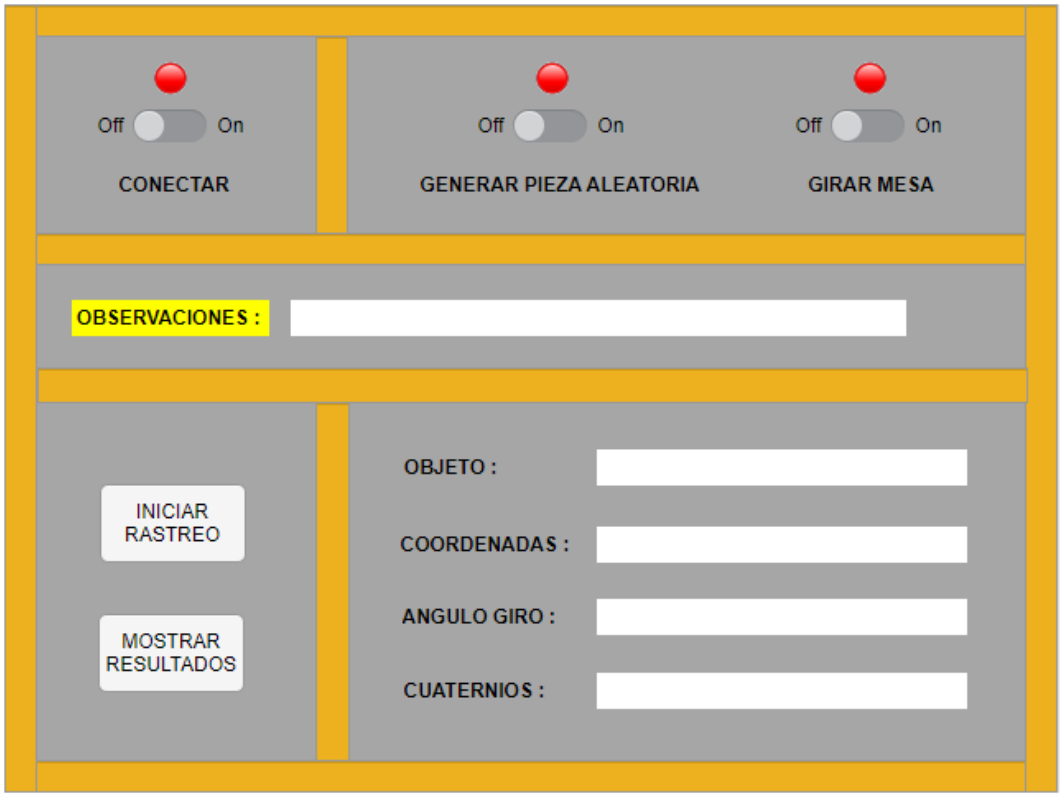

Figura 92. Interfaz gráfica de usuario.

La presente interfaz consta de tres interruptores con sus respectivas lámparas de señalización, así como de dos pulsadores y diferentes cuadros de observación, que permitirán en todo momento al usuario conocer el estado en que se encuentra la simulación, junto con los resultados finales.

A continuación, se procede a explicar cada una de las funciones creadas para todos los interruptores y pulsadores presentes en la Figura 92 y sus diversos cuadros de observaciones.

Conectar

En primer lugar, se crea un objeto de cliente de acceso a datos de OPC y se establece la conexión con el servidor mediante “connect”.

Seguidamente, se crea un grupo de objetos de acceso a datos OPC de donde posteriormente van a ser leídas las variables empleadas para esta función.

Para indicar al usuario de que se ha conectado correctamente al servidor, mostramos un mensaje en el cuadro de observaciones.

Debemos asegurarnos antes de comenzar la ejecución del programa de que todas las variables empleadas se encuentran inicializadas a sus valores correspondientes. Véase “Reset_Controlador1” y “Reset_Controlador2”.

Cabe destacar que se añaden dos variables muy importantes al grupo de datos de OPC. “Ejecucion_Rastreo” y “Rastreo_OK”. Estas nos van a indicar si el robot IRB 120 se encuentra ejecutando el proceso de detección de piezas o si ha finalizado y, por tanto, permanece a la espera para un nuevo proceso de rastreo.

Haciendo uso de un bucle While que va a estar siempre ejecutándose hasta que el usuario desee desconectarse del servidor y explorando el espacio de nombres del servidor, se consiguen leer ambas variables mediante el comando “read”.

```
function Conectar(app)

    %Creamos la comunicación
    app.DA = opcda('localhost', 'ABB.IRC5.OPC.Server.DA');
    connect(app.DA);
    app.grupo_C = addgroup(app.DA, 'Grupo_Conectar');
    set(app.grupo_C, 'UpdateRate', 0.001);

    app.Lamp_Conexion.Color = app.Color_Verde;
    app.Observaciones.FontColor = app.Color_Negro;
    app.Observaciones.Text = ' CONECTADO AL SERVIDOR CON ÉXITO.';
    app.TickRojo(0);
    app.TickLoading(0);
    app.TickVerde(1);

    app.Stop = 0;

    app.Reset_Controlador1;
    pause(1);
    app.Reset_Controlador2;


    app.Ejecucion_Rastreo = additem(app.grupo_C, [app.C1, ...
        'Controlador1.RAPID.T_ROB1.Module1.ejecucion_rastreo']);

    app.Rastreo_OK = additem(app.grupo_C, [app.C1, ...
        'Controlador1.RAPID.T_ROB1.Module1.rastreo_ok']);

    write(app.Ejecucion_Rastreo, 0);
    pause(0.1);
    write(app.Rastreo_OK, 0);
    pause(0.1);


    while app.Stop == 0

        serveritems(app.DA, [app.C1, ...
            'Controlador1.RAPID.T_ROB1.Module1.ejecucion_rastreo']);

        serveritems(app.DA, [app.C1, ...
            'Controlador1.RAPID.T_ROB1.Module1.rastreo_ok']);

        mostrar1 = read(app.Ejecucion_Rastreo);
        pause(1);
        disp(mostrar1);

        mostrar2 = read(app.Rastreo_OK);
        pause(1);
        disp(mostrar2);

    end
end
```

Figura 93. Función Conectar.

