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 Colaborativo YuMi (ABB) en entorno RobotStudio comandado desde MATLAB mediante protocolo OPC UA para tocar un Xilófono.

Autora:

Pozas Mata, Elena

Tutor:

Herreros López, Alberto

Departamento de Ingeniería de Sistemas y Automática

Valladolid, julio, 2022

Resumen

El objetivo de este trabajo es programar un robot colaborativo multitarea a través del software de simulación y programación RobotStudio, con el fin de tocar un xilófono en función de las notas solicitadas desde MATLAB mediante el protocolo de comunicación OPC UA. Las teclas golpeadas por el robot son leídas por MATLAB mediante el mismo protocolo de comunicación para generar el sonido de la nota correspondiente en cada momento.

Para solicitar las notas de una manera más dinámica, se ha desarrollado una Interfaz Hombre-Máquina con la que se puede controlar toda la simulación.

Palabras clave

Robot IRB14000 YuMi, RobotStudio, MATLAB, OPC UA, Interfaz Hombre-Máquina

Summary

The main objective of this project is to program a multitasking collaborative robot by using the simulation and programming software RobotStudio, in order to play a xylophone based on the notes requested from MATLAB through the OPC UA communication protocol. The keys struck by the robot are read by MATLAB using the same communication protocol to generate the sound of the corresponding note at each moment.

To request the notes in a more dynamic way, a Human-Machine Interface has been developed, which allows to control the entire simulation.

Key Words

IRB14000 YuMi Robot, RobotStudio, MATLAB, OPC UA, Human-Machine Interface

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

Introducción y objetivos

El crecimiento de la robótica colaborativa en los últimos años se debe principalmente al éxito de sus aplicaciones industriales. Estas ventajas no solo pueden utilizarse en este ámbito, sino que también tienen cabida en aplicaciones artísticas, creativas e innovadoras, como la desarrollada para este trabajo, que aprovecha las capacidades de un robot colaborativo para realizar tareas que van más allá de la industria.

A lo largo del presente informe, se va a desarrollar y explicar la elaboración de la estación y la programación de un robot colaborativo multitarea, en la que se ha pretendido abarcar, de una manera original y entretenida, varias ramas de la ingeniería utilizando un protocolo de comunicación para leer y escribir diferentes variables declaradas en el programa RAPID del robot, desde una interfaz de usuario.

1.1. Objetivos del proyecto

Para la realización de este trabajo se ha escogido el robot colaborativo de dos brazos de la compañía ABB Robotics IRB 14000 YuMi (Figura 1).

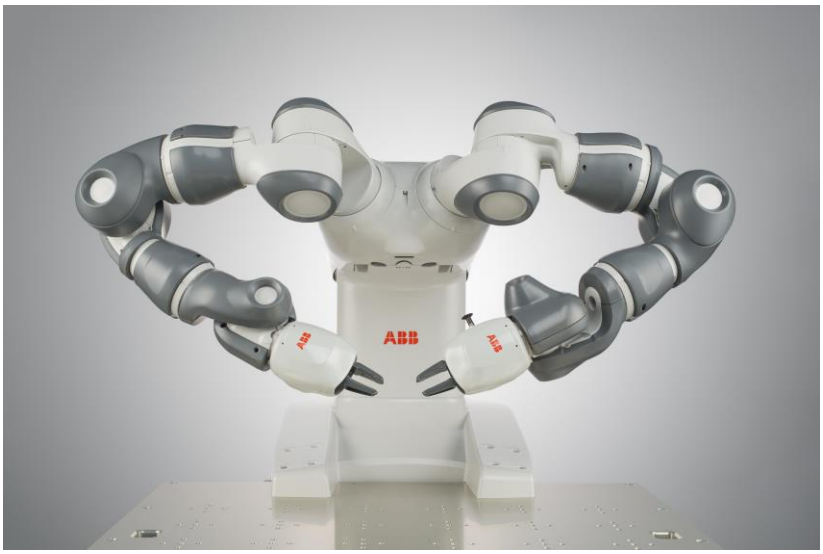


Figura 1. Robot IRB14000 YuMi.

El objetivo principal es programar este robot a través del software de simulación y programación RobotStudio, donde se crea de forma virtual el modelo de la estación, junto con la configuración de todas las entradas y salidas digitales del controlador necesario, y el código de programación que contendrá los movimientos e instrucciones a realizar por el robot, con el fin de tocar un xilófono en función de las notas solicitadas a través de la interfaz de usuario desarrollada con MATLAB. Las láminas golpeadas por el robot son leídas por MATLAB mediante el mismo protocolo de comunicación para generar el sonido de la nota correspondiente en cada momento. Por ello es necesaria la comunicación entre la interfaz y el robot.

Para la comunicación que se quiere establecer entre RobotStudio y MATLAB se ha escogido el protocolo de comunicación OPC UA, que permite intercambiar información y datos entre una máquina y otra de una manera simplificada.

De lo anteriormente expuesto, se extraen más esquemáticamente los siguientes objetivos relativos al presente Trabajo de Fin de Grado:

- Crear una estación en el entorno de simulación RobotStudio que cuente con los componentes necesarios para que el robot YuMi sea capaz de tocar un xilófono creado con este software.

- Programar en lenguaje RAPID el código necesario para definir los movimientos e instrucciones a realizar por el robot.

- Establecer una comunicación entre RobotStudio y MATLAB utilizando el protocolo OPC UA, con el fin de leer y escribir desde MATLAB variables declaradas en RobotStudio.

- Desarrollar con MATLAB una Interfaz Hombre-Máquina con la que se controle toda la simulación de un modo más dinámico.

- Implementar conocimientos en diferentes campos de la ingeniería.

- Dar a la robótica una aplicación artística, creativa e innovadora.

En la Figura 2 se puede ver un esquema donde se reflejan las acciones que se llevan a cabo en la comunicación entre RobotStudio y MATLAB.

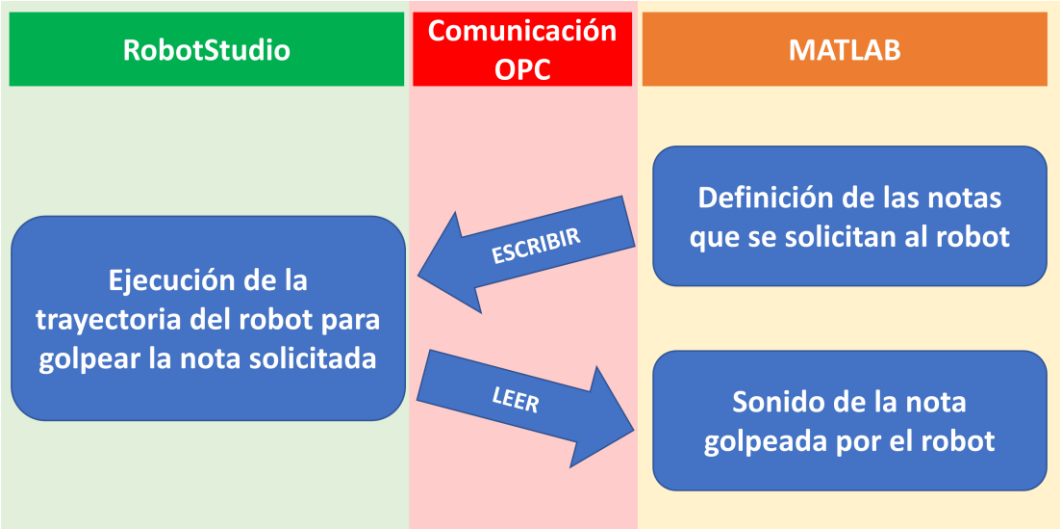


Figura 2. Esquema comunicación OPC entre RobotStudio-MATLAB.

A lo largo de los siguientes capítulos se muestran y explican las diferentes fases de las que ha constado este proyecto, así como todos los elementos que han sido necesarios para su realización.

1.2. Planteamiento del trabajo

Todo el proyecto se ha desarrollado de manera virtual, simulando en RobotStudio tanto los movimientos del robot como el xilófono, incorporando unos sensores de posición sobre las láminas para poder saber en qué momento y qué nota se ha golpeado.

En la Figura 3 se muestra un esquema que muestra la relación que se plantea de manera virtual entre los diferentes elementos de la estación, junto con la comunicación con MATLAB, es decir, la relación que se ha llevado a cabo para el presente trabajo.

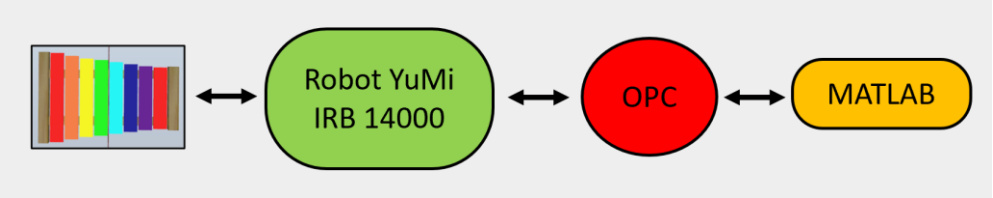


Figura 3. Simulación virtual Robot YuMi y comunicación OPC con MATLAB.

Sin embargo, la intención final de este proyecto es implementar sobre un robot real la ejecución de la simulación. Esto podría realizarse de dos formas distintas:

- Conservando la misma relación de elementos que en la simulación virtual, con un robot que golpee el xilofóno real conectado a la tarjeta de señales del robot para que active las entradas del controlador, a través de una señal discreta, en función de la nota que es golpeada, o bien,

- siendo el xilófono independiente de las entradas del controlador del robot, conectándose directamente a MATLAB y enviándole una señal continua en este caso.

Esta última nueva relación entre los diferentes elementos de la estación, ambos físicos, junto con la comunicación con MATLAB, elemento virtual, el cual funcionaría como sistema de identificación de sonidos, se puede ver en la Figura 4.

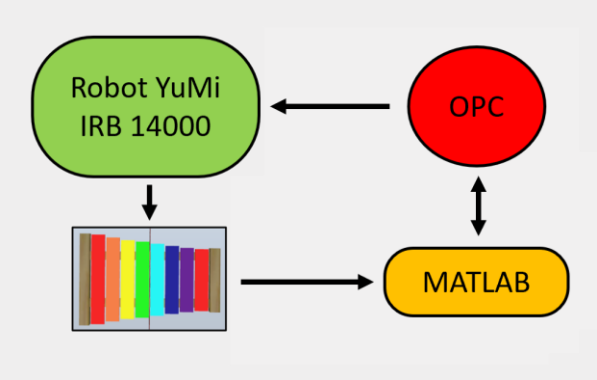


Figura 4. Posible comunicación entre MATLAB, el Robot y el Xilófono si la simulación fuera real.

Se tendrían dos comunicaciones reales a través de cables para los elementos físicos, y otra no tan real, con el protocolo OPC UA.

Esto no se ha podido llevar a cabo debido a que, actualmente, la Universidad de Valladolid no cuenta con un robot colaborativo YuMi, pero se pretende comprar uno para poder trabajar con él, siendo este proyecto un anticipo para ese futuro robot colaborativo.

A continuación, se muestran las ventajas que se tendrían con el mismo:

- Se puede posicionar de forma manual al punto deseado, sin necesidad de llevarlo únicamente a un punto mediante instrucciones RAPID a través de RobotStudio. Esto puede ser muy útil a la hora de definir las coordenadas de los puntos de la estación, ya que permite conseguir los valores buscados llevando al robot directamente al punto objetivo.

- Es más seguro. Los robots colaborativos están equipados con sensores para reducir su velocidad o detenerse en caso de entrar en contacto con alguien. Si se mueve el robot con la mano, se puede alejar del usuario en caso de que haya algún fallo en el programa o una persona se

encuentre demasiado cerca de él, a diferencia del robot actual con el que cuenta la Universidad, el IRB120, que no puede ser utilizado por un alumno sin supervisión por seguridad.

- Para el caso del presente trabajo, suponiendo que la sensibilidad del robot fuese suficiente para detectar el golpe con la tecla, el robot no tendría que recibir ninguna señal del xilófono, ya que un robot colaborativo puede detectar que está golpeando algo. Sin embargo, en simulación un sensor no detecta esfuerzos. Se pueden leer esfuerzos sobre el robot real, pero no sobre el robot simulado.

1.3. Estructura de la memoria

El contenido del proyecto se ha estructurado en seis capítulos, de la siguiente forma:

- **Capítulo 1: Introducción y objetivos,** en el que se presentan los objetivos principales y la justificación de la realización del proyecto.

- **Capítulo 2: Marco teórico y estado del arte.** En este capítulo se recopila información relacionada con la robótica, para poder entender el contexto actual sobre el tema.

- **Capítulo 3: Metodología y software utilizado,** donde se especifican los diferentes campos de la ingeniería a los que pertenecen los conocimientos desarrollados para la realización de este proyecto, y se describe el software utilizado para cada uno de ellos.

- **Capítulo 4: Desarrollo del proyecto,** donde se describen las diferentes fases de las que se constituye el proyecto.

- **Capítulo 5: Resultados.** Se muestra el resultado final de la simulación en RobotStudio, junto con la utilización de la interfaz de usuario diseñada en MATLAB.

- **Capítulo 6: Conclusiones y líneas futuras,** donde se expone una valoración de los resultados obtenidos y se analiza el grado de cumplimiento de los objetivos del proyecto. Además, se plantean futuras líneas de trabajo relacionadas con el proyecto realizado.

CAPÍTULO 2

Marco teórico y estado del arte

En este capítulo se recopila información relacionada con el ámbito en el que se puede enmarcar este proyecto, para poder entender el contexto actual sobre el tema.

2.1. Marco teórico

En el primer tercio del siglo XX se inicia el desarrollo de la ingeniería en sus diferentes ramas, (mecánica, electrónica, informática y telecomunicaciones), que van a permitir la construcción de robots modernos. La construcción de máquinas que imitan las tareas humanas se remonta a la antigüedad [1].

Isaac Asimov utilizó por primera vez el término "robótica" en los relatos cortos reunidos en su libro "I Robot" ("Yo robot"), publicado en 1950. En el relato titulado "Runaround", ambientado en el año 2056, se postulan las tres leyes de la robótica, expuestas en la Tabla 1.

LEYES DE LA ROBÓTICA	
PRIMERA	Un robot no debe dañar a un ser humano ni, por su pasividad, dejar que un ser humano sufra daño.
SEGUNDA	Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto cuando estas órdenes están en oposición con la primera Ley.
TERCERA	Un robot debe proteger su propia existencia, hasta donde esta protección no esté en conflicto con la primera o segunda ley.

Tabla 1. Las tres leyes de la robótica según Asimov.

Asimov consideró necesario añadir una cuarta ley, antepuesta a las demás, la denominada ley número cero, que afirma que un robot no debe actuar simplemente para satisfacer intereses individuales, sino que sus acciones deben preservar el beneficio común de toda la humanidad.

En 1983, H. Roselund y W.Pollard, construyeron el primer brazo articulado (o manipulador) para pintura al spray, lo que representó una nueva forma de entender la producción industrial al incorporar robots a las cadenas de producción [2].

En 1939 se presentan diversas novedades en robótica popular, como el robot humanoide Elektro (Figura 5), y el perro mecánico Sparko, los cuales eran atracciones en la feria mundial de Nueva York celebrada aquel año.

Figura 5. Robot humanoide Elektro.

Se trataba de máquinas que realizaban movimientos simples, de escasa articulación, con rutinas mecánicas repetitivas. Con estos robots surgen en esta época dos de las grandes ramas de la robótica moderna: la rama industrial y la rama del entretenimiento.

En 1947, D.S. Halder, de la compañía automovilística Ford de Detroit, acuña el término "automatización" y pone en marcha una estrategia para ir sustituyendo al ser humano de muchas de las tareas del proceso de fabricación de automóviles.

G.C. Devol, uno de los pioneros de la robótica industrial, patenta en 1956 un controlador que registraba señales eléctricas por medio de magnetos que accionan un dispositivo mecánico, logrando una máquina flexible, adaptable al entorno y fácil de manejar. A partir de esto es cuando puede denominarse robot a una máquina. Los aspectos relacionados con el control son fundamentales para la asignación de tareas a un robot.

En el año 1962 ocurren diversos acontecimientos de gran importancia para la robótica. H.A. Enst publica un trabajo sobre sensores táctiles MH-1 aplicados a una mano robotizada de tipo ANL 8, dotada de 6 grados de libertad y de un procesador TX-0 que revoluciona el sector e inicia el desarrollo de los sensores y la retroalimentación, demostrando la conducta adaptativa de un robot por primera vez en la historia [3].

Por su parte R. Tomovic y G. Boni desarrollan una mano con sensores de presión que proporciona una señal de realimentación de entrada al motor para

iniciar uno de los dos modelos de aprehensión disponibles en función del peso del objeto.

Al mismo tiempo C.A. Petri desarrolla un sistema de redes para el diseño y análisis de automatismos secuenciales y concurrentes, en el que son tan importantes los componentes mecánicos y eléctricos como los sistemas de control, que se aplica en el análisis y modelado de sistemas, no sólo en el campo de la automática, sino también en el de la informática y las comunicaciones [4].

En 1976, el programa espacial norteamericano Viking aterriza en Marte y pone en marcha un brazo manipulador capaz de recoger muestras del suelo y tomar imágenes en detalle. El robot PUMA (Figura 6), desarrollado en 1978, comienza a trabajar para General Motors en tareas de montaje.

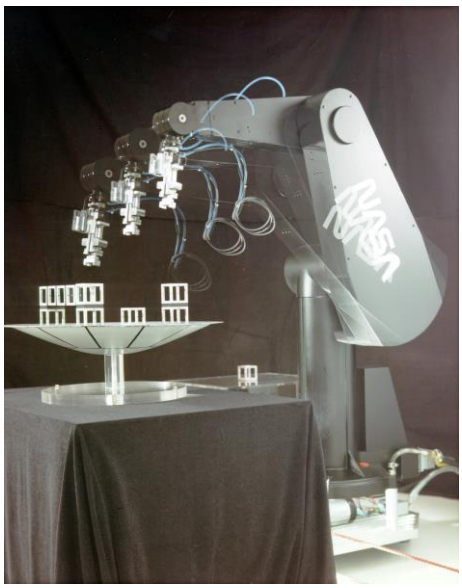


Figura 6. Robot PUMA (Programmable Universal Machine for Assembly).

En 1997, Honda presenta el P-3, un androide de 1,60 m de altura y 130 Kg de peso, capaz de caminar con seguridad sobre dos piernas, imitando la marcha humana gracias a sensores de gravedad, visuales, de detección de ángulos y de aceleración. Entre sus habilidades figuran las de pasear por terreno llano, subir y bajar escaleras, abrir puertas, pulsar interruptores y empujar obstáculos.

Los androides de tipo dinámico pasivo como SIGMO, QRIO, ASIMO (Figura 7) y Hubo son capaces de caminar, entablar conversación (con evidentes limitaciones) y realizar algunas tareas simples.

Figura 7. Robot ASIMO.

Estos solo son algunos de los proyectos que se han desarrollado a lo largo de la historia, los cuales se han ido mejorando y evolucionando hasta llegar al momento actual. La presencia de millones de robots en todo el mundo, en sectores muy diferentes, nada tiene que ver con las 3.500 unidades que funcionaban en 1974.

Tipos generales de robots

A continuación, se van a clasificar los diferentes robots que existen, de varias formas diferentes [5].