Generar Pieza Aleatoria

Este interruptor permite crear de manera aleatoria las diversas piezas en RobotStudio.

Cuando el usuario activa el interruptor, se pone a uno la variable "Switch_Generador_Piezas_On", asociada a la interrupción correspondiente de RobotStudio ("di25"), permitiendo de esta forma crear la pieza.

```
function Generar_Pieza_Aleatoria(app)

    app.Lamp_Generador.Color = app.Color_Verde;

    if app.Contador_Piezas > 0
        write(app.Switch_Generador_Piezas_Off,0)
        pause(0.1);
    end

    write(app.Switch_Generador_Piezas_On, 1);
    pause(0.1);

    app.Observaciones.FontColor = app.Color_Negro;
    app.Observaciones.Text = ' PIEZA ALEATORIA GENERADA.';
    app.TickRojo(0);
    app.TickLoading(0);
    app.TickVerde(1);
    app.Contador_Piezas = app.Contador_Piezas + 1;
    app.Valor_Pieza = 1;

end
```

Figura 94. Función Generar Pieza Aleatoria.

Por el contrario, si el usuario desactiva el interruptor, se elimina inmediatamente la pieza creada con anterioridad. Para ello, se escribe un 0 en la variable descrita anteriormente y se activa la variable asociada a la interrupción de eliminar la pieza ("di26").

Cabe mencionar que en todo momento se le indica al usuario la posible creación o eliminación de la pieza en el cuadro de observaciones creado.

```
function Reset_Pieza_Aleatoria(app)

    app.Lamp_Generador.Color = app.Color_Rojo;

    write(app.Switch_Generador_Piezas_On, 0);
    pause(0.1);
    write(app.Switch_Generador_Piezas_Off,1)
    pause(0.1);

    app.Observaciones.FontColor = app.Color_Negro;
    app.Observaciones.Text = ' PIEZA ALEATORIA ELIMINADA.';
    app.TickRojo(0);
    app.TickLoading(0);
    app.TickVerde(1);
    app.Contador_Reset_Piezas = app.Contador_Reset_Piezas + 1;
    app.Valor_Pieza = 0;

end
```

Figura 95. Función Reset Pieza Aleatoria.

Girar Mesa

El funcionamiento de este interruptor es idéntico al de crear piezas aleatorias, pero empleando las variables e interrupciones correspondientes a este caso ("di27" y "di28"). El usuario decide en todo momento si desea que la mesa gire o no. Véanse las Figuras 96 y 97.

```
function Girar_Mesa_1(app)

    app.Lamp_Mesa.Color = app.Color_Verde;

    if app.Contador_Giro_Mesa > 0
        write(app.Switch_Mesa_Giratoria_1_Off, 0)
        pause(0.1);
    end

    write(app.Switch_Mesa_Giratoria_1_On, 1);
    pause(0.1);


    app.Observaciones.FontColor = app.Color_Negro;
    app.Observaciones.Text = ' MESA 1 GIRANDO...';
    app.TickVerde(0)
    app.TickRojo(0);
    app.TickLoading(1);
    app.Contador_Giro_Mesa = app.Contador_Giro_Mesa + 1;
    app.Valor_Mesa = 1;
    app.Contador_Giro = app.Contador_Giro + 1;

end
```

Figura 96. Función Girar Mesa.

```
function Parar_Mesa_1(app)

    app.Lamp_Mesa.Color = app.Color_Rojo;

    write(app.Switch_Mesa_Giratoria_1_On, 0);
    pause(0.1);


    write(app.Switch_Mesa_Giratoria_1_Off, 1)
    pause(0.1);


    app.Observaciones.FontColor = app.Color_Negro;
    app.Observaciones.Text = ' MESA 1 PARADA.';
    app.TickRojo(0);
    app.TickLoading(0);
    app.TickVerde(1);

    app.Contador_Paro_Mesa = app.Contador_Paro_Mesa + 1;
    app.Valor_Mesa = 0;

end
```

Figura 97. Función Parar Mesa.

Iniciar Rastreo

Para que el usuario pueda iniciar el rastreo, será necesario que se haya creado previamente una pieza. Es decir, el interruptor “Crear Pieza Aleatoria” deberá estar activado. Del mismo modo, para poder iniciar el rastreo de piezas, la mesa deberá estar parada; es decir, el interruptor deberá estar desactivado.

Finalmente, cuando el usuario pulse el botón “Iniciar Rastreo”, se manda la orden a RobotStudio mediante la interrupción “di29” asociada a la variable “Button_Rastreo” y el robot ABB IRB 120 comienza dicho proceso.

Si por cualquier caso el usuario decide pulsar el botón de “Mostrar Resultados” sin que haya terminado el proceso de detección de piezas, aparecerá en el cuadro de observaciones un mensaje indicándole que debe esperar a que finalice el proceso.

```
% Button pushed function: IniciarRastreo
function IniciarRastreoButtonPushed(app, event)

    di29 = 'Controlador1.IOSYSTEM.IOSIGNALS.di29';

    app.Objeto.Text = ' ';
    app.AnguloGiro.Text = ' ';
    app.Cuaternios.Text = ' ';

    if app.Valor_Pieza == 0

        write(app.Ejecucion_Rastreo, 0);
        app.Observaciones.FontColor = app.Color_Rojo;
        app.Observaciones.Text = ' DEBE HABERSE GENERADO UNA PIEZA PREVIAMENTE.';
        app.TickVerde(0);
        app.TickLoading(0);
        app.TickRojo(1);

    elseif app.Valor_Mesa == 1

        write(app.Ejecucion_Rastreo, 0);
        app.Observaciones.FontColor = app.Color_Rojo;
        app.Observaciones.Text = ' LA MESA DEBE ESTAR PARADA PREVIAMENTE.';
        app.TickVerde(0);
        app.TickLoading(0);
        app.TickRojo(1);

    else

        if app.Contador_Rastreo == 0
            app.grupo_R = addgroup(app.DA, 'Grupo_Rastreo');
            set(app.grupo_R, 'UpdateRate', 0.001);
            app.Button_Rastreo = additem(app.grupo_R, [app.C1, di29]);
        end

        write(app.Button_Rastreo, 1);
        pause(1);

        while app.Ejecucion_Rastreo.value == 1
            app.Observaciones.FontColor = app.Color_Negro;
            app.Observaciones.Text = ' REALIZANDO RASTREO...';
            app.TickVerde(0);
            app.TickRojo(0);
            app.TickLoading(1);

            disp('Realizando Rastreo...');
            pause(0.5);
        end


        write(app.Button_Rastreo, 0);
        pause(0.1);

        app.Observaciones.FontColor = app.Color_Negro;
        app.Observaciones.Text = ' RASTREO REALIZADO CORRECTAMENTE.';
        app.TickRojo(0);
        app.TickLoading(0);
        app.TickVerde(1);

        app.Contador_Rastreo = app.Contador_Rastreo + 1;

    end
end
```

Figura 98. Función Iniciar Rastreo

Mostrar Resultados

Una vez el rastreo ha finalizado, el usuario puede pulsar “Mostrar Resultados”.

```
% Button pushed function: MostrarResultados
function MostrarResultadosButtonPushed(app, event)

    %di15 = 'Controlador1.IOSYSTEM.IOSIGNALS.di15';


    if app.Rastreo_OK.value == 0 && app.Ejecucion_Rastreo.value == 1

        app.Observaciones.FontColor = app.Color_Rojo;
        app.Observaciones.Text = ' DEBE ESPERAR A QUE FINALICE EL PROCESO DE RASTREO.';
        app.TickVerde(0);
        app.TickLoading(0);
        app.TickRojo(1);
        pause(2);

    elseif app.Rastreo_OK.value == 0 && app.Ejecucion_Rastreo.value == 0

        Leer_Ficheros(app);

        app.Observaciones.FontColor = app.Color_Negro;
        app.Observaciones.Text = ' FICHEROS LEÍDOS CORRECTAMENTE.';
        app.TickRojo(0);
        app.TickLoading(0);
        app.TickVerde(1);

        Calcular_Area(app);

        app.Contador_MR = app.Contador_MR + 1;
        app.Contador_Giro = 0;

    end

end
```

Figura 99. Función Mostrar Resultados.

Seguidamente, esta función llama a otras dos funciones.

Una de ellas es “Leer_Ficheros”, que se encarga de recorrer todas las filas del fichero que contiene los datos de los veinticuatro sensores durante el avance del proceso de detección de piezas creado en RobotStudio. Cabe destacar que el algoritmo empleado para recorrer las filas del fichero, así como almacenar todos los datos en una matriz, que va a ser utilizada en la segunda función, han sido creados por nosotros mismos. Puede verse dicho algoritmo en la siguiente ilustración.

```
function Leer_Ficheros(app)

    cd
    ruta = cd(['C:\Users\JORGE\Documents\RobotStudio\' ...
        'Projects\TFG_JR\Virtual Controllers\Controlador1\HOME']);
    ficheros = ls(ruta)

    fichJR = fopen('Fichero_Sensores.txt');


    NF = 4951;      %Filas del FICHERO
    contador = 1;   %Contador para recorrer todas las filas del fichero
    fila = 1;       %Los múltiplos de 25 contienen la fila del rastreo donde se encuentra


    filas = 198;    %Filas AVANCE DETECTOR
    columnas = 24;  %Numero de sensores
    Matriz = zeros(filas,columnas);
    Mascara = ones(filas, columnas);


    while contador<NF  %Recorrer el fichero fila por fila

        aux = fgetl(fichJR);
        datos = str2num(aux)

        if contador/fila == 1

            j=25;        %j hace referencia al contador de las columnas
            i = datos   %i hace referencia a la fila donde se encuentra
            fila = fila + 25;   %Necesitamos 1+24 digitos para realizar 1 avance

        else

            j = j - 1
            Matriz(i,j) = datos;  %guardamos los datos en la matriz

        end

        contador = contador + 1;

    end

    Matriz

    %Recorro la matriz elemento a elemento para detectar los 1
    for x=1:filas

        v = ['Fila: ', num2str(x)]

        for y=1:columnas

            if Mascara(x,y) .* Matriz(x,y) == 1
                disp('Yes');
                app.f3 = [app.f3,x];
                app.f4 = [app.f4,y];
                %y = [num2str(i), '/' ,num2str(j)]

            else
                disp('No')
                app.f1 = [app.f1,x];
                app.f2 = [app.f2,y];
            end

        end

        disp('---------------------------------')
    end

    length(app.f3)

end
```

Figura 100. Función Leer Ficheros.