- Clasificación por tipos:
    - Androides y zoomórficos: los androides están diseñados con la idea de crear robots análogos al hombre, como el mostrado en la Figura 7. Se destinan fundamentalmente al estudio y experimentación. Los zoomórficos tienen forma de animales y se intenta conseguir con ellos alguna de las facultades que tienen los animales.
    - Móviles: están provistos de patas, ruedas u orugas que los capacitan para desplazarse en función de su programación de la información que reciben. Pueden llevar diversos sistemas de sensores para captar información.

- De servicio: en esta clasificación podrían entrar todos los robots no industriales, como los robots de limpieza o de servicios médicos.

- Industriales: están destinados a realizar de forma automática determinados procesos de fabricación o manipulación.

- Clasificación según la configuración de sus ejes:
  Viene determinada por el tipo de las tres primeras articulaciones, que son las que determinan la posición de la herramienta en el espacio y el tipo de coordenadas con las que se determina esta posición o localización.

  - Robot polar o esférico: la primera y segunda articulación son de ejes de rotación perpendiculares entre sí, y la tercera es prismática, es decir, se tienen dos giros y un desplazamiento, que permiten posicionar un punto en el espacio mediante coordenadas polares.

  - Robot cilíndrico: utiliza un giro en la base y dos desplazamientos perpendiculares entre sí, para determinar la posición de los puntos por medio de coordenadas cilíndricas. Se controla fácilmente y es rápido, pero solo se usa para casos en que no haya obstáculos en su zona de trabajo y el acceso a ella se haga horizontalmente.

  - Robot cartesiano: sus tres articulaciones principales son prismáticas y los ejes ortogonales entre sí. Son rápidos, muy precisos, de fácil control, amplia zona de trabajo y elevada capacidad de carga, pero ocupan mucho espacio relativo. Se utilizan en aplicaciones que requieren movimientos lineales de alta precisión.

  - Robot SCARA: robot con dos articulaciones de rotación y una prismática. Es rápido, barato y preciso, pero solo tiene accesibilidad a zonas de trabajo que estén en planos perpendiculares a su eje vertical.

  - Robot angular o antropomórfico: tiene sus tres principales articulaciones de tipo rotacional, por lo que emplea las coordenadas angulares para determinar las posiciones de su elemento terminal. Se llama antropomórfico porque simula los movimientos de un brazo humano.

Otro tipo de clasificación de robots, en función de su modalidad, podría ser el robot colaborativo y el robot no colaborativo.

Las aplicaciones colaborativas son diferentes a las de los sistemas robóticos tradicionales, ya que las personas pueden trabajar próximos al sistema del robot cuando está en funcionamiento colaborativo y se permite el contacto físico humano-robot bajo ciertas condiciones, sin necesidad de aislar el robot mediante resguardos o vallas [6].

Se pueden distinguir diferentes modos de colaboración en este tipo de robots [7]:

- Parada de robot con reinicio automático:

  La premisa es que, en el espacio compartido con una persona, el robot no se mueva bajo ninguna circunstancia.

- Guiado manual:

  El robot colaborativo y el operario trabajarán conjuntamente de forma segura y ergonómica para realizar tareas que requieren precisión humana. En esta zona de colaboración, el operario es quien dirige el movimiento del robot con un guiado manual, hacia la posición en la que resto de zonas el robot trabaja en modo normal, con las seguridades necesarias.

- Monitorización de velocidad y posición:

  El robot colaborativo y el operario se mueven simultáneamente por dentro del mismo espacio. Si la distancia que los separa se reduce, el robot se detiene, pasando a estar en el modo de “Parada de seguridad monitorizada”.

- Limitación de fuerza y potencia:

  Es el modo más interesante, ya que permite la interactuación más directa entre personas y el robot. Las velocidades y la potencia del robot colaborativo están limitadas.

Para este trabajo se ha escogido un modelo de robot colaborativo de la compañía ABB. Como se puede ver en la Figura 1, este robot consta de dos brazos robóticos.

Cada brazo cuenta con siete ejes, lo que le permite realizar una gran variedad de movimientos, para alcanzar los puntos deseados. A la característica de poder trabajar colaborativamente con humanos, se le une la posibilidad de sincronizar sus brazos.

El espacio de trabajo que alcanza este robot se puede ver en la Figura 8, donde aparece representado para diferentes vistas, como son el alzado frontal, la vista isométrica, la vista en planta y el alzado lateral del robot [8].

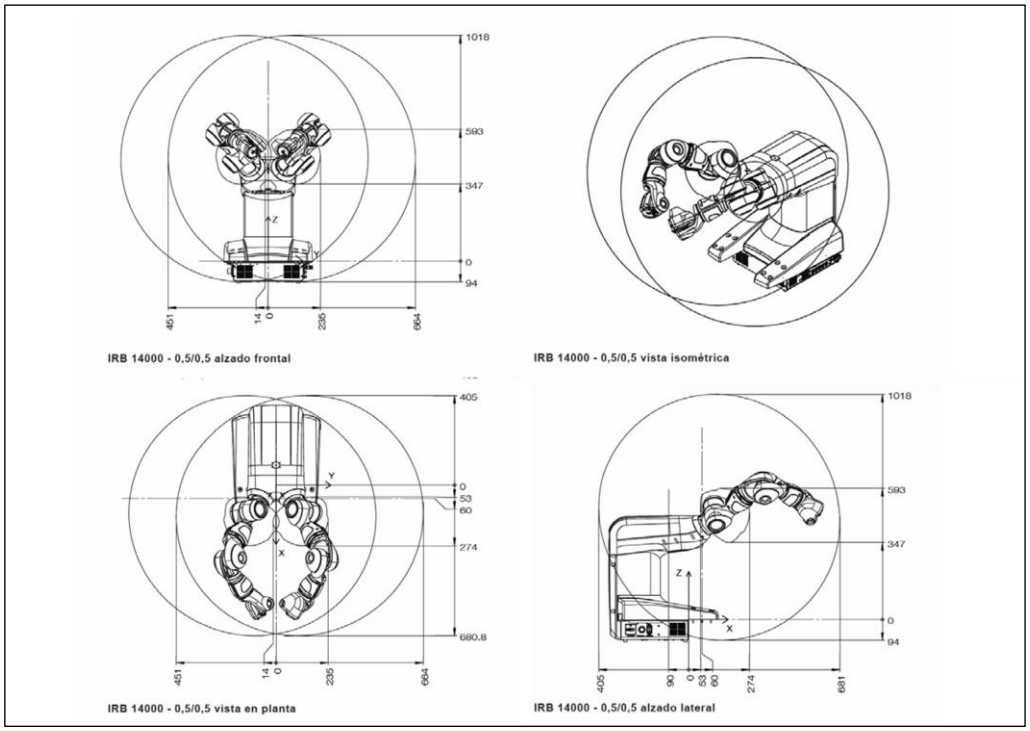


Figura 8. Data Sheet, Vistas robot YuMi ABB.

A continuación, se muestran los datos fundamentales que aparecen en la hoja de datos del robot YuMi para poder definir en su totalidad a éste. En la Tabla 2 se muestran sus características, en la Tabla 3 el movimiento de cada eje, definiendo la amplitud de trabajo y la velocidad máxima de cada uno de ellos, y en la Tabla 4 aparecen algunos resultados.

Características técnicas			
Versión del robot	Alcance	Carga útil	Carga del brazo
IRB 14000 - 0,5/0,5	500 mm	500 g	Sin carga en el brazo

Características	
Alimentación y señal integradas	24 V Ethernet o 4 señales
Suministro de aire integrado	1 por brazo en el borde de la herramienta (4 bar)
Ethernet integrado	Un puerto ethernet 100/10 base-TX por brazo
Repetibilidad de posición	0.02
Montaje del robot	Banco
Grado de protección	IP30
Controladores	Integrados

Características físicas	
Total parte inferior	399 mm * 496 mm
Punteras	399 mm * 134 mm
Peso	38 kg

Tabla 2. Data Sheet, Características robot YuMi ABB.

Movimiento		
Movimiento de los ejes	**Amplitud de trabajo**	**Velocidad máxima**
Rotación eje 1	-168,5° a 168,5°	180°/s
Brazo eje 2	-143,5° a 43,5°	180°/s
Brazo eje 3	-123,5° a 80,0°	180°/s
Muñeca eje 4	-290,0° a 290,0°	400°/s
Flexión eje 5	-88,0° a 138,0°	400°/s
Giro eje 6	-229,0° a 229,0°	400°/s
Rotación eje 7	-168,5° a 168,5°	180°/s

El orden físico de ejes es 1, 2, 7, 3, 4, 5, 6

Tabla 3. Data Sheet, Movimiento robot YuMi ABB.

Resultados	
Ciclo de recogida 0,5 kg	
25* 300 * 25 mm	0,86 s
Velocidad de TCP máxima	1,5 m/s
Aceleración de TCP máxima	11 m/s*s
Tiempo de aceleración 0-1 m/s	0,12 s

Tabla 4. Data Sheet, Resultados robot YuMi ABB.

2.2. Estado del arte

Una vez se ha introducido el marco teórico en el que se ha desarrollado el presente proyecto, en este apartado se muestran algunos de los trabajos realizados previamente en esta materia en la Universidad de Valladolid.

En 2015, Gonzalo Muinelo Garrido realizó un trabajo final de grado que consistía en la simulación de una célula robotizada, para el tratamiento de una pieza de aluminio. Esta célula se muestra en la Figura 9. El objetivo era la programación del robot, para poder gestionar las piezas, ya que tenían que pasar por tres procesos distintos, cada uno con una determinada máquina.

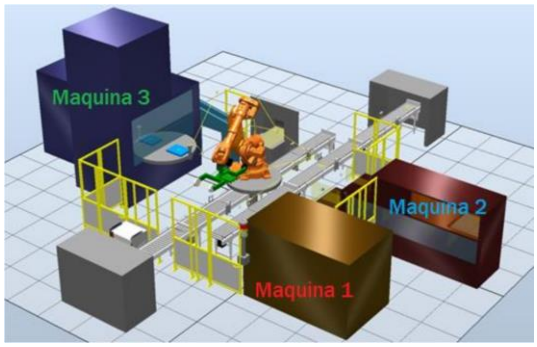


Figura 9. Célula de Trabajo, TFG de Gonzalo Muinelo Garrido [9].

En ese año mismo año, Juan Antonio Ávila Herrero diseñó otra célula robótica enfocada en este caso con fines educativos, ya que se diseñaban una serie de prácticas para aprender a manejar el software RobotStudio, programando el robot para que fuera capaz de jugar a las tres en raya, o de escribir en una mesa inclinada. Esta célula robótica se muestra en la Figura 10.

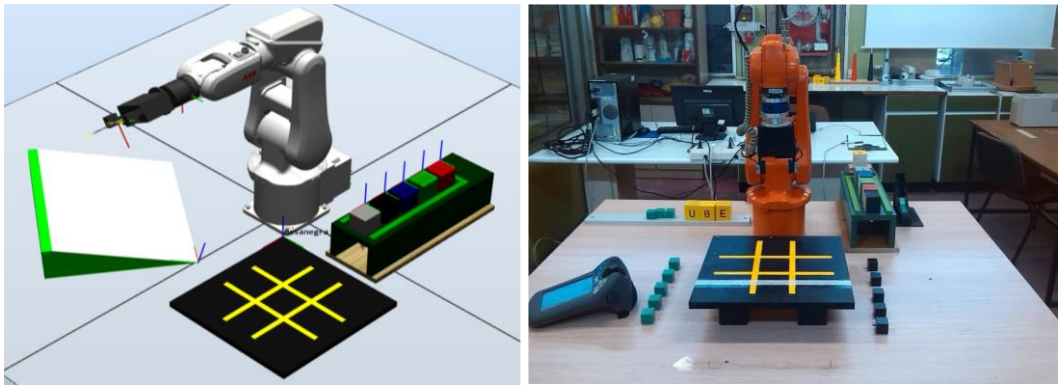


Figura 10. Célula de Trabajo, TFG de Juan Antonio Ávila Herrero [10].

En el 2019, Carlos Jiménez Jiménez desarrolló un sistema robótico educativo con el objetivo de poder jugar al ajedrez contra un robot industrial, el cual se puede ver en la Figura 11. El usuario sería el encargado de definir sus jugadas a través de la interfaz desarrollada, y el robot se encargaría de desplazar las piezas por el tablero.

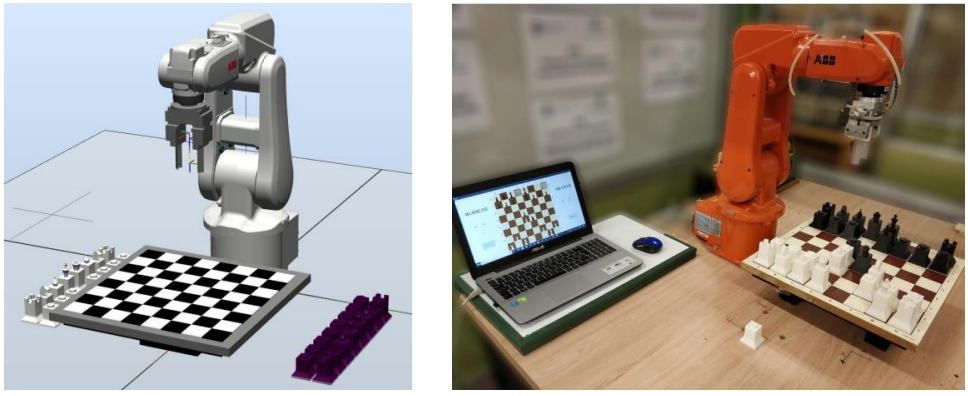


Figura 11. Célula de Trabajo, TFG de Carlos Jiménez Jiménez [11].

En 2020, Víctor Lobo Granado desarrolló una serie de elementos, como barras de diferentes tamaños, un cilindro y una semiesfera con una serie de agujeros equiespaciados, proponiendo una serie de prácticas en las que dar uso a estos componentes. Estos elementos pueden verse en la Figura 12.

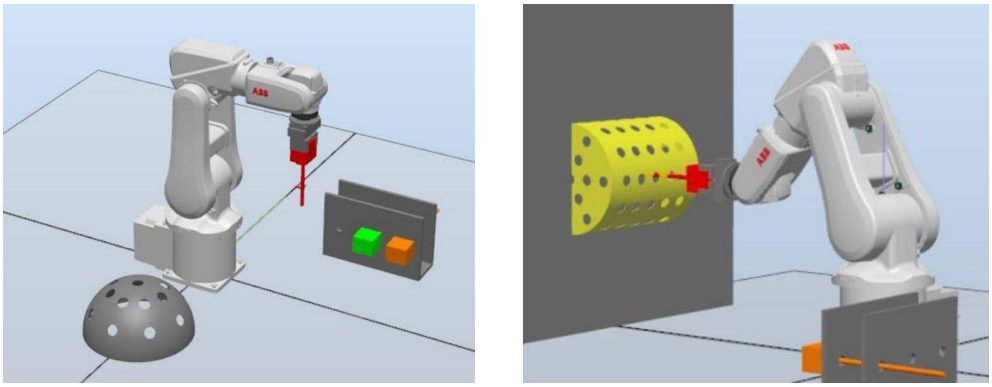


Figura 12. Célula de Trabajo, TFM de Víctor Lobo Granado [12].

Finalmente, en el año 2021 Rodrigo Sancho García desarrolló una célula de trabajo robótica que contaba con dos robots, y que tenía como objetivo realizar dos tareas. La primera, un ensamblaje automático por piezas de un coche de juguete, y la segunda, detectar el tamaño, la posición o el color de una serie de cubos. Esta célula se puede ver en la Figura 13.

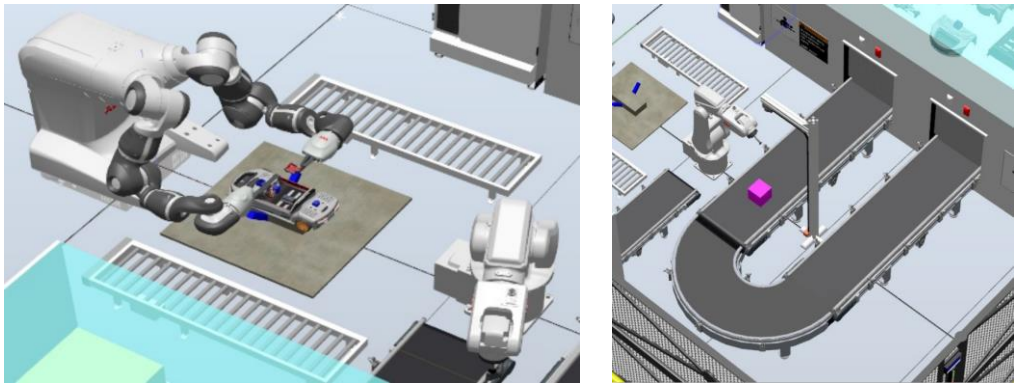


Figura 13. Célula de Trabajo, TFG de Rodrigo Sancho García [13].

CAPÍTULO 3

Metodología y software utilizado

En este proyecto, se ha buscado integrar varias ramas de la ingeniería, como son la robótica y las comunicaciones industriales. Para conseguir esto, se han utilizado varios softwares diferentes.

Para la parte de la robótica, tanto para el modelado de la estación como la programación en lenguaje RAPID se ha utilizado el software de simulación y programación offline de ABB, RobotStudio, con el que se ha podido diseñar la estación de trabajo, y simular y programar los movimientos del robot multitarea YuMi.

Este programa se basa en ABB VirtualController, una copia exacta del software real que ejecuta sus robots en producción. Esto permite realizar simulaciones muy realistas, utilizando programas de robot reales y archivos de configuración idénticos a los utilizados en el taller, aunque para este trabajo solo se realizará la parte virtual, ya que en la universidad todavía no se cuenta con un robot YuMi de dos brazos.

Para la parte de las comunicaciones se han utilizado dos softwares distintos:

- MATLAB para crear la interfaz Hombre-Máquina con la aplicación "AppDesigner",
- y ABB IRC5 OPC para la conexión OPC, con el que se ha podido sincronizar y coordinar MATLAB y RobotStudio.

3.1. Descripción del software

RobotStudio

Es el software de simulación y programación offline de ABB, que ofrece una réplica digital completa (Digital Twin) de activos o sistemas físicos para que pueda ver lo que sucede en su línea de producción de forma remota [14].

La herramienta de programación offline RobotStudio de ABB permite a los usuarios crear, simular y probar una instalación completa de robot en un entorno virtual 3D sin tener que visitar o perturbar su línea de producción real.

Permite crear, programar y simular células y estaciones de robots industriales de la compañía ABB. Es un simulador comercial potente, con diversas características y capacidades, entre las que destacan [15]:

- Creación automática de cualquier tipo de estación.
- Importación de geometrías y modelos 3D de cualquier formato (RobotStudio trabaja sobre CATIA).
- Programación y simulación cinemática de las estaciones.
- Facilidad de diseño y creación de células robóticas (robot y dispositivos).
- Permite exportar los resultados obtenidos en simulación a la estación real.

Además, proporciona un entrono virtual muy realista que permite simular de manera muy precisa una aplicación o un proceso real.

El simulador RobotStudio funciona sobre RobotWare, el cual se instala junto con RobotStudio, y proporciona el conjunto de archivos necesarios para implementar todas las funciones, configuraciones, datos y programas requeridos para el control del sistema del robot.