La segunda función empleada es “Calcular_Area”. Dicha función a través de la matriz citada anteriormente permite calcular el área de la superficie de las piezas generadas. El algoritmo se basa en el recuento total de “1” que hay almacenados en la matriz. Finalmente se realiza una representación gráfica.

```
%Distinguimos los 3 posibles casos ("AREA FIGURA")
if length(app.f3)>240 && length(app.f3)<320
    Opcion = 'Rectangulo';
elseif length(app.f3)>440 && length(app.f3)<510
    Opcion = 'Triangulo';
elseif length(app.f3)>580 && length(app.f3)<660
    Opcion = 'Pentagono';
end
plot(app.f2,app.f1,'o',12.5,100,'*');
```

Figura 101. Función Calcular Área.

Dentro de esta función también se implementa el algoritmo que permite calcular la orientación de la pieza en función de los grados de giro de la mesa. Para ello, se tiene en cuenta el tiempo que permanece la mesa realizando el movimiento de rotación. Destacar que en todo momento se conoce la velocidad de giro de la mesa, siendo esta de 20 mm/s.

Dicho algoritmo también ha sido diseñado por nosotros mismos.

```
%Calculamos el Ángulo de Giro

if app.Contador_Giro == 0

    if strcmp(Opcion,'Triangulo') == 1
        ANG = -30;
    elseif strcmp(Opcion,'Rectangulo') == 1
        ANG = 0;
    elseif strcmp(Opcion,'Pentagono') == 1
        ANG = -36;
    end

elseif app.Contador_Giro > 0


    if strcmp(Opcion,'Triangulo') == 1

        ANG = -30 - (app.Tiempo.Value*90)/3.1455 - ((0.022*90)/3.1455)/4

    elseif strcmp(Opcion,'Rectangulo') == 1

        ANG =  - (app.Tiempo.Value*90)/3.1455 - ((0.022*90)/3.1455)/4

    elseif strcmp(Opcion,'Pentagono') == 1

        ANG = -36 - (app.Tiempo.Value*90)/3.1455 - ((0.022*90)/3.1455)/4

    end

end
```

Figura 102. Función Cálculo Ángulo Giro.

Finalmente, debemos transformar los grados de giro en radianes para poder calcular los Quaternios. Es la forma en que trabajan los robots la orientación de sus ejes. Véase la Figura 103.

```
%Calculamos los Cuaternios (Q1,Q2,Q3,Q4)

ANG_Rad = (ANG*pi)/180;

eul = [-ANG_Rad 0 pi];
Q = eul2quat(eul);   %ZYX
```

Figura 103. Cálculo de Quaternios.

El último paso que se debe realizar es escribir todos los resultados en los cuadros de observación de la interfaz con el objetivo de mostrar la información al usuario. Para ello se ha creado un menú, con tres opciones, una por cada tipo de pieza creada. Véase a continuación.

```
switch Opcion

    case 'Triangulo'

        if app.Figura ~= ' '

            %Mostramos al usuario la informacion por pantalla
            app.Objeto.FontColor = app.Color_Negro;
            app.Objeto.Text = ' PRISMA TRIANGULAR.';
            app.Coordenadas.FontColor = app.Color_Negro;
            app.Coordenadas.Text = [' X : 0 mm', '   Y : 0 mm', '   Z : 260 mm'];
            app.AnguloGiro.FontColor = app.Color_Negro;
            app.AnguloGiro.Text = [' ',num2str(round(-ANG-30,2,"decimals")), ' º'];
            app.Cuaternios.FontColor = app.Color_Negro;
            app.Cuaternios.Text = [' Q1 : ',num2str(round(Q(1),2,"decimals")),
                '   Q2 : ',num2str(round(Q(2),2,"decimals")),
                '   Q3 : ',num2str(round(Q(3),2,"decimals")),
                '   Q4 : ',num2str(round(Q(4),2,"decimals"))];

            %Pasamos la informacion al Robot 2
            write(app.Figura, 'Triangulo');
            pause(0.1);

            write(app.Q_1, Q(1));
            write(app.Q_2, Q(2));
            write(app.Q_3, Q(3));
            write(app.Q_4, Q(4));
            pause(1);

            %Colocamos finalmente la pieza en el lugar correspondiente
            write(app.Inicio_CogerPieza, 1);
            pause(0.1);

        end
```

Figura 104. Caso Prisma Triangular.

Mencionar que se escribe toda la información necesaria en las variables que van ser modificadas en RobotStudio, logrando así que el robot CRB 15000 GoFa succione con una orientación específica la pieza creada.

Desconectar

Cuando el usuario lo desee, puede desconectarse del servidor.

Se dejan de leer las variables empleadas, se hace un Reset de todas esas variables y contadores utilizados y finalmente se desconecta el servidor.

```
function Desconectar(app)

    %Hacemos reset de los contadores e interrupciones de ambos controladores
    app.Lamp_Conexion.Color = app.Color_Rojo;
    app.Reset_Controlador1;
    pause(1);
    app.Reset_Controlador2;
    pause(1);

    %Dejamos de leer las variables del servidor.
    app.Stop = 1;
    pause(1);

    %Nos desconectamos del servidor y eliminamos el objeto creado para la conexion.
    disconnect(app.DA);
    app.Observaciones.FontColor = app.Color_Negro;
    app.Observaciones.Text = ' DESCONECTADO DEL SERVIDOR CON ÉXITO.';
    app.TickRojo(0);
    app.TickLoading(0);
    app.TickVerde(1);
    delete(app.DA)
end
```

Figura 105. Función Desconectar.

CAPÍTULO 5

Funcionamiento y resultados

Tras la explicación y desarrollo del proyecto, se muestran a continuación de forma resumida los pasos a seguir para la correcta ejecución de la simulación.

- **Paso 1:** Conectarse al servidor ("Switch Conectar"). Este apartado es de obligado cumplimiento. MATLAB y RobotStudio deben estar sincronizados en todo momento.

- **Paso 2:** Generar una pieza ("Switch Generar Pieza Aleatoria"). También es obligatorio generar una pieza. En caso contrario, el proyecto carece de sentido.

- **Paso 3:** Girar la mesa ("Switch Girar Mesa"). El usuario puede girar o no la mesa, viéndose de esta manera afectada la orientación de la pieza creada previamente.

- **Paso 4:** Iniciar el proceso de detección de pieza ("Button Iniciar Rastreo"). Permite al robot IRB 120 realizar un rastreo a lo largo de la mesa giratoria y, aplicando los algoritmos diseñados en MATLAB, consigue adivinar de qué pieza se trata y su respectiva orientación.

- **Paso 5:** Finalmente se le muestra al usuario los resultados obtenidos ("Button Mostrar Resultados") y se le ordena al robot CRB 15000 GoFa que succione la pieza correspondiente y la deposite en el hueco indicado de la otra mesa.

- **Paso 6:** Finalmente, el usuario puede desconectarse del servidor.

- **Paso 7:** (Opcional) Sin desconectarse del servidor, se puede volver al paso 2 tantas veces como el usuario lo desee.

En las próximas ilustraciones se muestra el funcionamiento de los robots paso a paso durante la ejecución del programa, así como la interfaz gráfica creada.

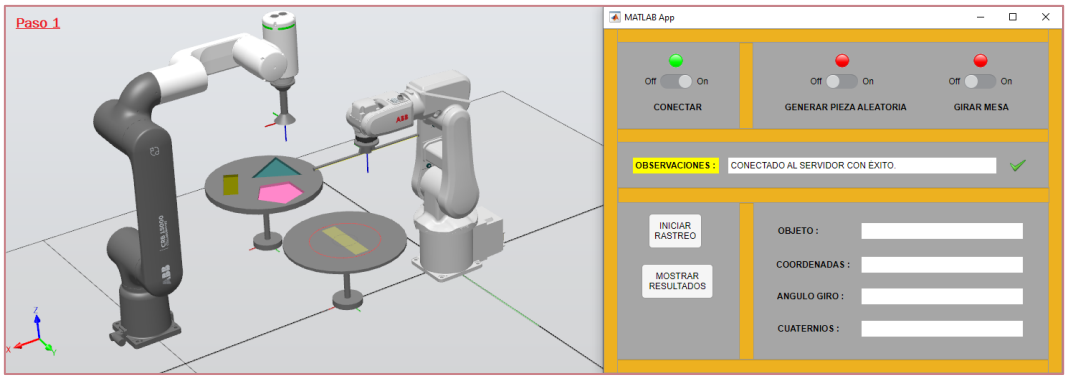

Figura 106. Paso1: Conexión con el servidor.

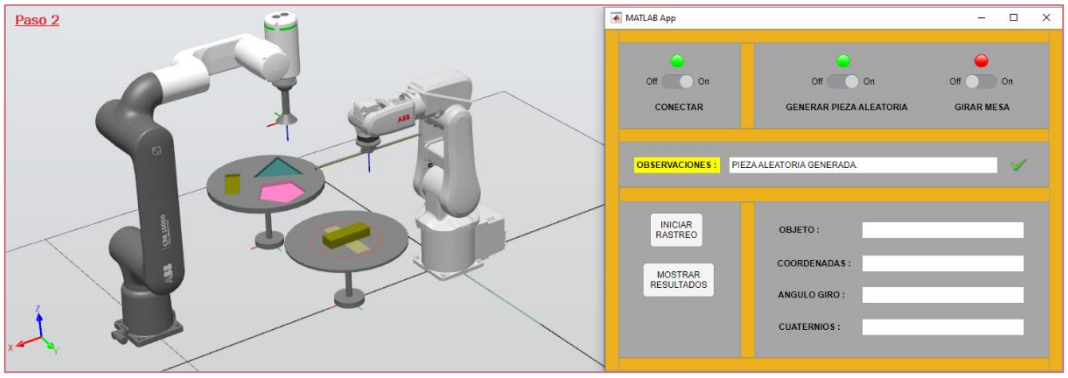

Figura 107. Paso 2: Pieza aleatoria generada.

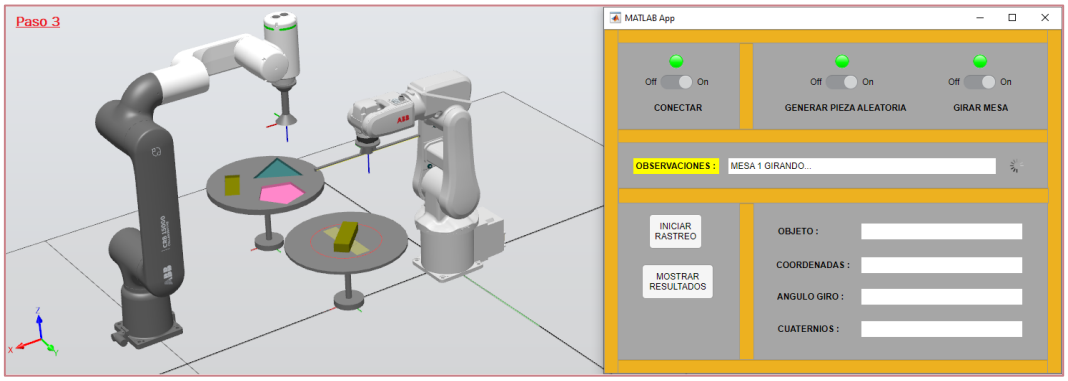

Figura 108. Paso 3: Mesa girando.