El lenguaje de programación de alto nivel para controlar robots industriales ABB, utilizado por este software, se denomina RAPID. Las características de este lenguaje de programación son las siguientes:

- Parámetros de rutina:
    - Procedimientos: utilizados como un subprograma.
    - Funciones: devuelven un valor de un tipo específico y se utilizan como argumento de una instrucción.
    - Rutinas TRAP: un medio para responder a las interrupciones.
- Expresiones aritméticas y lógicas.
- Manejo automático de errores.
- Programas modulares.
- Multitarea.

MATLAB

Es una plataforma de programación y cálculo numérico utilizada por millones de ingenieros y científicos para analizar datos, desarrollar algoritmos y crear modelos [16].

Combina un entorno de escritorio perfeccionado para el análisis iterativo y los procesos de diseño con un lenguaje de programación que expresa las matemáticas de matrices y arrays directamente.

Una de las prestaciones con las que cuenta MATLAB es la aplicación “AppDesigner”, para crear apps web y de escritorio en MATLAB, la ha sido utilizada para la realización de la interfaz de usuario de este trabajo. Permite crear apps profesionales de una manera sencilla, arrastrando y colocando los componentes visuales para crear el diseño de una interfaz gráfica de usuario

(GUI) y usando el editor integrado para programar rápidamente su comportamiento.

Integra las dos tareas principales en la creación de una app:

- la distribución de los componentes visuales de una interfaz gráfica de usuario (GUI),
- y la programación del comportamiento de la app.

Además, también ofrece elementos de control como medidores, indicadores luminosos, controles y conmutadores que permiten replicar el aspecto y las acciones de los paneles de instrumentación.

ABB IRC5 OPC

La aplicación de configuración del servidor ABB IRC5 OPC UA se utiliza para crear y administrar alias para los controladores ABB IRC5 Robot [17].

Un Alias es un descriptor fácil de usar que representa una interfaz de comunicaciones con el controlador de robot ABB IRC5. Se debe crear un alias para cada controlador de robot al que accederá el servidor ABB IRC5 OPC UA.

Gracias a esta aplicación se puede establecer la comunicación entre RobotStudio y MATLAB. OPC UA (Open Platform Communications-Unified Architecture) es un estándar abierto para la comunicación horizontal de máquina a máquina (M2M) y para la comunicación vertical de máquina a nube (Figura 14). Es independiente del proveedor y de la plataforma, admite amplios mecanismos de seguridad y puede combinarse de forma óptima con PROFINET en una red Ethernet industrial compartida [18].

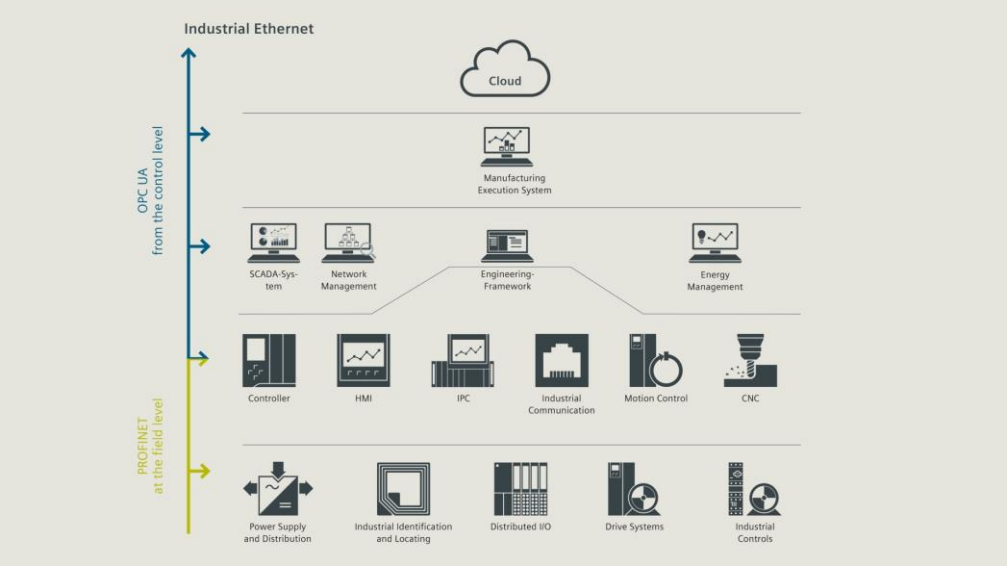


Figura 14. Niveles de Ethernet Industrial.

En este trabajo se ha utilizado OPC UA con comunicación cliente/servidor, la cual es ampliamente utilizada en automatización. Con este eficaz mecanismo de comunicación one-to-one, cada cliente OPC UA obtiene acceso a los datos del servidor OPC UA a través de una comunicación punto a punto. El cliente OPC UA envía una solicitud al servidor OPC UA, del que recibe una respuesta.

A continuación, se muestran algunas de las principales razones por las que se ha seleccionado el protocolo OPC UA para la implementación de este trabajo:

- Características:
    - Independiente de vendedor y neutral en cuanto a plataforma.
    - Concepto de seguridad integrado (cifrado, firma y autentificación).
    - Consistente, de extremo a extremo y escalable.
    - Modelo de información y servicios semánticos.
    - Funcionamiento en paralelo ilimitado con PROFINET.
    - Capacidad en tiempo real de OPC UA PubSub en combinación con Time-Sensitive Networking (TSN).
- Beneficios:
    - Interfaz estandarizada y disponibilidad extendida.
    - Comunicación segura directamente en el protocolo, sin necesidad de hardware adicional.
    - Conexión y comunicación directas a través de todos los niveles de automatización.
    - Interpretación sencilla e inequívoca de los datos.
    - Redes sencillas basadas en Ethernet, que utilizan la infraestructura de Ethernet industrial existente.
    - Interfaces estandarizadas internacionalmente para una integración sencilla de la máquina (especificaciones complementarias).
    - Alto rendimiento gracias a una comunicación rápida.
    - Cuando se utiliza hardware TSN y OPC UA PubSub, los datos OPC UA pueden transmitirse de forma determinística, independientemente de la carga de la red.

Mientras que el protocolo OPC convencional podía ocasionar fallos de vulnerabilidad, la nueva versión con arquitectura unificada resuelve este conflicto y aporta una mayor compatibilidad. Con el sistema OPC convencional únicamente nos limitábamos a enviar datos entre sistemas SCADA y sensores, mientras que, con OPC UA, podemos enviar datos a cualquier tipo de sistema de una manera más sencilla y segura [19].

CAPÍTULO 4

Desarrollo del proyecto

En este capítulo se detalla en profundidad cómo se ha desarrollado el trabajo, describiendo las diferentes fases de éste, entre las que destacan la programación en RobotStudio, la programación en RAPID, y el establecimiento de la comunicación OPC UA entre RobotStudio-MATLAB.

4.1. PRIMERA FASE: Programación RobotStudio

En este apartado se explicará cómo se ha modelado la estación de trabajo del robot, así como el diseño de los componentes inteligentes de los que se compone la estación, la configuración del controlador virtual, y la lógica de la estación que se ha utilizado para que todos los componentes inteligentes cumplan su función.

4.1.1. Modelado de la estación

El primer paso a la hora de utilizar el software de RobtStudio, es crear una solución con estación vacía para partir desde cero, y añadir poco a poco el controlador y los demás elementos de los que se compone la estación.

A continuación, se abre la Biblioteca ABB, situada en la pestaña “Posición inicial”, donde se encuentran todos los robots, posicionadores y tracks de ABB disponibles para la simulación en RobotStudio, y se selecciona el robot colaborativo IRB 14000 YuMi (Figura 15) que se utilizará en esta estación.

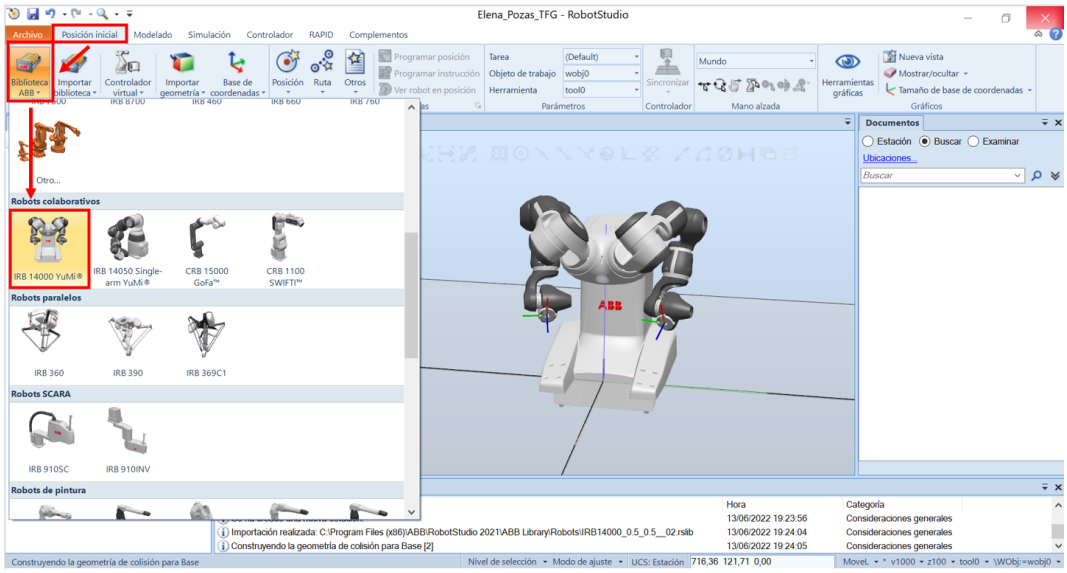


Figura 15. Importar robot colaborativo IRB 14000 YuMi de la biblioteca ABB.

Posteriormente, se crean las herramientas del robot, una para cada brazo. Para ello, se importan las herramientas ya diseñadas por ABB para este robot, situadas nuevamente en la pestaña “Posición inicial”, y seleccionamos “Equipamiento” desde “Importar biblioteca” (Figura 16).

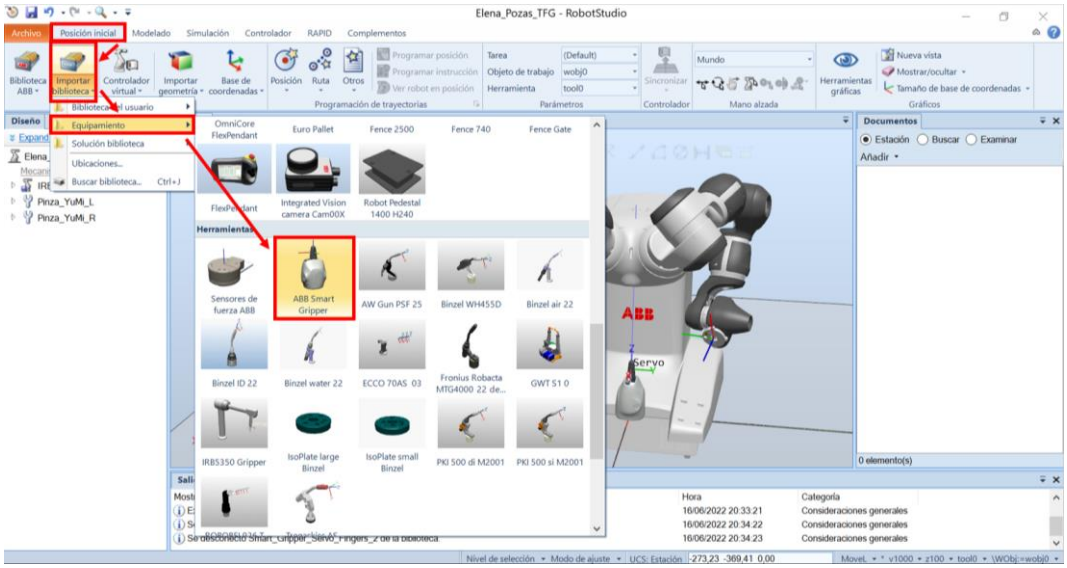


Figura 16. Importar herramientas ABB Smart Gripper del equipamiento de ABB.

Con el objeto de situarlas en la posición correcta, se arrastran las herramientas hasta el brazo correspondiente del robot, y se actualiza la posición de estas. Automáticamente se colocarán en el eslabón 7 del robot (Figura 17).

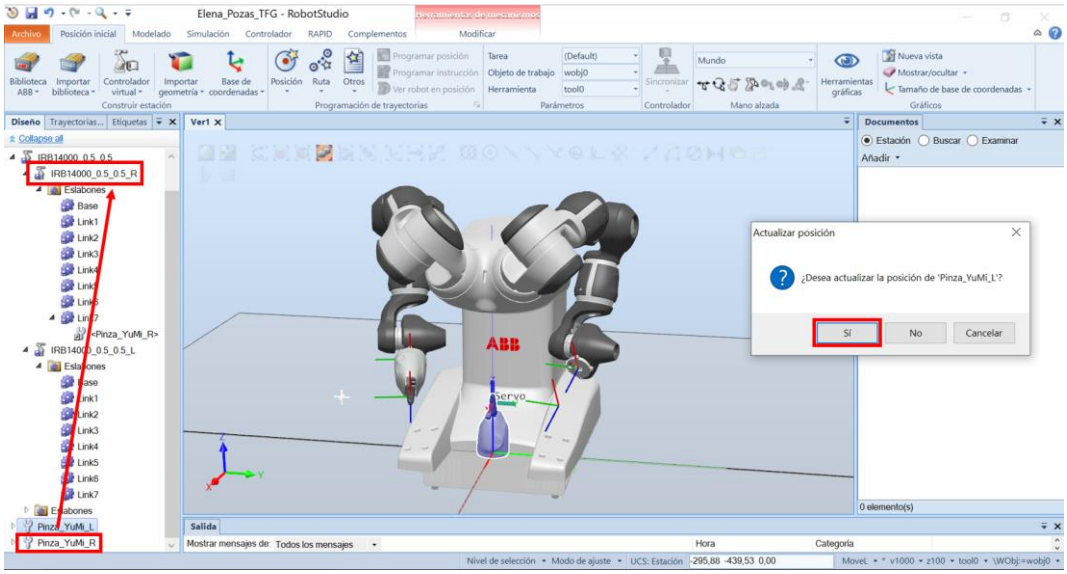


Figura 17. Colocar herramientas en el robot.

Para modelar el xilófono en la estación, se utilizan las opciones de modelado propias del software RobotStudio. En la pestaña de "Modelado", se crean 8 tetraedros, uno para cada lámina del xilófono, correspondiente a un tono de la escala diatónica, y 4 tetraedros para la base del xilófono (Figura 18).

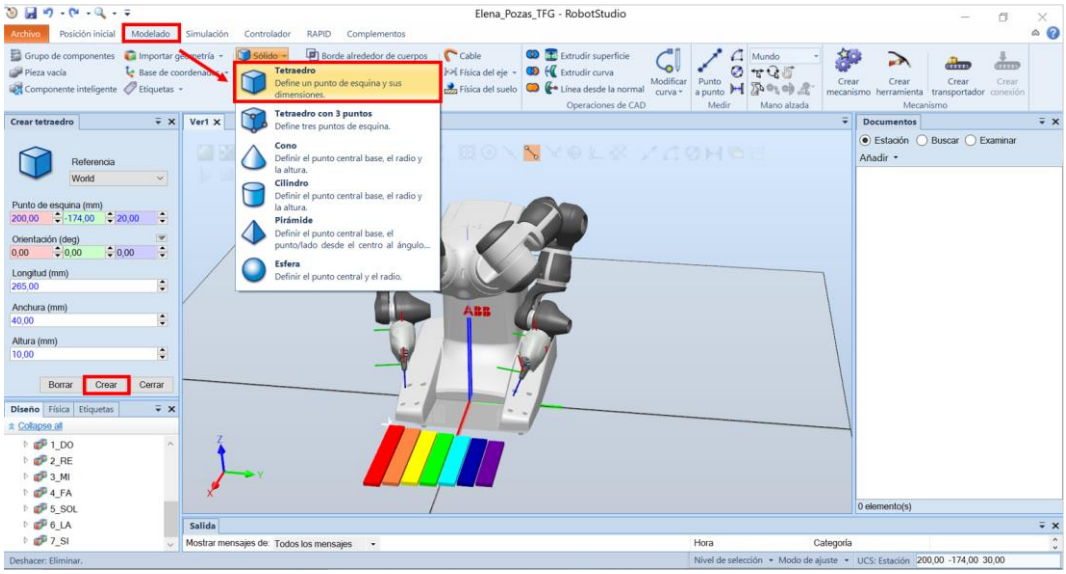


Figura 18. Crear tetraedro.

Se coloca delante del robot, en una posición centrada respecto a la base de coordenadas del robot. Cada lámina tendrá un color y una dimensión determinada, cuyo centro estará alineado en el eje y para diferenciar cada tono, y generar el sonido adecuado (Figura 19).

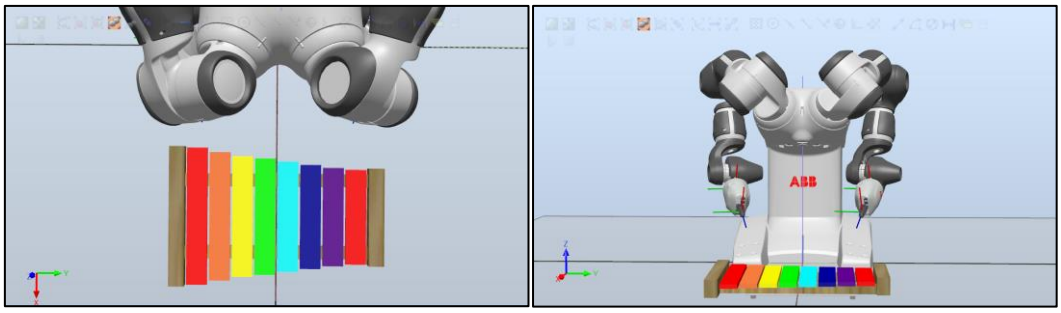


Figura 19. Resultado de modelar el xilófono.

Para poder situar el objeto en la posición deseada con mayor facilidad, se reposiciona el sistema de coordenadas local seleccionando el objeto deseado y, en la pestaña “Modificar”, se selecciona la herramienta de pieza “Establecer origen local” (Figura 20).

Figura 20. Establecer origen local

El tamaño de la lámina determina la altura del sonido. A mayor tamaño su sonido es más grave, y viceversa, a menor tamaño su sonido es más agudo.

Es por ello que las dimensiones escogidas para las láminas del xilófono son las siguientes:

- Distancia entre cada lámina: 4 milímetros.
- Alto: 10 milímetros.
- Ancho: 40 milímetros.
- Largo (Tabla 5):

DO	265
RE	248
MI	233
FA	225
SOL	213
LA	201
SI	190
DO	183

Tabla 5. Dimensiones en milímetros del largo de cada lámina.

Para crear las baquetas, se utilizan las operaciones de CAD que se pueden encontrar también en la pestaña de “Modelado”. Primero se crea una esfera de 20 milímetros y un cilindro de unos 200 milímetros. Posteriormente se crea un nuevo cuerpo a partir de la unión de estos. Además, para que el robot pueda coger las baquetas, se ha modelado un soporte para las mismas, mediante el uso de la operación restar, la cual crea un nuevo cuerpo sustrayendo un cuerpo de otro. De este modo, se resta la esfera a la base del soporte y el cilindro a la sujeción superior (Figura 21). Las baquetas se colocan delante del xilófono para que el robot tenga un correcto alcance y sea capaz de cogerlas y dejarlas sin ningún tipo de problema sobre su base.