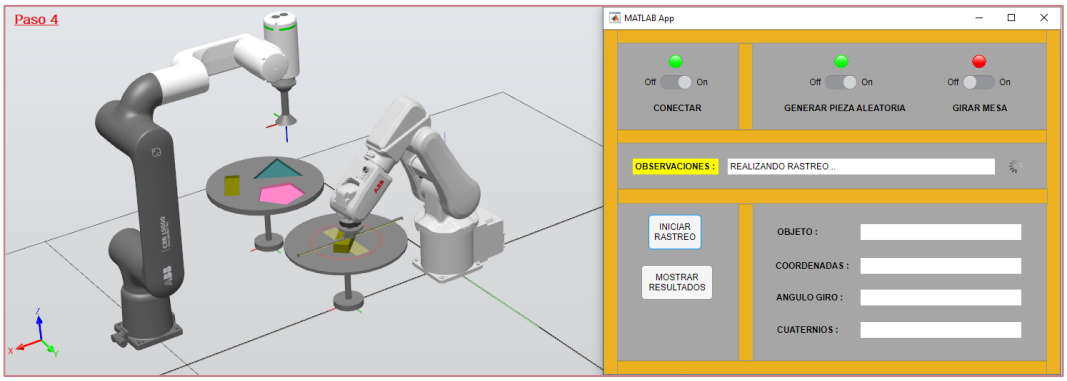

Figura 109. Paso 4: Robot IRB 120 realizando proceso de rastreo.

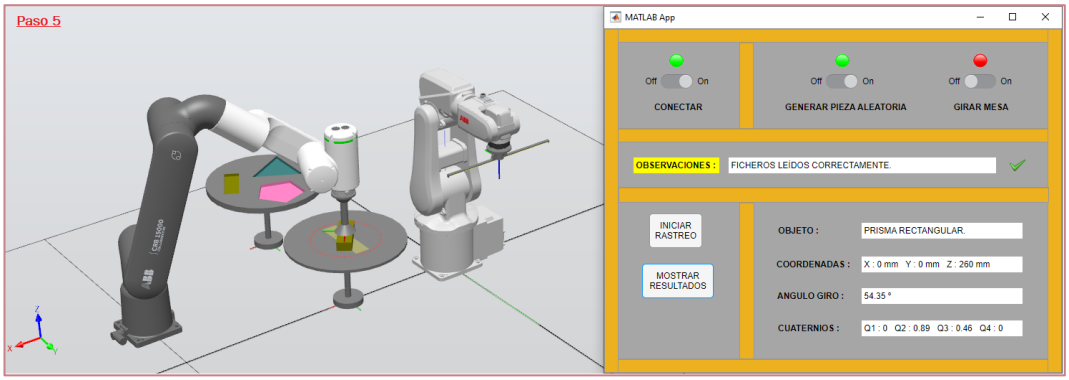

Figura 110. Paso 5: Robot CRB 15000 GoFa succionando la pieza.

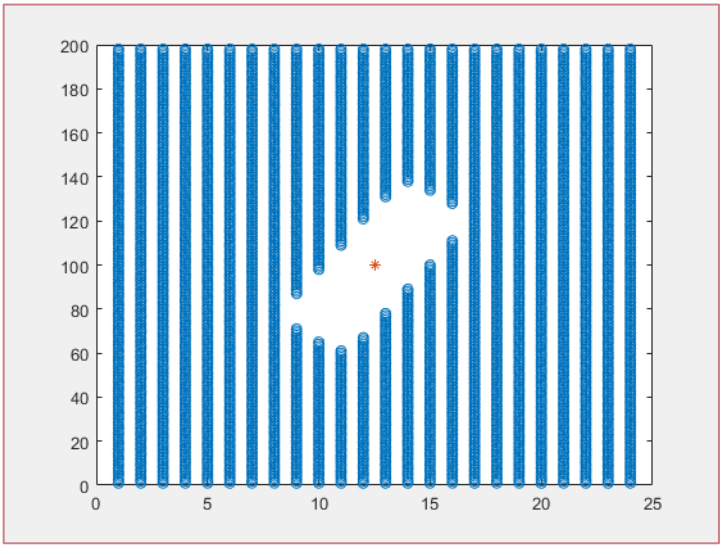

Figura 111. Representación gráfica de la pieza y su orientación en MATLAB.

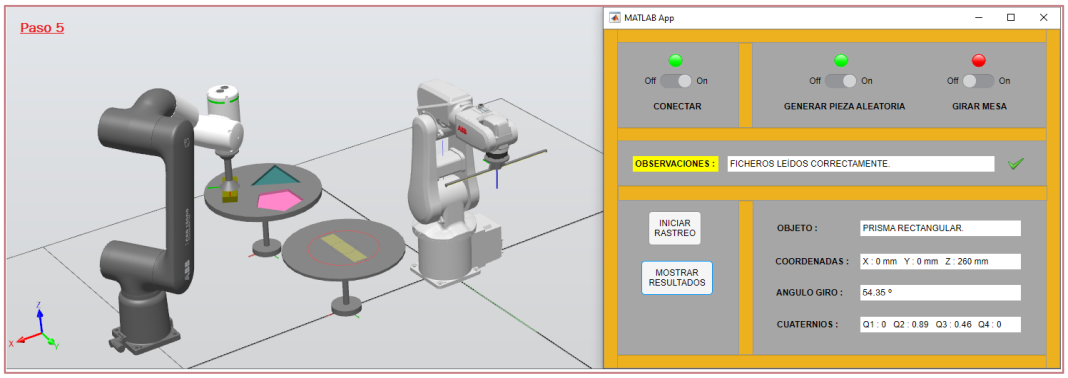

Figura 112. Paso 5: Robot CRB 15000 GoFa colocando la pieza.