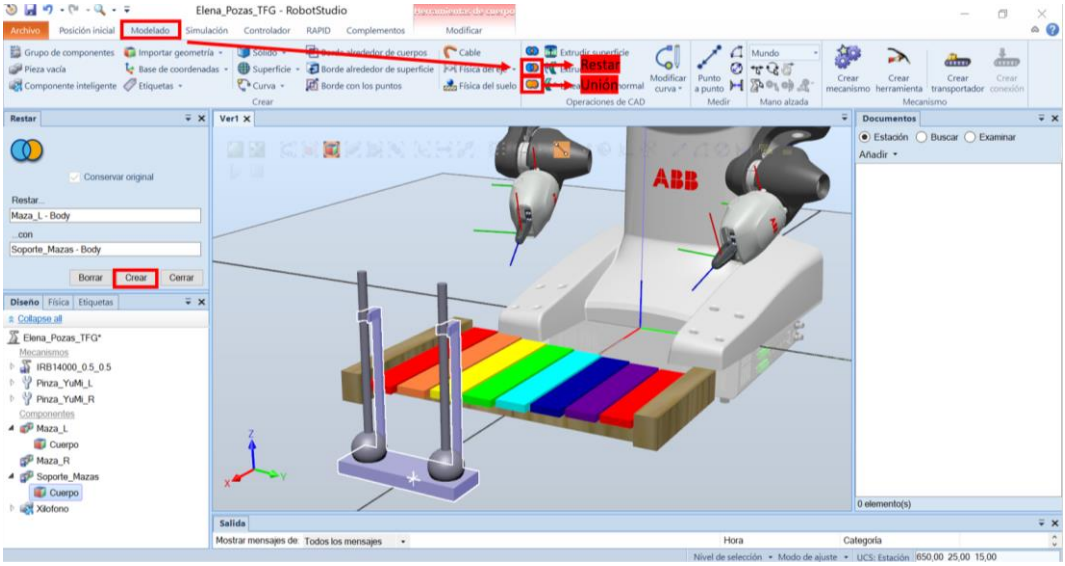


Figura 21. Operaciones de CAD.

Además, se ha incluido una botonera para poder visualizar las entradas y salidas que se activan en el controlador, la cual fue diseñada por el alumno Juan Antonio Ávila Herrero para su trabajo de fin de grado de la escuela, titulado “Modelado de una célula robótica con fines educativos usando el programa Robot-Studio”.

4.1.2. Componentes inteligentes

El siguiente paso que se va a seguir para diseñar los componentes de la estación es llevar a cabo la creación de los componentes inteligentes [20]. Esta estación se compone de cuatro componentes inteligentes, las dos herramientas, el xilófono y la botonera. Se puede crear un componente inteligente desde la pestaña “Modelado” (Figura 22). A continuación, se analiza cada uno de los componentes creados.

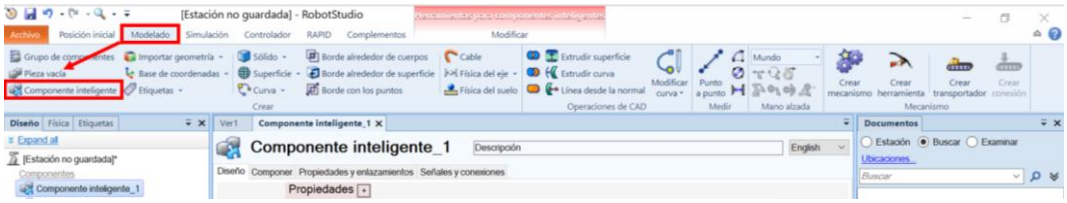


Figura 22. Crear Componente inteligente.

Pinza YuMi

Este componente se ha creado de tal forma que sea capaz de coger cualquier pieza, no solo las baquetas del xilófono. Para ello se han añadido tres sensores, dos sensores planos en la superficie interna de los dedos, y un sensor lineal en el centro de la herramienta (Figura 23). Los sensores planos tienen por objeto detectar hasta dónde tiene que cerrar la pinza el robot. Por su parte, el sensor lineal se emplea para detectar hasta dónde tiene que descender el robot para coger el objeto. Este mecanismo es análogo a la herramienta que se utiliza en otros trabajos para el robot IRB 120.

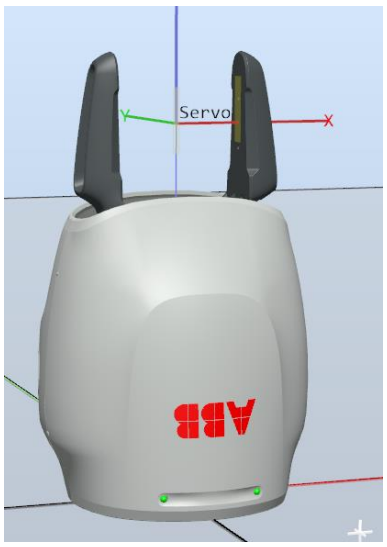


Figura 23. Componente inteligente "Smart Gripper Servo Fingers".

Los componentes de los que está formada la herramienta del robot son los siguientes:

- **Logic Gate [AND]** (Figura 24)**:** la señal Output es activada por la operación lógica especificada, en este caso AND, en las dos señales InputA y InputB, con el retardo especificado en Delay. Para que se active la salida, ambos sensores planos deben tener valor 1.

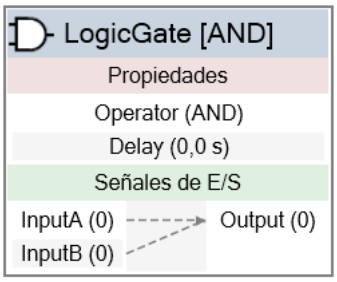


Figura 24. Componente LogicGate [AND] Pinza YuMi.

- **Logic SRLatch** (Figura 25)**:** tiene un estado estable. Si la señal Set vale 1, se activa la señal de salida, mientras que la señal de salida inversa InvOutput vale 0. Si la señal Reset vale 1, se desactiva la señal de salida, mientras que la señal de salida inversa InvOutput vale 1.
  La señal Set será activada cuando los dos sensores planos valgan 1, y la señal Reset cuando se solicite abrir la pinza. Si la salida vale 1, la salida de la pinza "PiezaAgarrada" también valdrá 1.

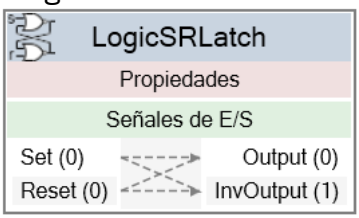


Figura 25. Componente LogicSRLatch Pinza YuMi.

- **Line Sensor** (Figura 26)**:** define una línea por sus parámetros Start, End y Radius. El parámetro Start especifica el punto de inicio. El parámetro End especifica el punto final. Finalmente, el parámetro Radius especifica el radio. Cuando una señal Active tiene el valor alto, el sensor detecta los objetos que están en intersección con la línea. Los objetos que están en intersección se muestran en la propiedad SensedPart y el punto de la pieza en intersección más cercana al punto inicial del sensor de línea se muestra en la propiedad SensedPoint. Cuando se produce la intersección, se activa la señal de salida SensorOut.
  Se ha colocado un sensor lineal en el centro de la herramienta para detectar hasta dónde tiene que descender el robot para coger el objeto.

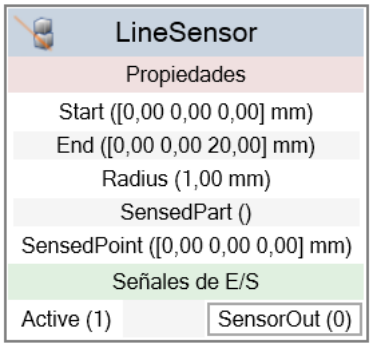


Figura 26. Componente LineSensor Pinza YuMi.

- **Plane Sensor** (Figura 27)**:** define un plano mediante los parámetros Origin, Axis1 y Axis2. El parámetro Origin especifica el origen del plano, y los parámetros Axis1 y Axis2 especifican los ejes del primer y segundo

plano, respectivamente. Cuando la señal de entrada Active está activada, el sensor detecta los objetos que presentan intersección con este plano. Los objetos en intersección se muestran en la propiedad SensedPart y cuando se produce la intersección, se activa la señal de salida SensorOut.
Se han colocado dos sensores planos sobre la superficie interna de los dedos de la pinza para detectar cuando se llega a tocar el objeto que se desea coger por ambos lados, y así saber en qué momento debe dejar de cerrarse.

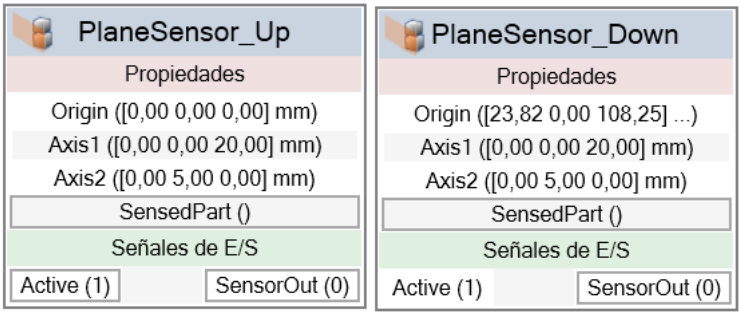


Figura 27. Componentes PlaneSensor Pinza YuMi.

- **Attacher** (Figura 28)**:** conectará el objeto Child a Parent cuando se activa la señal Execute. Si Parent es un mecanismo, como en este caso, también es necesario especificar la brida Flange a la que conectarse. Si la entrada Execute está activada, el objeto subordinado se conecta al objeto superior. Si Mount está activado, el objeto subordinado también se montará sobre el objeto superior, con los parámetros Offset y Orientation especificados. La salida Executed se activa al finalizar.
  Se utiliza para que el objeto detectado por la pinza se mueva junto con ella.

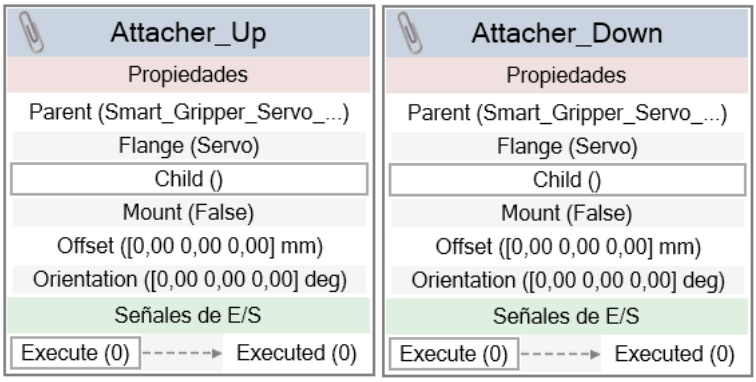


Figura 28. Componentes Attacher Pinza YuMi.

- **Detacher** (Figura 29)**:** desconectará el objeto Child del objeto cuando se activa la señal Execute. Si Keep position está activado, la posición se mantendrá. De lo contrario, el objeto subordinado se posiciona con respecto a su objeto superior. Al finalizar, la señal Executed se activa. Se utiliza para que el objeto detectado por la pinza deje de moverse junto con ella, y así poder dejar el objeto en la posición deseada.

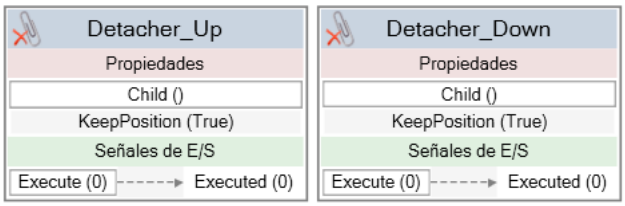


Figura 29. Componentes Detacher Pinza YuMi.

- **Joint Mover** (Figura 30)**:** utiliza como propiedades un mecanismo, un conjunto de valores de eje y una duración. Cuando se activa la señal de entrada Execute, los valores de eje del mecanismo se mueven hasta la posición indicada. Una vez alcanzada la posición, se activa la señal de salida Executed. La señal GetCurrent obtiene los valores de eje actuales del mecanismo. El primer componente JointMover servirá para cerrar la pinza, con una apertura mínima de 1 milímetro, y el segundo componente JointMover_2 servirá para abrir la pinza, con una apertura de los dedos de la herramienta de 20 milímetros.

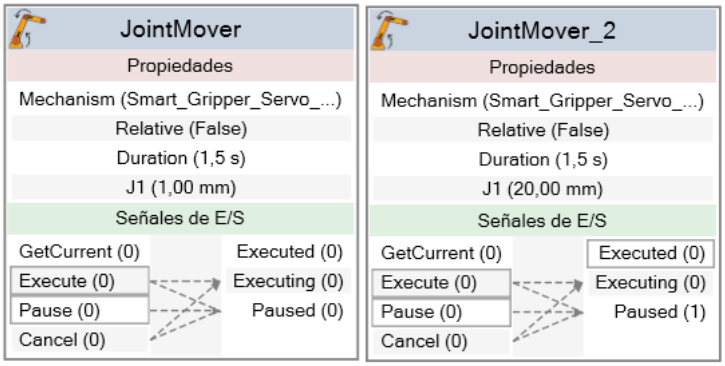


Figura 30. Componentes JointMover Pinza YuMi.

Una vez definidos todos estos componentes, se añaden dos entradas y dos salidas al componente (Tabla 6), y se unen todas las señales de forma correcta para que la herramienta funcione correctamente (Tabla 7 y Tabla 8).

Nombre	Tipo de señal
Cerrar	DigitalInput
Abrir	DigitalInput
PiezaDetectada	DigitalOutput
PiezaAgarrada	DigitalOutput

Tabla 6. Señales de E/S Pinza YuMi.

Objeto de Origen	Propiedad de origen	Objeto de destino	Propiedad de destino
PlaneSensor_Down	SensedPart	**Attacher_Down**	Child
Attacher_Down	Child	**Detacher_Down**	Child
PlaneSensor_Up	SensedPart	**Attacher_Up**	Child
Attacher_Up	Child	**Detacher_Up**	Child

Tabla 7. Enlazamientos de propiedad Pinza YuMi.

Objeto de origen	Señal de origen	Objeto de destino	Señal o propiedad de destino
LineSensor	SensorOut	**Pinza_YuMi**	PiezaDetectada

Pinza_YuMi	Cerrar	PlaneSensor_Up	Active
Pinza_YuMi	Cerrar	PlaneSensor_Down	Active
Pinza_YuMi	Abrir	LogicSRLatch	Reset
Pinza_YuMi	Abrir	Detacher_Up	Execute
PlaneSensor_Down	SensorOut	LogicGate [AND]	InputB
PLaneSensor_Up	SensorOut	LogicGate [AND]	InputA
LogicGate [AND]	Output	LogicSRLatch	Set
Pinza_YuMi	Abrir	Detacher_Down	Execute
PlaneSensor_Up	SensorOut	Attacher_Up	Execute
PlaneSensor_Down	SensorOut	Attacher_Down	Execute
LogicSRLatch	Output	Pinza_YuMi	PiezaAgarrada
Pinza_YuMi	Cerrar	JointMover	Execute
LogicGate [AND]	Output	JointMover	Pause
Pinza_YuMi	Abrir	JointMover_2	Execute
JointMover_2	Executed	JointMover_2	Pause

Tabla 8. Conexiones de E/S Pinza YuMi.

En la Figura 31 se puede ver el resultado final del diseño de la pinza. Se muestran todas sus entradas y salidas, así como las conexiones de todos los componentes.

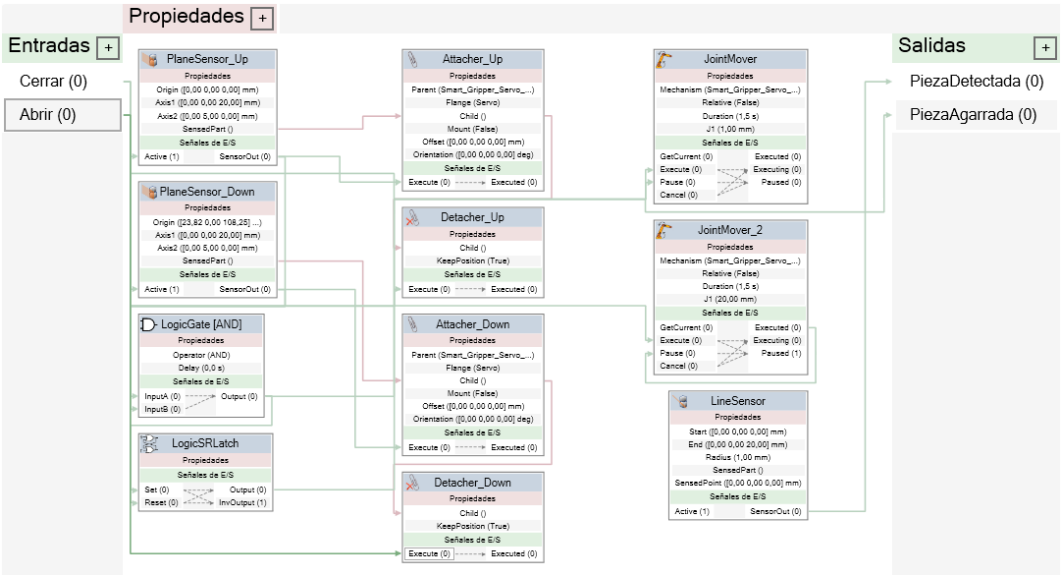


Figura 31. Diseño Componente inteligente Pinza YuMi.

Xilófono

El xilófono también se ha convertido en un componente inteligente (Figura 32) para que el robot pueda detectar tanto la tecla que ha golpeado, como la altura a la que se encuentra esa tecla. En el momento que llega a golpearla, el robot comenzará a subir. Para esto, se han colocado sensores planos en cada una de las láminas del xilófono, y un componente denominado Highlighter que permite ver de una manera más visual el momento en el que el robot llega a

golpear la tecla cambiando temporalmente a blanco el color de la tecla golpeada.

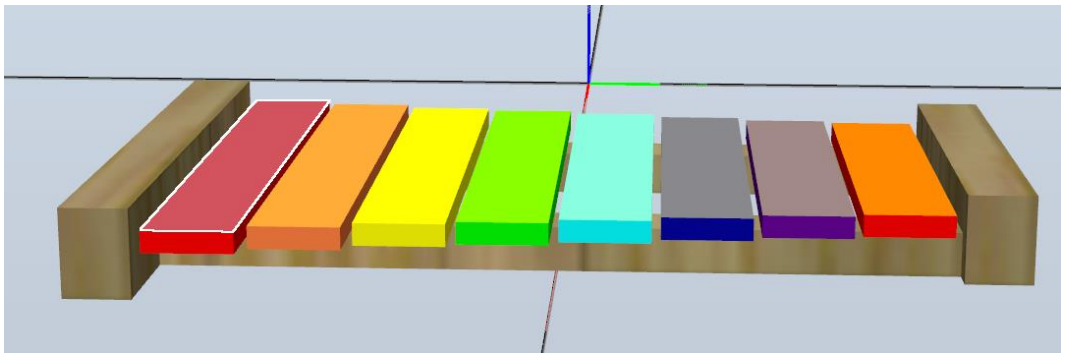


Figura 32. Componente inteligente Xilófono.

Los elementos de los que se compone el componente inteligente del xilófono son los siguientes:

- **Plane Sensor** (Figura 33)**:** define un plano mediante los parámetros Origin, Axis1 y Axis2. El parámetro Origin especifica el origen del plano, y los parámetros Axis1 y Axis2 especifican los ejes del primer y segundo plano, respectivamente. Cuando la señal de entrada Active está activada, el sensor detecta los objetos que presentan intersección con este plano. Los objetos en intersección se muestran en la propiedad SensedPart y cuando se produce la intersección, se activa la señal de salida SensorOut.

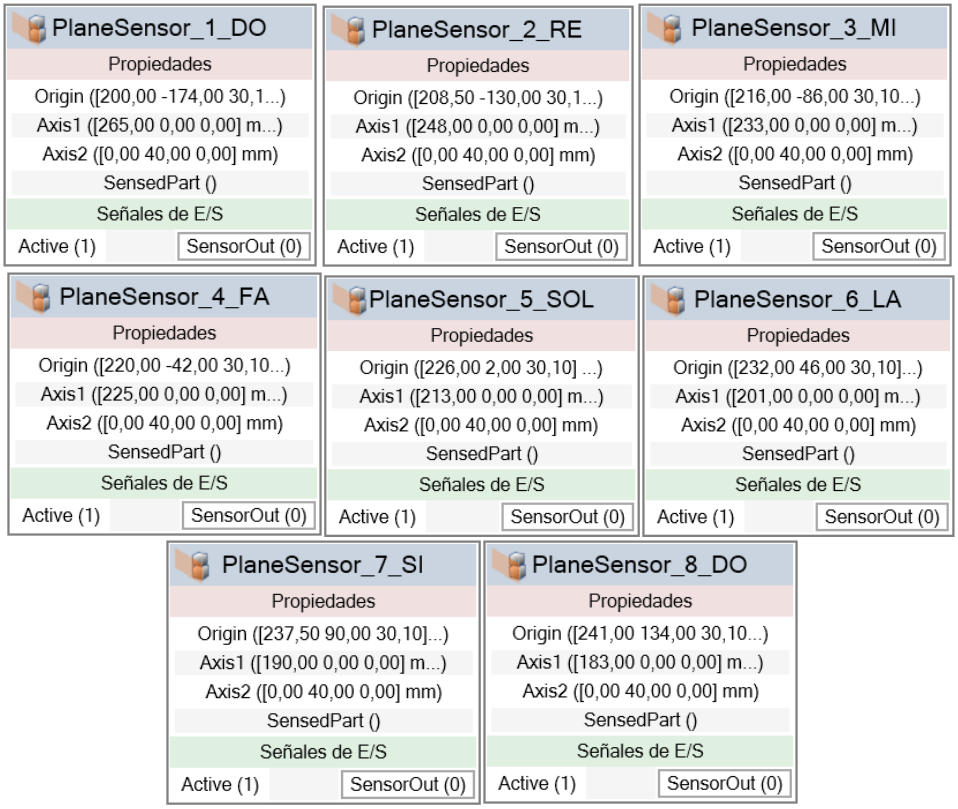


Figura 33. Componentes Plane Sensor Xilófono.

- **Hightlighter** (Figura 34): cambia temporalmente el color de Object a los valores RGB especificados en Color (el [255 255 255] es el valor del color blanco en RGB). El color se mezcla con el color original de los objetos tal y como se definen por Opacity. En este caso no se utililiza esta opción, ya que Opacity tiene valor 255. Cuando la señal Active se desactiva, Object recibe sus colores originales.

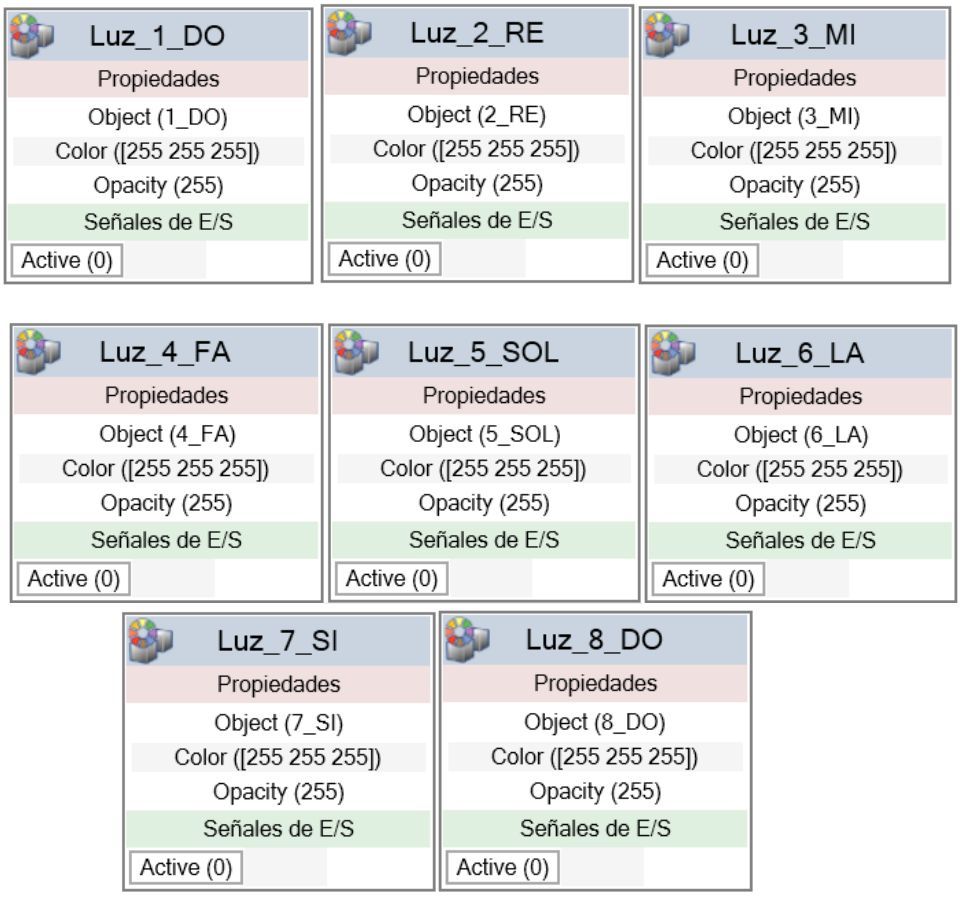


Figura 34. Componentes Highlighter Xilófono.

Este componente cuenta con ocho salidas y ninguna entrada (Tabla 9). Cuando se active un sensor se activará también el cambio de color del objeto asociado y se mandará la señal a las salidas del xilófono (Tabla 10).

Nombre	Tipo de señal
DO_0	DigitalOutput
DO_1	DigitalOutput
DO_2	DigitalOutput
DO_3	DigitalOutput
DO_4	DigitalOutput
DO_5	DigitalOutput
DO_6	DigitalOutput
DO_7	DigitalOutput

Tabla 9. Señales de E/S Xilófono.

Objeto de origen	Señal de origen	Objeto de destino	Señal o propiedad de destino
PlaneSensor_1_DO	SensorOut	Xilofono	DO_0
PlaneSensor_2_RE	SensorOut	Xilofono	DO_1
PlaneSensor_3_MI	SensorOut	Xilofono	DO_2
PlaneSensor_4_FA	SensorOut	Xilofono	DO_3
PlaneSensor_5_SOL	SensorOut	Xilofono	DO_4
PlaneSensor_6_LA	SensorOut	Xilofono	DO_5
PlaneSensor_7_SI	SensorOut	Xilofono	DO_6
PlaneSensor_8_DO	SensorOut	Xilofono	DO_7
PlaneSensor_1_DO	SensorOut	Luz_1_DO	Active
PlaneSensor_2_RE	SensorOut	Luz_2_RE	Active
PlaneSensor_3_MI	SensorOut	Luz_3_MI	Active
PlaneSensor_4_FA	SensorOut	Luz_4_FA	Active
PlaneSensor_5_SOL	SensorOut	Luz_5_SOL	Active
PlaneSensor_6_LA	SensorOut	Luz_6_LA	Active
PlaneSensor_7_SI	SensorOut	Luz_7_SI	Active
PlaneSensor_8_DO	SensorOut	Luz_8_DO	Active

Tabla 10. Conexiones de E/S Xilófono.

En la Figura 35 se muestra el resultado final del xilófono, con todas sus salidas y las conexiones de todos los componentes.

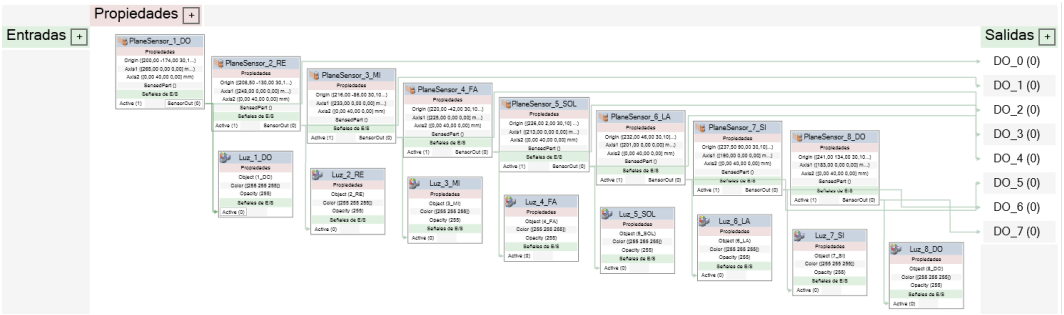


Figura 35. Diseño Componente inteligente Xilófono.

Botonera

La botonera sirve para activar alguna entrada del controlador, o para visualizar en todo momento el estado de las diferentes señales. Para ello, contaremos con dieciséis botones que activarán las entradas del controlador, y dieciséis "bombillas" que cambiarán a color rojo cada vez que se active una salida, creando el efecto de que están encendidas (Figura 36). Las dieciséis entradas y salidas corresponden a las dieciséis entradas y salidas de las que cuenta el controlador. Cada uno de los botones que activarán las entradas también son componentes inteligentes que pertenecen a la botonera.

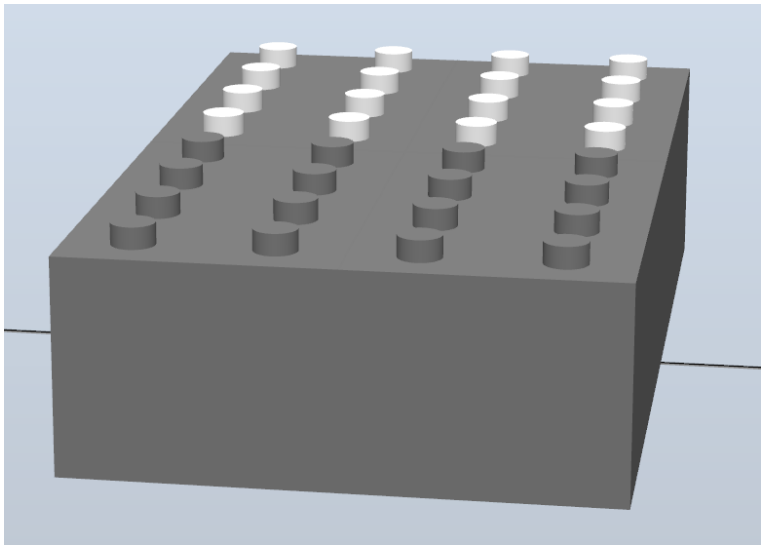


Figura 36. Componente inteligente Botonera.

Los elementos de los que se compone la botonera son los siguientes:

- **Hightlighter** (Figura 37): cambia temporalmente el color de Object a los valores RGB especificados en Color (el [255 0 0] corresponde al color rojo en RGB). El color se mezcla con el color original de los objetos tal y como se definen por Opacity. En este caso no se utililiza esta opción, ya que Opacity tiene valor 255. Cuando la señal Active se desactiva, Object recibe sus colores originales. Habrá uno igual por cada salida del controlador.

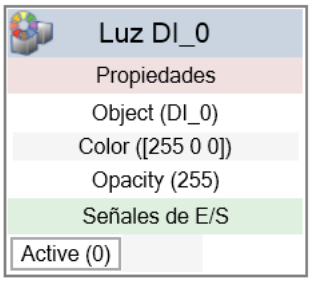


Figura 37. Componente Highlighter Botonera.

- **Componente inteligente botón** (Figura 38). Habrá uno igual por cada entrada del controlador.

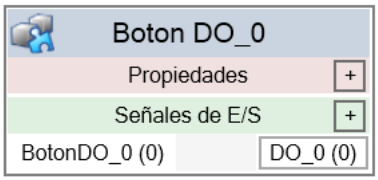


Figura 38. Componente inteligente Botón DO_0.

Se compone de los siguientes elementos:

  - **Hightlighter** (Figura 39): cambia temporalmente el color de Object a los valores RGB especificados en Color (el [255 0 0] corresponde al color rojo en RGB). El color se mezcla con el color original de los objetos tal y como se definen por Opacity. En este

caso no se utililiza esta opción, ya que Opacity tiene valor 255. Cuando la señal Active se desactiva, Object recibe sus colores originales.
Se utiliza para indicar las salidas del controlador que están activadas.

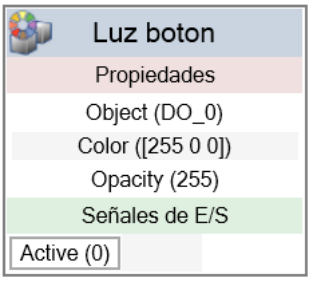


Figura 39. Componente Highlighter Botón.

- **Logic Gate [NOT]** (Figura 40)**:** la señal Output es activada por la operación lógica especificada, en este caso NOT, en la señal InputA, con el retardo especificado en Delay.
  Se utiliza para activar el bloque que reposiciona el botón en su posición inicial cuando la señal que se activa al pulsar el botón vale 0.

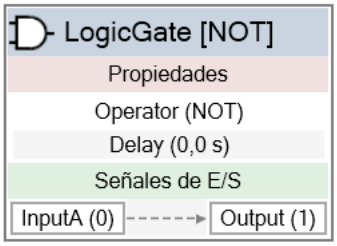


Figura 40. Componente LogicGate [NOT] Botón.

- **Positioner** (Figura 41)**:** toma un objeto, una posición y una orientación como propiedades. Cuando se activa la señal Execute, el objeto es reposicionado en la posición determinada con respecto a Reference. Al finalizar, se activa la salida Executed.
  Se utiliza para dar el mismo efecto que se produce cuando se pulsa un botón real. Al pulsar un botón, se posiciona unos milímetros por debajo de donde estaba, y al dejar de pulsarlo vuelve a su posición inicial.

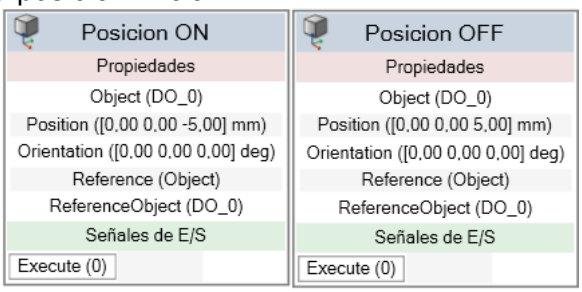


Figura 41. Componente Positioner Botón.

Este componente cuenta con quince salidas y quince entradas (Tabla 11), correspondientes con las quince salidas y las quince entradas con las que cuenta el controlador. Su funcionamiento es el siguiente:

- Cuando se active una salida del controlador, se activará también el cambio de color del objeto asociado y se mandará la señal a las salidas de la botonera.

- Cuando se active una entrada del controlador, se activará la entrada del botón, haciendo que el botón cambie de posición para simular que está pulsado (Tabla 12).

Nombre	Tipo de señal
DO_0	DigitalOutput
DI_1	DigitalInput
DI_2	DigitalInput
DI_3	DigitalInput
DI_4	DigitalInput
DI_5	DigitalInput
DI_6	DigitalInput
DI_7	DigitalInput
DI_8	DigitalInput
DI_9	DigitalInput
DI_10	DigitalInput
DI_11	DigitalInput
DI_12	DigitalInput
DI_13	DigitalInput
DI_14	DigitalInput
DI_15	DigitalInput

Nombre	Tipo de señal
DO_0	DigitalOutput
DO_1	DigitalOutput
DO_2	DigitalOutput
DO_3	DigitalOutput
DO_4	DigitalOutput
DO_5	DigitalOutput
DO_6	DigitalOutput
DO_7	DigitalOutput
DO_8	DigitalOutput
DO_9	DigitalOutput
DO_10	DigitalOutput
DO_11	DigitalOutput
DO_12	DigitalOutput
DO_13	DigitalOutput
DO_14	DigitalOutput
DO_15	DigitalOutput

Tabla 11. Señales de E/S Botonera.

Objeto de origen	Señal de origen	Objeto de destino	Señal o propiedad de destino
Botonera	DI_0	Luz DI_0	Active
Botonera	DI_1	Luz DI_1	Active
Botonera	DI_2	Luz DI_2	Active
Botonera	DI_3	Luz DI_3	Active
Botonera	DI_4	Luz DI_4	Active
Botonera	DI_5	Luz DI_5	Active
Botonera	DI_6	Luz DI_6	Active
Botonera	DI_7	Luz DI_7	Active
Botonera	DI_8	Luz DI_8	Active
Botonera	DI_9	Luz DI_9	Active
Botonera	DI_10	Luz DI_10	Active
Botonera	DI_11	Luz DI_11	Active
Botonera	DI_12	Luz DI_12	Active

Botonera	DI_13	**Luz DI_13**	Active
Botonera	DI_14	**Luz DI_14**	Active
Botonera	DI_15	**Luz DI_15**	Active
Boton DO_0	DO_0	**Botonera**	DO_0
Boton DO_1	DO_1	**Botonera**	DO_1
Boton DO_2	DO_2	**Botonera**	DO_2
Boton DO_3	DO_3	**Botonera**	DO_3
Boton DO_4	DO_4	**Botonera**	DO_4
Boton DO_5	DO_5	**Botonera**	DO_5
Boton DO_6	DO_6	**Botonera**	DO_6
Boton DO_7	DO_7	**Botonera**	DO_7
Boton DO_8	DO_8	**Botonera**	DO_8
Boton DO_9	DO_9	**Botonera**	DO_9
Boton DO_10	DO_10	**Botonera**	DO_10
Boton DO_11	DO_11	**Botonera**	DO_11
Boton DO_12	DO_12	**Botonera**	DO_12
Boton DO_13	DO_13	**Botonera**	DO_13
Boton DO_14	DO_14	**Botonera**	DO_14
Boton DO_15	DO_15	**Botonera**	DO_15

Tabla 12. Conexiones de E/S Botonera.

El componente inteligente del botón, perteneciente también a la botonera, cuenta con una salida (Tabla 13). Cuando se active una entrada del controlador, se activará también el cambio de color del objeto asociado y cambiará de posición, simulando que se ha pulsado ese botón. Se mandará la misma señal de entrada del botón a la salida del botón. (Tabla 14).

Nombre	Tipo de señal
DO_0	DigitalOutput

Tabla 13. Señales de E/S Botón DO_0.

Objeto de origen	Señal de origen	Objeto de destino	Señal o propiedad de destino
LogicGate [NOT]	Output	**Posicion OFF**	Execute
Boton DO_0	BotonDO_0	**LogicGate [NOT]**	InputA
Boton DO_0	BotonDO_0	**Luz boton**	Active
Boton DO_0	BotonDO_0	**Posicion ON**	Execute
Boton DO_0	BotonDO_0	**Boton DO_0**	DO_0

Tabla 14. Conexiones de E/S Botón DO_0.

En la Figura 42 se muestra el diseño final de la botonera, con todas sus salidas y las conexiones de todos los componentes.