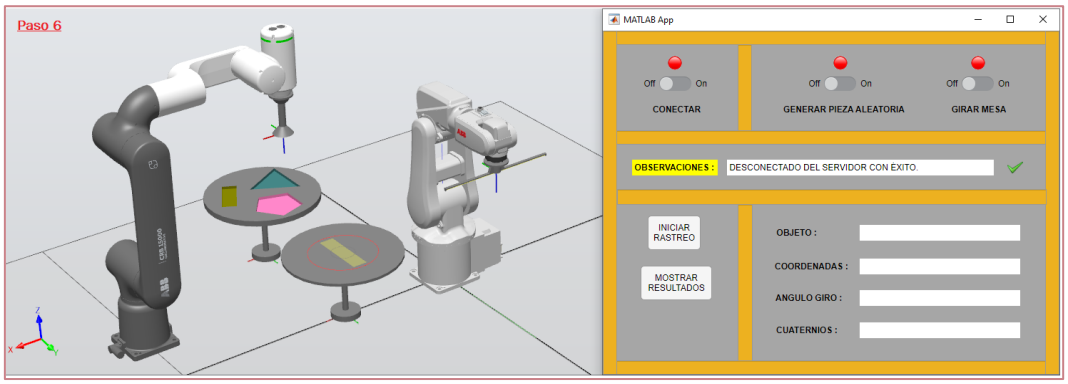

Figura 113. Paso 6: Desconexión del servidor.

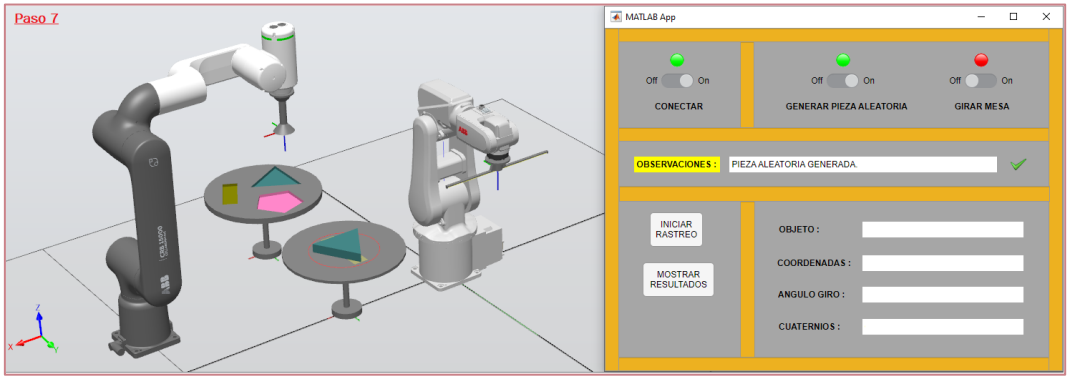

Figura 114. Paso 7: (Opcional) Generación de otra pieza para su posterior rastreo y emplazamiento.

CAPÍTULO 6

Conclusiones y líneas futuras

En este último capítulo se recogen las conclusiones finales, explicando a su vez la consecución de los objetivos propuestos. También se plantean una serie de líneas de cara al futuro con objeto de ampliar el proyecto y dotarlo de una mayor versatilidad.

6.1. Conclusiones

Acorde con los resultados obtenidos tras la elaboración del proyecto, podemos afirmar que se han alcanzado los objetivos propuestos. A continuación, se citan las presentes conclusiones:

- Se han empleado dos modelos de robots diferentes ("IRB 120" y "CRB 15000 GoFa"), estudiando sus características y parámetros de configuración con el fin de poder establecer una comunicación entre MATLAB y RobotStudio.

- Se ha creado un entorno de simulación en RobotStudio, dotado de diversos componentes inteligentes creados por el propio alumno. Los componentes creados y diseñados son los siguientes:

    - Prisma Rectangular.
    - Prisma Triangular.
    - Prisma Pentagonal.
    - Herramienta de detección de piezas compuesta por 24 sensores lineales empleada en el proceso de rastreo.
    - Herramienta que succiona las piezas, utilizada por el robot para depositar las piezas en sus respectivos huecos.
    - Mecanismo giratorio en torno al eje central de la propia mesa.
    - Mesa con huecos específicos para depositar finalmente las piezas.

    - Generador aleatorio de piezas.

- Se ha programado el código para cada controlador en el lenguaje RAPID. Esto permite a cada robot ejecutar los diversos movimientos y llevar a cabo la simulación.

- Se han diseñado en MATLAB los algoritmos necesarios para detectar el tipo de pieza y la orientación de esta, así como la programación específica para leer datos de un fichero que posteriormente es representado de forma gráfica.

- Otro de los objetivos era dotar al proyecto de un toque realista. Es por ello que se ha implementado una comunicación entre RobotStudio y MATLAB, haciendo uso del protocolo de comunicaciones OPC UA. Esto permite intercambiar información a tiempo real entre ambos softwares.

- Se ha diseñado y programado una interfaz gráfica de usuario, permitiéndole a este ejecutar la simulación de una forma más sencilla e intuitiva, logrando de esta manera una mayor versatilidad en el proyecto realizado.

- Se han aplicado diversos conocimientos de distintas áreas o ramas de la ingeniería que han sido adquiridos durante la etapa de formación en la Universidad de Valladolid.

6.2. Líneas futuras

Algunos de los cambios o mejoras a realizar del presente proyecto se citan a continuación:

- Ejecutar la simulación con un robot CRB 15000 GoFa de verdad. Esto no ha sido posible dado que la Universidad de Valladolid no cuenta todavía con un ejemplar. Sin embargo, se pretende comprar uno en los próximos años. Es por ello que se ha realizado el proyecto con este modelo de robot, pudiendo así estudiar las características y comprender su funcionamiento.

- Colocar las piezas sobre la mesa en una posición inicial distinta del centro de rotación. Se lograría dotar al proyecto de una mayor complejidad si cabe.