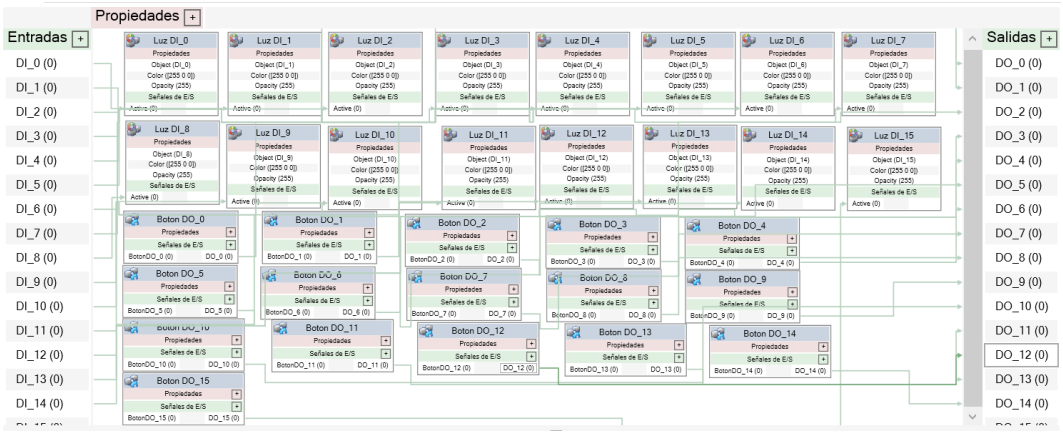


Figura 42. Diseño Componente inteligente Botonera.

En la Figura 43 se muestra el diseño de los botones, con todas sus salidas y las conexiones de todos los componentes.

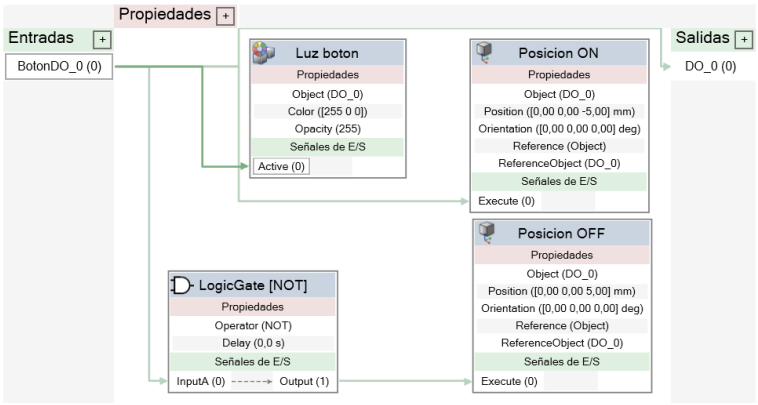


Figura 43. Diseño Componente inteligente Botón.

4.1.3. Configuración del controlador

Para crear un controlador, en la pestaña “Posición inicial” encontramos la opción de Controlador virtual, la cual nos proporciona tres posibilidades:

- Crear un controlador virtual desde diseño.
- Crear un nuevo controlador.
- Añadir un controlador ya existente (Figura 44).

Para este caso, se crea un controlador virtual en función de un diseño existente, al cual se le ha denominado IRB14000.

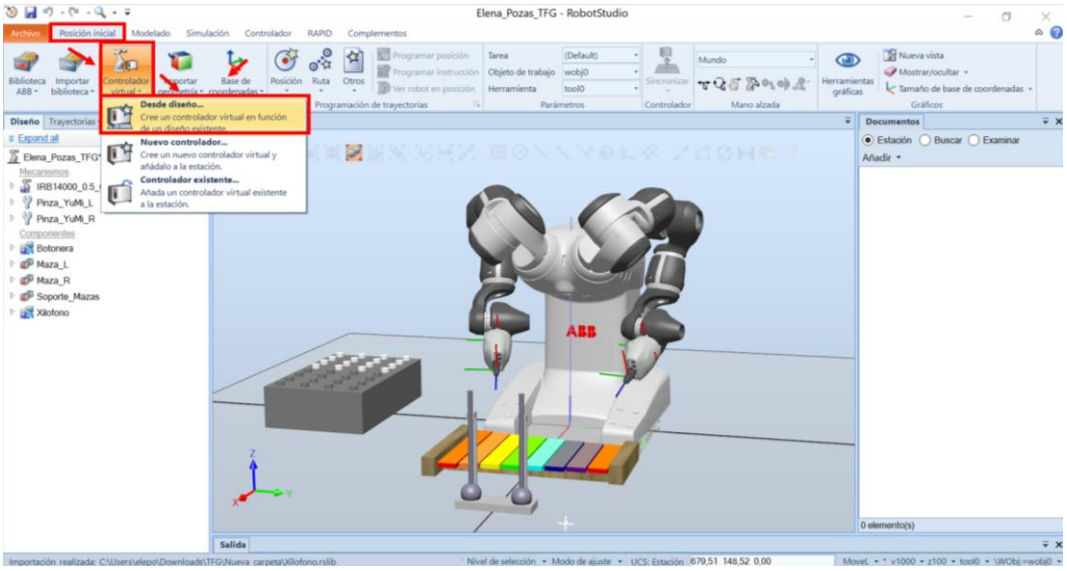


Figura 44. Crear un controlador virtual.

Se dejan todas las opciones del controlador que vienen por defecto, cambiando únicamente el lenguaje del controlador de inglés a español (Figura 45).

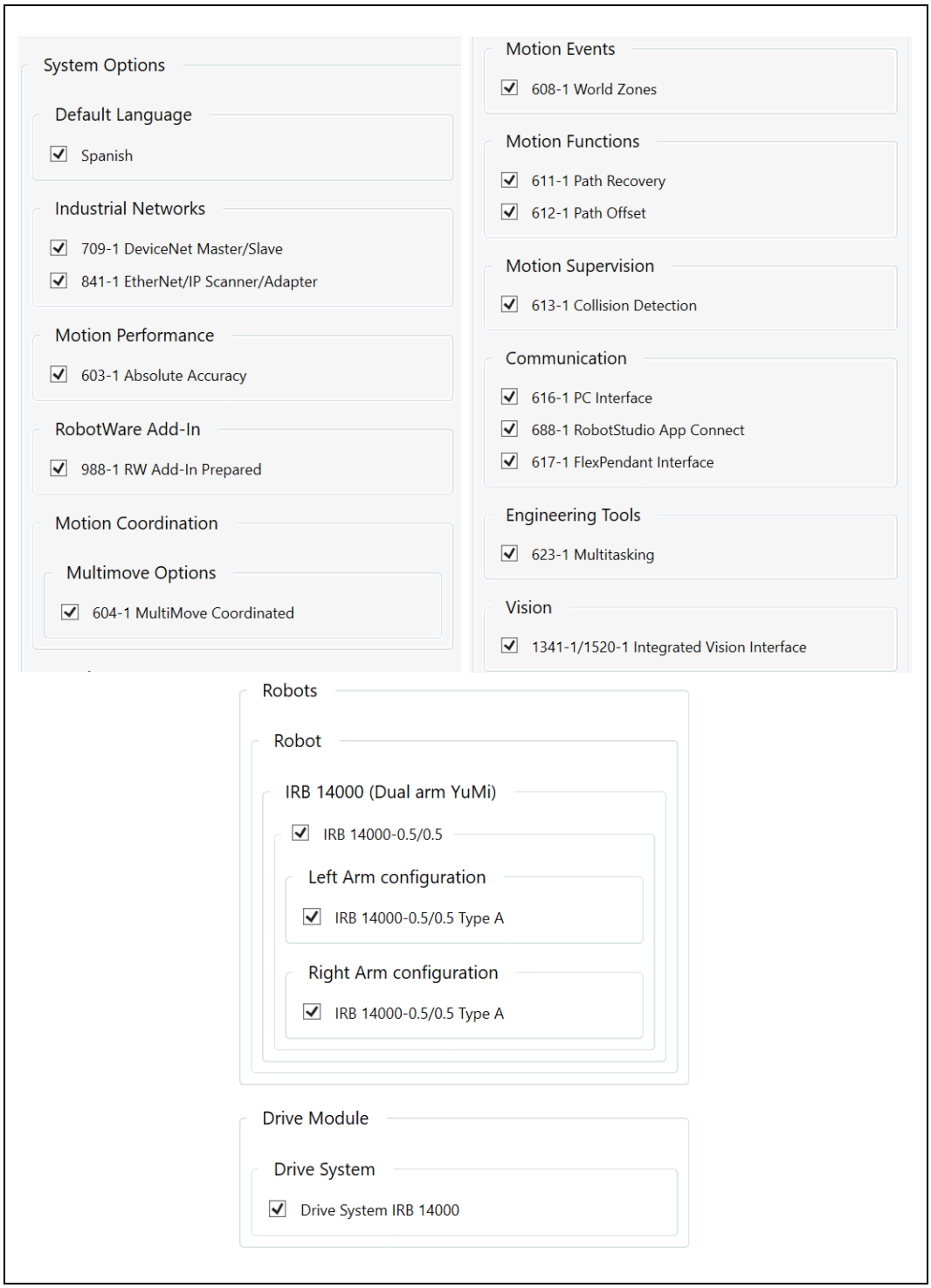


Figura 45. Opciones controlador IRB14000.

Este controlador cuenta con dos tareas independientes entre sí, para poder controlar cada uno de los brazos del robot. Cada tarea tendrá sus propias variables independientes unas de otras.

El siguiente paso es modificar las entradas y las salidas dadas por defecto. El controlador cuenta con ocho entradas y ocho salidas. Se necesita una salida por tecla del controlador, por lo que se va a aumentar el número de entradas y salidas a dieciséis. Para ello, en la pestaña de “Controlador”, en “Configuración”, se selecciona “I/O System” para abrir el editor de configuración y modificar las entradas y salidas del controlador (Figura 46).

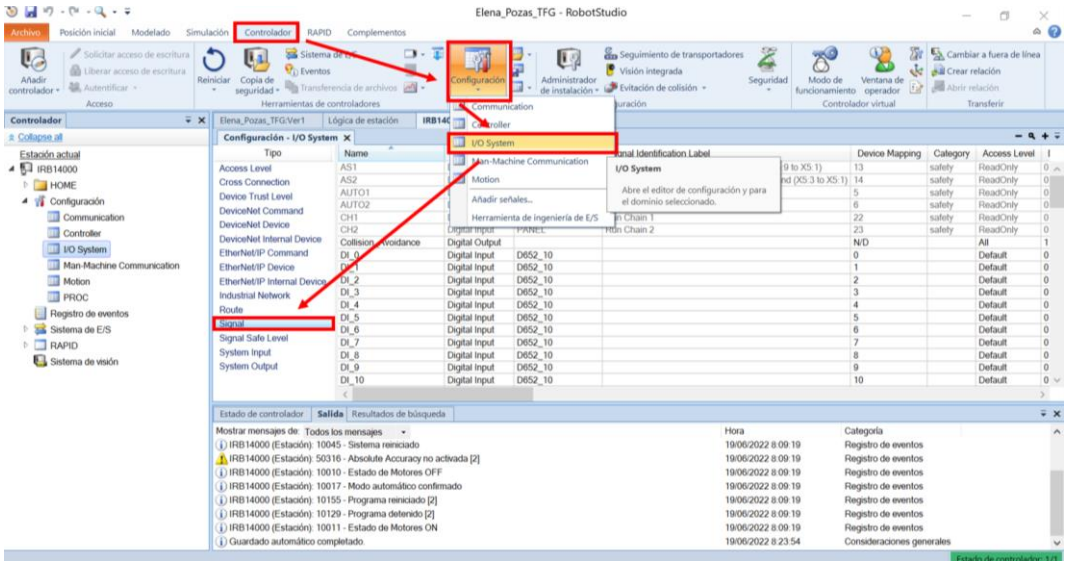


Figura 46. Editor de configuración de señales del controlador.

A la hora de modificar las señales y las salidas, hay que tener en cuenta el valor que tiene Device Mapping en cada señal. No se puede asignar el mismo valor de Device Mapping a dos señales del mismo tipo a la vez. Si hay una entrada que ya tenga un valor asignado, ese valor solo se puede volver a utilizar para una salida, pero no para otra entrada. Por este motivo se han modificado estos valores que venían por defecto, y se han reservado el rango del 0 al 15 para las entradas y salidas que se van a usar (Figura 47).

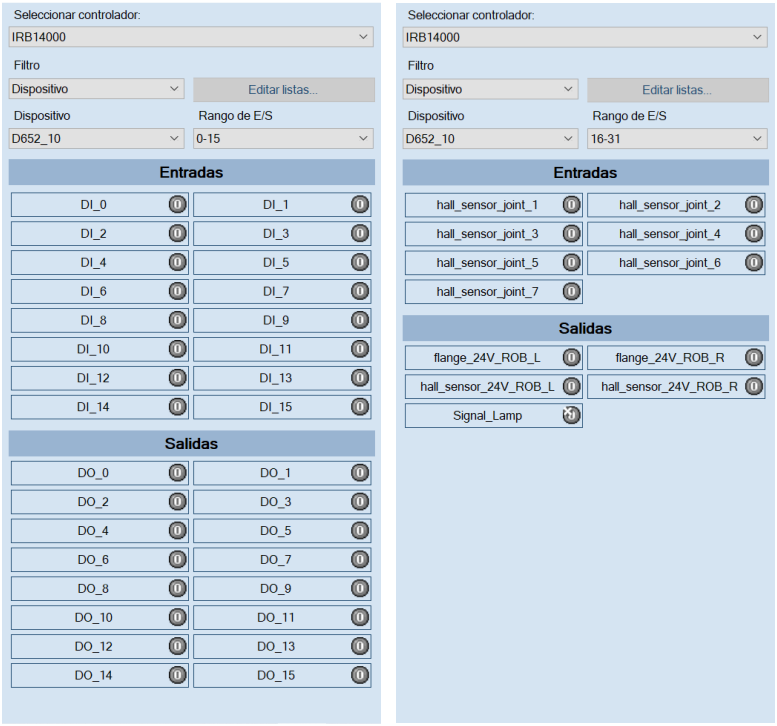


Figura 47. Simulador de E/S IRB14000.

4.1.4. Lógica de la estación

Dentro de la pestaña “Simulación”, en “Lógica de estación” (Figura 48), se puede diseñar la lógica de la simulación conectando los componentes inteligentes.

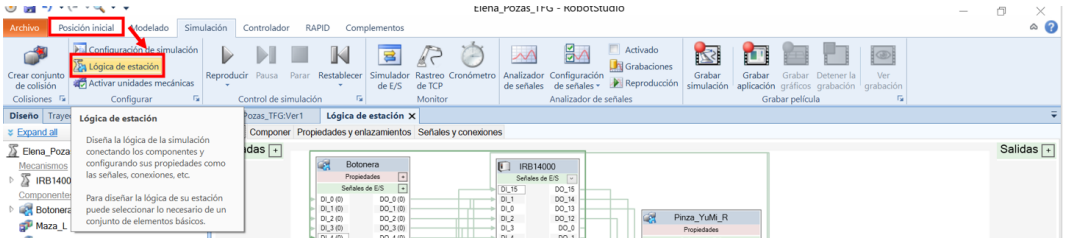


Figura 48. Diseñar la lógica de la simulación.

En la lógica de estación se unen las salidas de la botonera con las entradas del IRB 14000, y las salidas del IRB 14000 con las entradas de la botonera, para que cuando se active alguna salida, se vea el cambio de color de las “luces” de la botonera.

Las salidas del xilófono se unen con las entradas del controlador, ya que se necesita esta señal para saber en qué momento llega a golpear una tecla el robot.

Las salidas DO_15 y DO_14 del IRB 14000 se usarán para activar las entradas Cerrar y Abrir de la pinza YuMi del brazo derecho. Para la pinza YuMi del brazo izquierdo se usarán las salidas DO_13 y DO_12.

Todo esto se puede ver en la Figura 49 y en la Tabla 15.

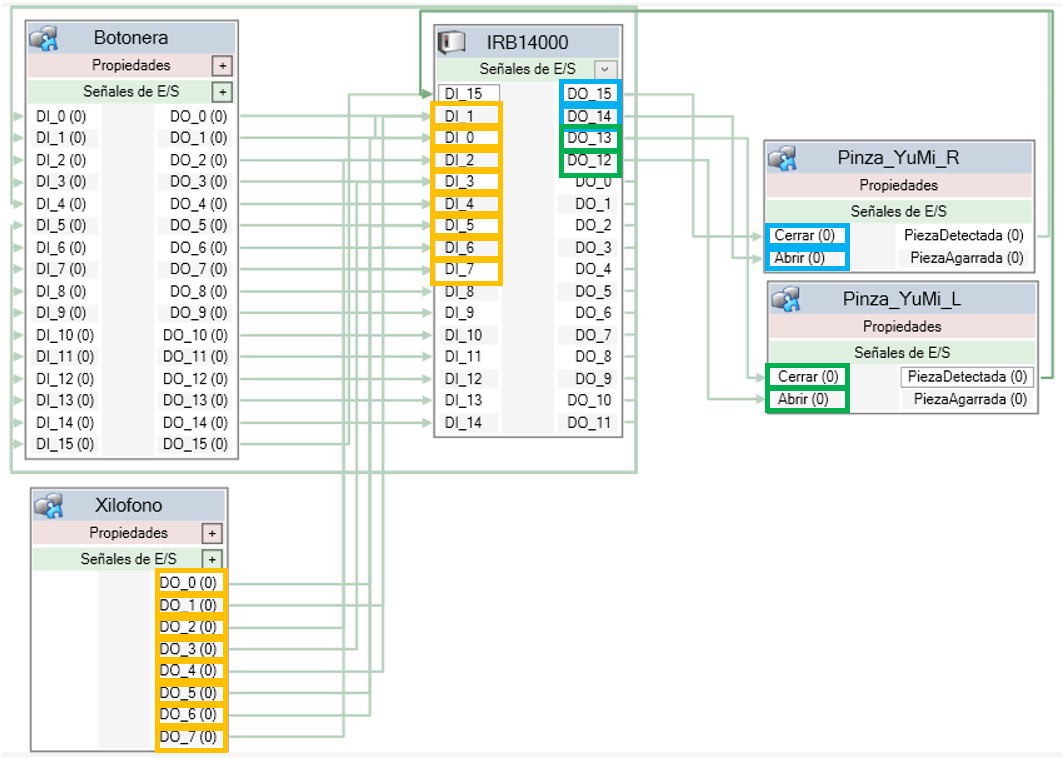


Figura 49. Lógica de estación.