- Una nueva idea a desarrollar podría ser que el robot IRB 120 detectase la altura de las piezas con ayuda de alguna de las herramientas creadas, permitiendo de esta forma desarrollar piezas de diferentes alturas.

- Otra posibilidad sería incorporar en el robot IRB 120 una cámara, y haciendo uso de algoritmos de visión artificial, se podrían sustituir los veinticuatro sensores lineales empleados.

- Otro de los motivos por los que el presente proyecto se ha llevado a cabo de forma simulada serían las herramientas empleadas. El departamento no cuenta con este tipo de herramientas, siendo interesante su posible fabricación y mejora del diseño para futuros proyectos.

- Finalmente, se podría sustituir el robot IRB 120 por otro robot colaborativo, dado que estos son más seguros en caso de que interaccionen con seres humanos.

Bibliografía

[1] ABB, «Información detallada para: IRB 120,» [En línea]. Available: https://new.abb.com/products/es/3HAC031431-001/irb-120. [Último acceso: Mayo 2023].

[2] ABB, «GoFa™ CRB 15000,» [En línea]. Available: https://webshop.robotics.abb.com/es/catalog/product/view/id/249/s/gofa-crb-15000-vision-inspection/. [Último acceso: Mayo 2023].

[3] c. ETI, «Robótica,» [En línea]. Available: http://www.etitudela.com/profesores/rpm/rpm/downloads/robotica.pdf. [Último acceso: Mayo 2023].

[4] J. C. Fraile, *Introducción a la robótica,* Valladolid, 2022.

[5] S. Novus, «Ancient Origins,» 20 Diciembre 2021. [En línea]. Available: https://www.ancient-origins.es/artefactos-tecnologia-antigua/paloma-vapor-007391. [Último acceso: Mayo 2023].

[6] E. U. d. I. T. I. d. Zaragoza, «Ärea de Ingeniería de Sistemas y Automática,» [En línea]. Available: http://automata.cps.unizar.es/Historia/Webs/automatas_en_la_historia.htm. [Último acceso: Mayo 2023].

[7] T. timelines, «Edad Moderna,» [En línea]. Available: https://www.timetoast.com/timelines/edad-moderna-f909f23f-b5ff-43b8-9587-b6de28ea927c. [Último acceso: Mayo 2023].

[8] A. Barrientos, L. F. Peñín, C. Balaguer y R. Aracil, FUNDAMENTOS DE ROBÓTICA, Madrid: McGraw-Hill, 1997.

[9] EUROINNOVA, «clasificacion de los robots,» [En línea]. Available: https://www.euroinnova.edu.es/blog/clasificacion-de-los-robots. [Último acceso: Junio 2023].

[10] P. Gómez, «Las 5 Generaciones de la robótica,» [En línea]. Available: https://www.hipernexo.com/robotica/generaciones-robotica/.

[11] E. B. School, «Clasificación de los robots según su función,» 3 Agosto 2018. [En línea]. Available: https://www.esneca.com/blog/clasificacion-de-los-robots-segun-su-funcion/. [Último acceso: Junio 2023].

[12] R. F. Eléctrico, «Tipos de robots | ¿Cuántos existen y cómo se clasifican?,» [En línea]. Available: https://futuroelectrico.com/tipos-de-robots/. [Último acceso: Junio 2023].

[13] J. A. Ávila Herrero y A. Herreros López, «Modelado de una célula robótica con fines educativos usando el programa Robot-Studio,» 2015. [En línea]. Available: http://uvadoc.uva.es/handle/10324/14441. [Último acceso: Junio 2023].

[14] R. Vidal del Cura y A. Herreros López, «Célula multi-robot para el montaje de una luminaria, basado en robots ABB controlados mediante un interface de Matlab, empleando el protocolo de comunicaciones OPC,» 2022. [En línea]. Available: https://uvadoc.uva.es/handle/10324/53931. [Último acceso: Junio 2023].

[15] E. Pozas Mata y A. Herreros López, «Simulación de un Robot Colaborativo YuMi (ABB) en entorno RobotStudio comandado desde MATLAB mdiante protocolo OPC UA para tocar un Xilófono,» 2022. [En línea]. Available: https://uvadoc.uva.es/handle/10324/54231. [Último acceso: Junio 2023].

[16] ABB, «Manual del operador. RobotStudio,» [En línea]. Available: library.e.abb.com/public/d16da0665d004920a687e6dfc9876b68/3HAC032104%20OM%20RobotStudio-es.pdf?x-sign=YufjnWf5oLD+krp4ZD0e1VVVn6FNXt+faOwQaOR8qaS6YvvYOGrIatZcwRCAM1X8. [Último acceso: Junio 2023].

[17] ABB, «RobotStudio® Suite,» [En línea]. Available: https://new.abb.com/products/robotics/es/robotstudio. [Último acceso: Junio 2023].

[18] MathWorks, «MATLAB. Matemáticas. Gráficas. Programación.,» [En línea]. Available: https://es.mathworks.com/products/matlab.html. [Último acceso: Junio 2023].

[19] a. 2. c. d. f. t. p. l. industria, «Qué es OPC y cómo funciona,» [En línea]. Available: https://www.cursosaula21.com/que-es-opc-y-como-funciona/. [Último acceso: Junio 2023].

[20] ABB, «Conecta tus robots usando IoT Gateway (OPC UA o MQTT),» [En línea]. Available: https://new.abb.com/products/robotics/es/controladores/iot. [Último acceso: Junio 2023].

[21] ABB, «Familia de Controladores Omnicore,» [En línea]. Available: https://new.abb.com/products/robotics/es/controladores/omnicore. [Último acceso: junio 2023].