Objeto de origen	Señal de origen	Objeto de destino	Señal o propiedad de destino
IRB14000	DO_15	Pinza_YuMi_R	Cerrar
IRB14000	DO_14	Pinza_YuMi_R	Abrir
Pinza_YuMi_L	PiezaDetectada	IRB14000	DI_15
Pinza_YuMi_R	PiezaDetectada	IRB14000	DI_15
Xilofono	DO_1	IRB14000	DI_1
Xilofono	DO_0	IRB14000	DI_0
Xilofono	DO_2	IRB14000	DI_2
Xilofono	DO_3	IRB14000	DI_3
Xilofono	DO_4	IRB14000	DI_4
Xilofono	DO_5	IRB14000	DI_5
Xilofono	DO_6	IRB14000	DI_6
Xilofono	DO_7	IRB14000	DI_7
IRB14000	DO_13	Pinza_YuMi_L	Cerrar
IRB14000	DO_12	Pinza_YuMi_L	Abrir
IRB14000	DO_0	Botonera	DI_0
IRB14000	DO_1	Botonera	DI_1
IRB14000	DO_2	Botonera	DI_2

IRB14000	D0_3	Botonera	DI_3
IRB14000	D0_4	Botonera	DI_4
IRB14000	D0_5	Botonera	DI_5
IRB14000	D0_6	Botonera	DI_6
IRB14000	D0_7	Botonera	DI_7
IRB14000	D0_8	Botonera	DI_8
IRB14000	D0_9	Botonera	DI_9
IRB14000	D0_10	Botonera	DI_10
IRB14000	D0_11	Botonera	DI_11
IRB14000	D0_12	Botonera	DI_12
IRB14000	D0_13	Botonera	DI_13
IRB14000	D0_14	Botonera	DI_14
IRB14000	D0_15	Botonera	DI_15
Botonera	D0_0	IRB14000	DI_0
Botonera	D0_1	IRB14000	DI_1
Botonera	D0_2	IRB14000	DI_2
Botonera	D0_3	IRB14000	DI_3
Botonera	D0_4	IRB14000	DI_4
Botonera	D0_5	IRB14000	DI_5
Botonera	D0_6	IRB14000	DI_6
Botonera	D0_7	IRB14000	DI_7
Botonera	D0_8	IRB14000	DI_8
Botonera	D0_9	IRB14000	DI_9
Botonera	D0_10	IRB14000	DI_10
Botonera	D0_11	IRB14000	DI_11
Botonera	D0_12	IRB14000	DI_12
Botonera	D0_13	IRB14000	DI_13
Botonera	D0_14	IRB14000	DI_14
Botonera	D0_15	IRB14000	DI_15

Tabla 15. Conexiones de E/S Controlador IRB14000

4.1.5. Trayectorias y puntos

En este apartado se describen todos los puntos y objetos de trabajo que se han creado en la estación, necesarios para definir las instrucciones de movimiento del robot.

Al haber dos tareas distintas, independientes entre sí, existen puntos y objetos de trabajo únicos para cada brazo.

Jointtarget

Se utiliza para definir la posición a la que se moverán los ejes del robot y los ejes externos al ejecutar una instrucción MoveAbsJ, la cual mueve el robot a

una posición de ejes absoluta. Define las posiciones individuales de los distintos ejes, tanto de los del robot como de los externos [21].

En la Figura 50 se muestran los jointtarget definidos en la estación.

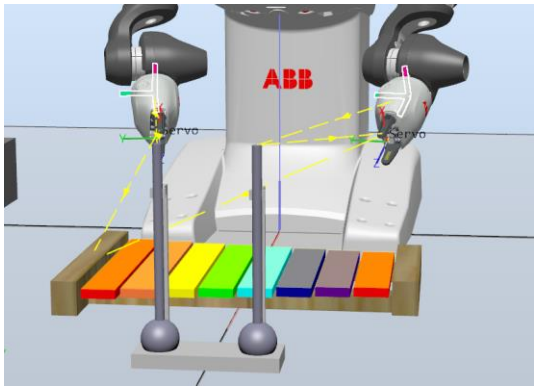


Figura 50. Jointtarget de la estación.

Para crear un jointtarget, en la pestaña “Posición inicial”, se abre la pestaña “Posición” y se selecciona “Crear posición de ejes”. Si se quiere definir la posición en la que está posicionado el robot, se pueden obtener los valores de los ejes a través de la FlexPendant del controlador. En la opción de Movimiento del menú aparecen los valores de la posición actual de los siete ejes del robot, con los que se definen los valores de ejes del jointtarget (Figura 51).

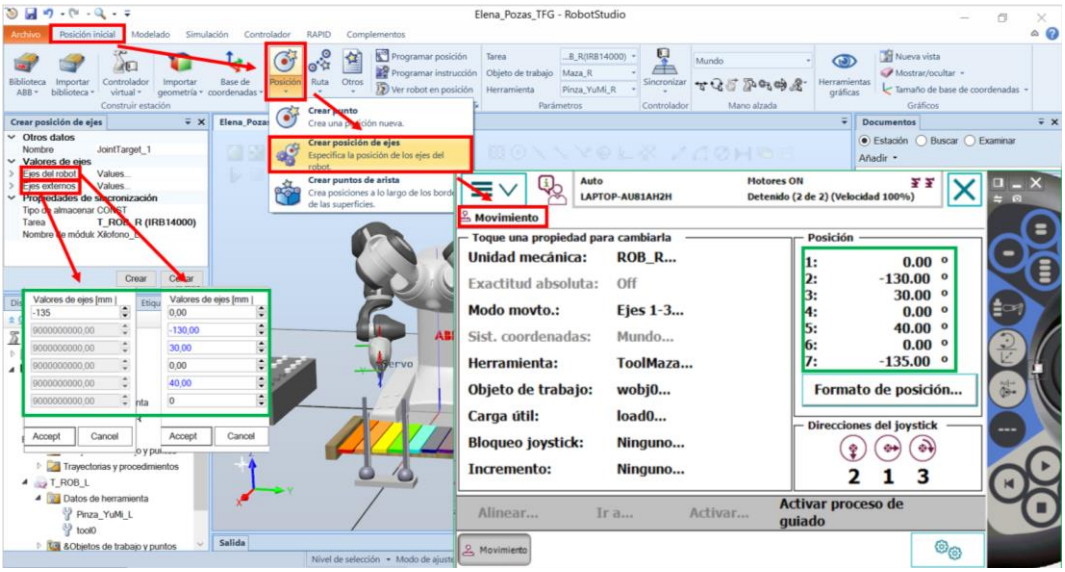


Figura 51. Crear Jointtarget.

Tooldata

Se utiliza para describir las características de una herramienta, por ejemplo, una pistola de soldadura o una pinza. Si la herramienta está fija en el espacio

(una herramienta estacionaria), los datos de la herramienta definen tanto la herramienta como la pinza que sostiene el objeto de trabajo.

Los datos de la herramienta tool0 que vienen por defecto corresponden al último eje del robot, justo en la muñeca, sin ninguna herramienta incorporada. Para poder posicionar correctamente el tooldata en el centro de los dedos de la herramienta Smart Gripper, se aumenta el valor de posición del eje z a 114,2 milímetros y se crea un dato de herramienta para cada brazo, denominados “Pinza_YuMi_R” y “Pinza_YuMi_L”. Este dato se usará para coger y dejar las mazas.

Para golpear el xilófono se necesita un nuevo dato de herramienta que incorpore tanto la herramienta Smart Gripper como las mazas. Para ello se le da un valor al eje z de 314,2, lo que correspondería con el centro de la cabeza de las mazas.

Estos datos de herramienta se pueden ver definidos en la estación en la Figura 52.

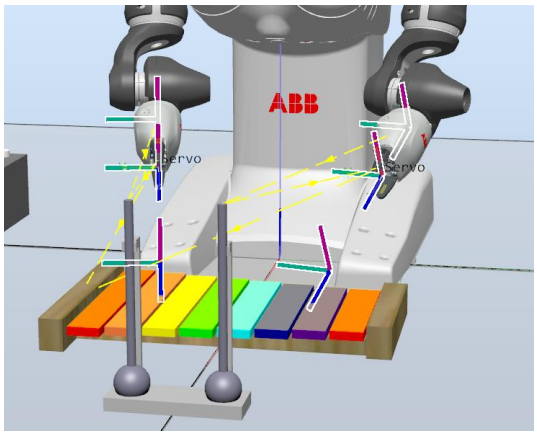


Figura 52. Tooldata de la estación.

En la pestaña “Posición inicial”, dentro de “Trayectorias y puntos”, se crea un nuevo dato de herramienta. Se pueden definir diferentes propiedades de la herramienta, como su posición y su carga. Para poder modificar los datos de herramienta, se selecciona “Objeto de trabajo” (Figura 53).

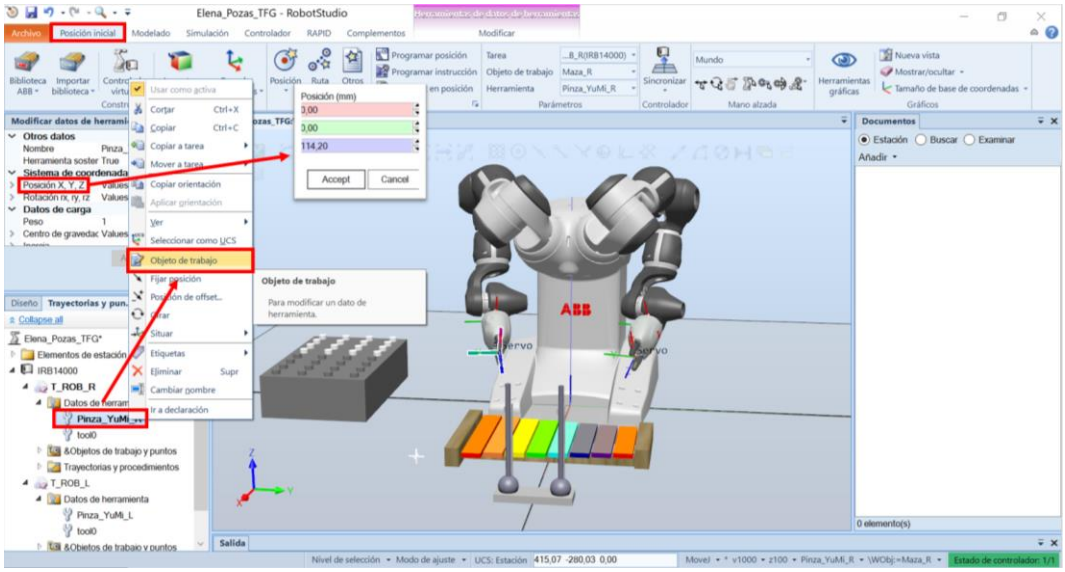


Figura 53. Crear Tooldata.

Wobjdata

Se utiliza para describir el objeto de trabajo que el robot está soldando, procesando, moviendo por sí solo, etc.

Si los objetos de trabajo están definidos en una instrucción de posicionamiento, la posición se basará en las coordenadas del objeto de trabajo.

Para esta estación se necesitan tres objetos de trabajo, uno para coger y dejar las mazas, otro para golpear las láminas del xilófono, y otro para realizar movimientos “base” del robot y llevarlo a una posición cercana a la posición de reposo. Estos objetos de trabajo se pueden ver en la Figura 54.

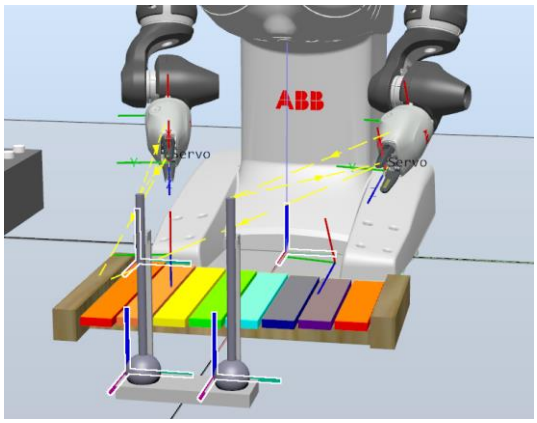


Figura 54. Wobjdata de la estación.

Los objetos de trabajo se crean en la pestaña “Posición inicial”, dentro de “Trayectorias y puntos”, se crea un nuevo objeto de trabajo. El sistema de coordenadas se puede definir seleccionando tres puntos pertenecientes al

objeto de trabajo deseado, dos puntos para el eje x, cuyo orden definirá el sentido del eje z, y un punto para el eje y (Figura 55).

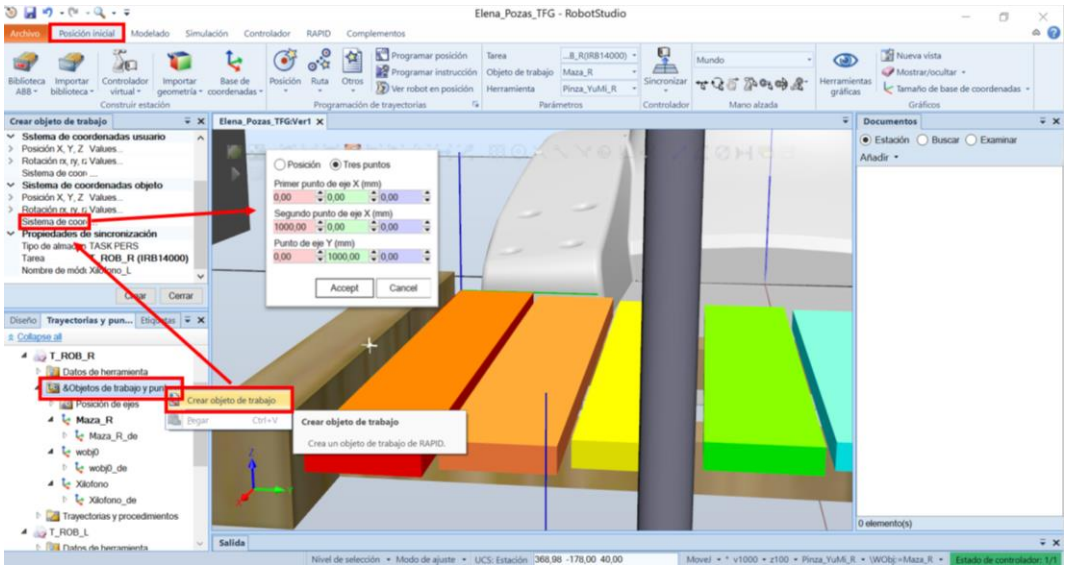


Figura 55. Crear Wobjdata.

Robtarget

Se utiliza para definir la posición del robot y de los ejes externos.

Los datos de posición se utilizan en las instrucciones de movimiento para indicar la posición hacia la que deben desplazarse los ejes del robot y los ejes externos.

Cada uno de los robtarget pertenece a un objeto de trabajo concreto, definido un punto por cada objeto de trabajo:

- uno para las mazas, en la parte superior de la varilla,
- otro genérico para todas las láminas del xilófono,
- y otro cercano a la posición de reposo del robot.

Estos puntos se pueden ver definidos en la estación en la Figura 56.

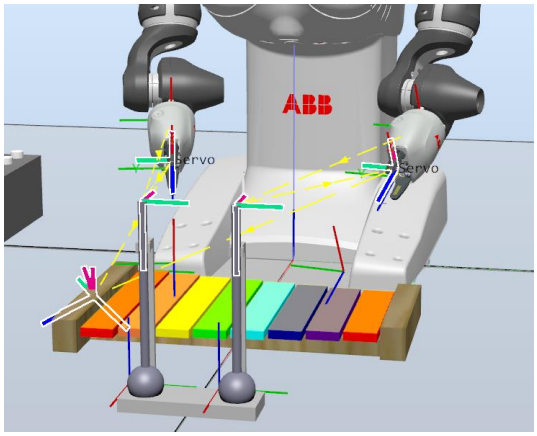


Figura 56. Robtarget de la estación.

En la pestaña “Posición inicial”, dentro de “Trayectorias y puntos”, se selecciona el objeto de trabajo al que se quiere que pertenezca el nuevo punto, y se abre el navegador “Crear punto” (Figura 57). Se define tanto la posición como la orientación deseada. Para que el robot tenga alcance a un punto, el eje z deberá estar en sentido contrario al eje z de la estación.

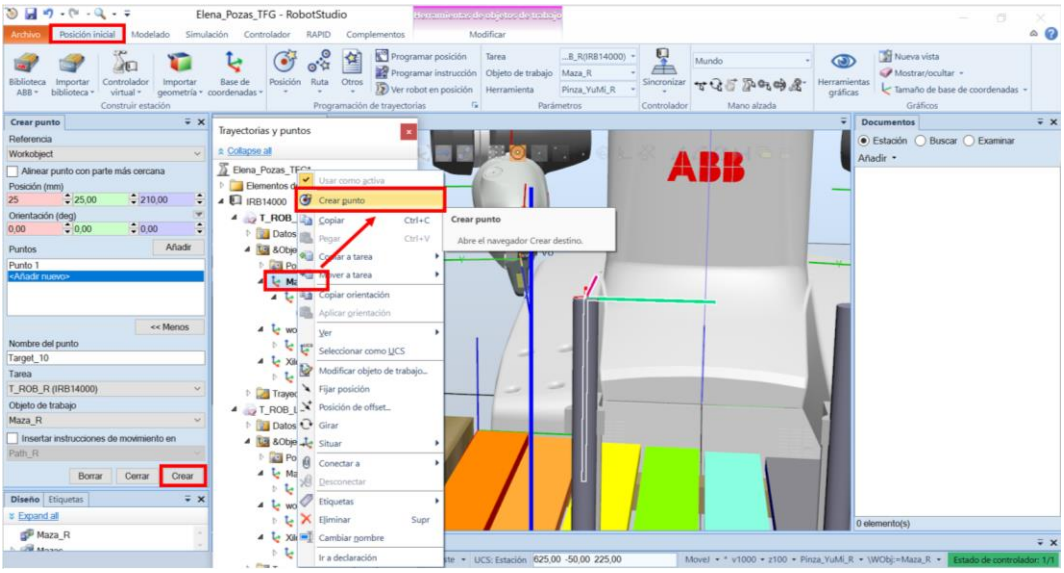


Figura 57. Crear Robtarget.

En la Figura 58 se muestran todas las trayectorias y puntos que están definidos en la estación.

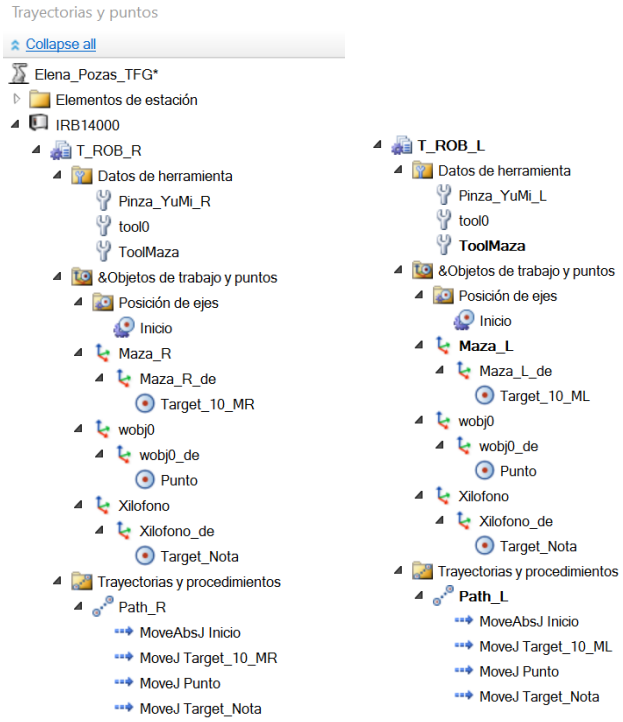


Figura 58. Trayectorias y puntos.

4.2. SEGUNDA FASE: Programación RAPID

En este apartado se analizará el código RAPID programado para el robot. Se pueden distinguir dos tipos de rutina distintos, procedimientos y rutinas TRAP.

A este conjunto de rutinas, junto con un conjunto de declaraciones de datos, se le conoce como módulo. Existen dos módulos distintos, uno para cada brazo, con variables y rutinas muy similares: el módulo "Xilofono_L" para el brazo izquierdo, y el módulo "Xilofono_R" para el brazo derecho. El procedimiento principal de estos módulos es llamado a su vez por otro módulo denominado "M_main". En la Figura 59 se muestra un diagrama de llamadas de procedimientos, donde se puede ver qué procedimiento es llamado por cuál.

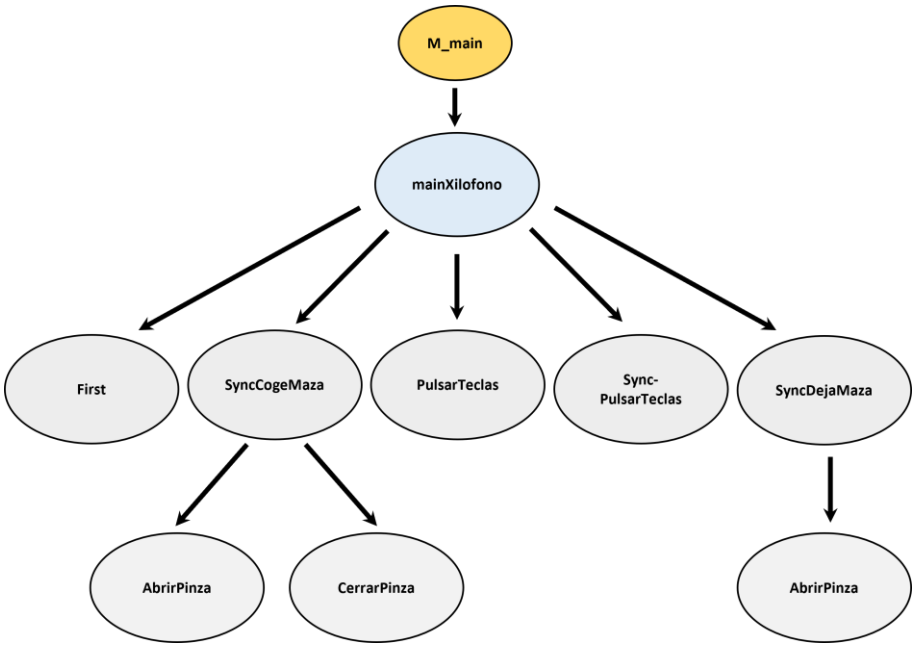


Figura 59. Diagrama de llamadas de procedimientos.

4.2.1. Variables globales

Las variables declaradas al inicio del código corresponden a los puntos definidos en la estación, explicados en el apartado 4.1.5. Para poder obtener las coordenadas de estos puntos y objetos de trabajo, existe la posibilidad de sincronizarlos con RAPID. En la pestaña "Posición inicial", en "Sincronizar", se selecciona la opción de sincronizar con RAPID y se abre una ventana donde poder seleccionar los puntos que se quieren sincronizar, así como el módulo donde se quieren utilizar esos puntos, y el tipo de variable requerida para cada punto. Para poder sincronizar un robtarget, los puntos tienen que estar dentro del Path de "Trayectorias y Posiciones" (Figura 60).

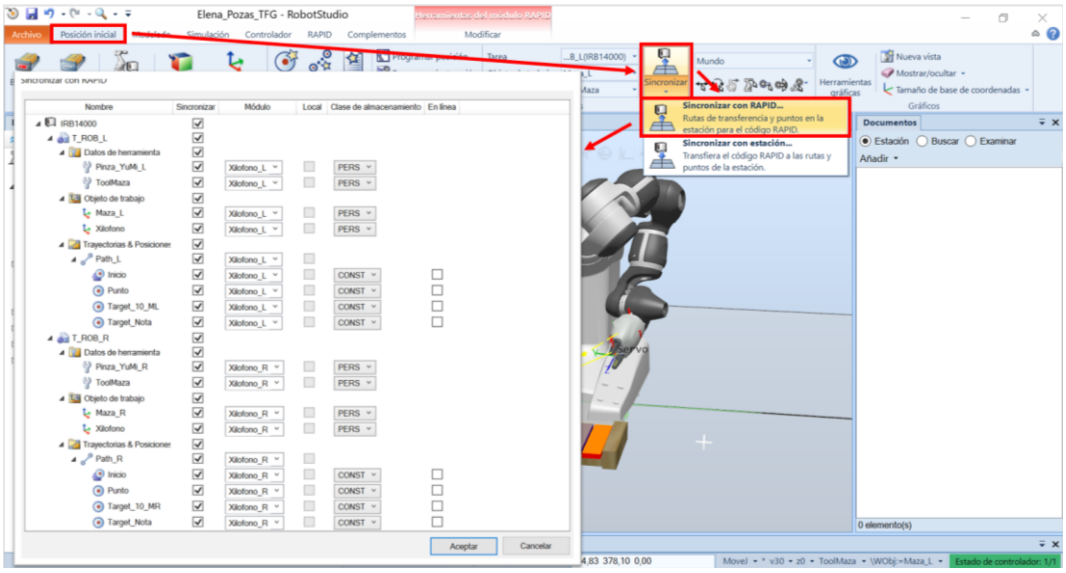


Figura 60. Sincronizar con RAPID.

Automáticamente aparecerán los puntos en el módulo seleccionado.

Los puntos pertenecientes al módulo de la tarea del brazo izquierdo se muestran en la Figura 61.

```
MODULE Xilofono_L
    CONST jointtarget Inicio:=[[0,-130,30,0,40,0],[135,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PERS tooldata Pinza_YuMi_L:=[TRUE,[[0,0,114.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    PERS tooldata ToolMaza:=[TRUE,[[0,0,314.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

    PERS wobjdata Maza_L:=[FALSE,TRUE,"",[[600,25,15],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    PERS wobjdata Xilofono:=[FALSE,TRUE,"",[[200,-174,30],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget Punto:=[[89.387953825,156.622113832,160],
[0.066010723,0.842420916,-0.111214903,0.523068666],[0,0,0,4],[101.964426653,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget Target_10_ML:=[[25,25,210],[0,0,1,0],[0,0,0,0],[101.96443128,9E+09,9E+09,9E
+09,9E+09,9E+09]];
    CONST robtarget Target_Nota:=[[132.5,-24,19.5],
[0.320796829,0.781743202,-0.266041626,0.463884484],[-1,1,-1,4],[111.912014269,9E+09,9E+09,9E+09,9E
+09,9E+09]];
```

Figura 61. Puntos de la estación en RAPID T_ROB_R.

Los puntos pertenecientes al módulo de la tarea del brazo derecho se muestran en la Figura 62.

```
MODULE Xilofono_R
    CONST jointtarget Inicio:=[[0,-130,30,0,40,0],[-135,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PERS tooldata Pinza_YuMi_R:=[TRUE,[[0,0,114.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    PERS tooldata ToolMaza:=[TRUE,[[0,0,314.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

    PERS wobjdata Maza_R:=[FALSE,TRUE,"",[[600,-75,15],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    PERS wobjdata Xilofono:=[FALSE,TRUE,"",[[200,-174,30],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget Punto:=[[89.387953825,-156.622113832,160],
[0.066010723,-0.842420916,-0.111214903,-0.523068666],[0,0,0,4],[-101.964426653,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget Target_10_MR:=[[25,25,210],[0,0,1,0],[0,0,0,0],[-101.964427132,9E+09,9E+09,9E
+09,9E+09,9E+09]];
    CONST robtarget Target_Nota:=[[132.5,-24,19.5],
[0.320796829,-0.781743202,-0.266041626,-0.463884484],[1,-1,1,4],[-111.912014269,9E+09,9E+09,9E+09,9E
+09,9E+09]];
```

Figura 62. Puntos de la estación en RAPID T_ROB_L.

A continuación, aparecen en el programa el resto de las variables globales que no son puntos de la estación y que se utilizan en varios procedimientos, o se inicializan fuera de estos. En la Figura 63 se puede ver la declaración de:

- Las variables necesarias para las interrupciones del programa.
- Variables de tipo syncident para identificar qué instrucciones de MultiMove de los distintos programas de tarea deben estar sincronizadas entre sí.
- Variables tipo num, string y bool para el resto de la programación del módulo.

Las variables que van a ser leídas o escritas por MATLAB deben estar declaradas como tipo PERS para poder establecer la comunicación.

Este tipo de variables se caracteriza porque cada vez que cambia su valor durante la ejecución del programa, también se cambia el valor de su inicialización. Por ello para inicializar siempre estas variables con un valor determinado se igualan a ese valor al principio del programa.

```
! Variables de interrupción
VAR intnum Idi0;
VAR intnum Idi1;
VAR intnum Idi2;
VAR intnum Idi3;
VAR intnum Idi4;
VAR intnum Idi5;
VAR intnum Idi6;
VAR intnum Idi7;
VAR intnum Idi8;
VAR intnum Idi9;
VAR intnum Idi10;
VAR intnum Idi11;
VAR intnum Idi12;

! VARIABLES Tasks y syncident (Sincronización)
PERS tasks Tareas{2}:=[["T_ROB_L"],["T_ROB_R"]];
VAR syncident sync1;
VAR syncident sync2;
VAR syncident sync3;
VAR syncident sync4;

! Distancia entre teclas
CONST num Tecla:=44;

! Variables de interrupción
PERS bool nosalir;
VAR num opcion;

! Variables escritas desde MATLAB
PERS string puls_nota_write:="3/1/6/8/6/5"; ! Notas leídas del fichero
PERS num puls_nota_RT:=1; ! Nota pulsada en tiempo real

! Variables leídas por MATLAB
PERS num ocupado:=0; ! Indica si el brazo izquierdo está ocupado o no
PERS num puls_nota:=0; ! Nota leída en todo momento por MATLAB

PERS bool puls_R:=FALSE; ! Variable a supervisar por SearchL
VAR num paso:=1;
VAR robtarget sp;
VAR string puls_nota_wr{100}; ! Notas leídas del fichero en string
VAR num puls_nota_num{100}; ! Notas leídas del fichero en num
CONST string notas{8}:=["1","2","3","4","5","6","7","8"];
VAR num orden:=0;
VAR num l:=1;
```

Figura 63. Variables globales.

4.2.2. Procedimientos

Un procedimiento es un conjunto de instrucciones que no devuelve ningún valor.

A continuación, se muestran todos los procedimientos de los que se compone el programa.

AbrirPinza

En la Figura 64 se muestra el procedimiento “AbrirPinza” del brazo izquierdo. Este procedimiento se utiliza para abrir la pinza izquierda del robot y así poder coger o dejar las mazas. En la Figura 49 se puede ver como la entrada de la pinza izquierda que sirve para abrir la pinza está conectada con la salida DO_14 del controlador. Por ello, se le da valor uno a esta salida. Después de 2 segundos, suficientes para que la pinza se haya abierto, se reinicia ese valor.

```
! Abre la pinza
LOCAL PROC AbrirPinza()
    Set DO_14; ! Abrir y desattacher
    WaitTime 2;
    Reset DO_14;
ENDPROC
```

Figura 64. PROC AbrirPinza T_ROB_R.

En la Figura 65 se recoge el procedimiento “AbrirPinza” del brazo derecho, muy similar al del brazo izquierdo, con la diferencia de que la entrada de la pinza derecha que sirve para abrir la pinza está conectada con la salida DO_12, y no con la salida DO_14, como se puede ver en la Figura 49.

```
! Abre la pinza
LOCAL PROC AbrirPinza()
    Set DO_12; ! Abrir y desattacher
    WaitTime 2;

    Reset DO_12;
ENDPROC
```

Figura 65. PROC AbrirPinza T_ROB_L.

CerrarPinza

En la Figura 66 se recoge el procedimiento “CerrarPinza”. Este procedimiento se utiliza para cerrar la pinza izquierda del robot y así poder coger las mazas. En la Figura 49 se puede ver como la entrada de la pinza izquierda que sirve para cerrar la pinza está conectada con la salida DO_15 del controlador. Por ello, se le da valor uno a esta salida. Después de 2 segundos, suficientes para que la pinza se haya abierto, se reinicia ese valor.

```
! Cerrar Pinza
LOCAL PROC CerrarPinza()
    Set DO_15;  ! Activar los sensores de attacher
    WaitTime 2;
    Reset DO_15;
ENDPROC
```

Figura 66. PROC CerrarPinza T_ROB_R.

En la Figura 67 se puede ver el procedimiento “CerrarPinza” del brazo derecho, muy similar al del brazo izquierdo, con la diferencia de que la entrada de la pinza derecha que sirve para cerrar la pinza está conectada con la salida DO_13, y no con la salida DO_15, como se puede ver en la Figura 49.

```
! Cerrar Pinza
LOCAL PROC CerrarPinza()
    Set DO_13;  ! Activar los sensores de attacher
    WaitTime 2;
    Reset DO_13;
ENDPROC
```

Figura 67. PROC CerrarPinza T_ROB_L.

First

Este procedimiento se ejecuta siempre al compilar el programa. Es fundamental para poder utilizar las interrupciones. Primero se cancelan todas las interrupciones que podrían haberse activado antes de comenzar la simulación, y después se conecta cada variable de interrupción, previamente definida, con la rutina TRAP deseada. Por último, se asocia una entrada del controlador a la variable de interrupción correspondiente. Este procedimiento es igual para los dos brazos. En el brazo derecho se iguala la variable que contiene el número de la nota pulsada a cero para que comience siempre con ese valor. En la Figura 68 se puede ver el código de este procedimiento.

```
LOCAL PROC First()
    puls_nota:=0;

    ! Cancelamos las interrupciones
    IDelete Idi0;
    IDelete Idi1;
    IDelete Idi2;
    IDelete Idi3;
    IDelete Idi4;
    IDelete Idi5;
    IDelete Idi6;
    IDelete Idi7;
    IDelete Idi8;
    IDelete Idi9;
    IDelete Idi10;
    IDelete Idi11;
    IDelete Idi12;

    ! Conectar var intnum con trap
    CONNECT Idi0 WITH trap_1_DO;
    CONNECT Idi1 WITH trap_2_RE;
    CONNECT Idi2 WITH trap_3_MI;
    CONNECT Idi3 WITH trap_4_FA;
    CONNECT Idi4 WITH trap_5_SOL;
    CONNECT Idi5 WITH trap_6_LA;
    CONNECT Idi6 WITH trap_7_SI;
    CONNECT Idi7 WITH trap_8_DO;
    CONNECT Idi8 WITH trap_CogeMaza;
    CONNECT Idi9 WITH trap_TiempoReal;
    CONNECT Idi10 WITH trap_Fichero;
    CONNECT Idi11 WITH trap_DejaMaza;
    CONNECT Idi12 WITH trap_Fin;

    ISignalDI DI_0, 1, Idi0;
    ISignalDI DI_1, 1, Idi1;
    ISignalDI DI_2, 1, Idi2;
    ISignalDI DI_3, 1, Idi3;
    ISignalDI DI_4, 1, Idi4;
    ISignalDI DI_5, 1, Idi5;
    ISignalDI DI_6, 1, Idi6;
    ISignalDI DI_7, 1, Idi7;
    ISignalDI DI_8, 1, Idi8;
    ISignalDI DI_9, 1, Idi9;
    ISignalDI DI_10, 1, Idi10;
    ISignalDI DI_11, 1, Idi11;
    ISignalDI DI_12, 1, Idi12;

    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;

ENDPROC
```

Figura 68. PROC First T_ROB_R.

Escritura

Una de las opciones que existen en este programa, es la de leer un fichero que incluye las notas de una canción. Este fichero es leído por MATLAB, el cual se encarga de escribir en RAPID todas las notas en una variable de tipo string. Para poder obtener el valor numérico de cada nota y separar cada uno de los caracteres de la variable string, se utiliza el procedimiento que se puede ver en la Figura 69. Además, también se utiliza para saber si el número de notas almacenadas en el fichero es par o impar. Este procedimiento es el mismo para las dos tareas.

```
! Crear vector de números con las notas que se leen del fichero
LOCAL PROC Escritura()
    VAR string frag;
    VAR string puls_nota_w{100};
    VAR num j;
    VAR num h;
    VAR num k;
    ! Se llena un vector con únicamente los números de cada nota
    FOR j FROM 1 TO StrLen(puls_nota_write) DO
        frag:= StrPart(puls_nota_write,j,1);
        puls_nota_w{j}:=frag;
        IF puls_nota_w{j}<>"/" THEN
            puls_nota_wr{j}:=puls_nota_w{j};
        ENDIF
    ENDFOR
    ! Se pasa el vector de las notas de string a num
    FOR h FROM 1 TO Dim(puls_nota_w,1) DO
        FOR k FROM 1 TO Dim(notas,1) DO
            IF puls_nota_w{h}=notas{k} THEN
                puls_nota_num{l}:=k;
                l:=l+1; ! Devuelve el total de notas solicitadas
            ENDIF
        ENDFOR
    ENDFOR
    ! Se calcula el orden del número total de notas, par o impar
    IF ((l-1) MOD 2) <>0 THEN
        orden:=1;
    ENDIF
ENDPROC
```

Figura 69. PROC Escritura T_ROB_R.

PulsarTeclas

Otra opción que existen en este programa es la de pulsar las notas según se van solicitando en tiempo real desde MATLAB. Las láminas que están más cerca del brazo derecho serán golpeadas por el brazo derecho y las notas que están más cerca del izquierdo serán golpeadas por el brazo izquierdo. Para que esto suceda así existe una variable que indica si el brazo está ocupado o no. Solo se golpeará una nota cuando ambos brazos estén libres. A continuación, se muestra este procedimiento para el brazo derecho (Figura 70) y para el brazo izquierdo (Figura 71).

```
! Pulsar teclas en tiempo real
LOCAL PROC PulsarTeclas(num notas)
    ocupado:=1;
    MoveL Offs(Target_Nota,0,2*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    puls_R:=FALSE;
    SearchL \Stop, puls_R, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    ocupado:=0;
    MoveL Offs(Target_Nota,0,2*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
ENDPROC
```

Figura 70. PROC PulsarTeclas T_ROB_R.

```
! Pulsar teclas en tiempo real
LOCAL PROC PulsarTeclas(num notas)
    ocupado:=1;
    MoveL Offs(Target_Nota,0,7*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    puls_L:=FALSE;
    SearchL \Stop, puls_L, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    ocupado:=0;
    MoveL Offs(Target_Nota,0,7*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
ENDPROC
```

Figura 71. PROC PulsarTeclas T_ROB_L.

SyncCogeMaza

Para poder coger las mazas, se ha programado el procedimiento "SyncCogeMaza", al cual se le llama al ser seleccionada la primera opción del programa. El objetivo principal es que el robot coja las dos mazas de manera sincronizada, sin que colisione un brazo con otro. Para ello se han utilizado distintos tipos de datos de MultiMove, los cuales se describen en la Tabla 16 [22].

Tipo de dato	Descripción
syncident	Se utiliza para identificar qué instrucciones "WaitSyncTask" de los distintos programas de tarea deben estar sincronizadas entre sí. El nombre de la variable "syncident" debe ser igual en todos los programas de tarea.
tasks	La variable persistente del tipo de dato "tasks" contiene los nombres de las tareas que se sincronizarán con "WaitSyncTask" o "SyncMoveOn". Las variables "tasks" deben estar declaradas como una variable persistente global del sistema, con el mismo nombre y el mismo contenido en todos los programas de tarea.

Tabla 16. Datos de MultiMove.

Además, se ha utilizado una instrucción de MultiMove para definir puntos de sincronización, convertidos en puntos de paro, ya que el programa de tarea tiene que esperar a otro programa de tarea. Esta instrucción aparece descrita en la Tabla 17.

Instrucción	Descripción
WaitSyncTask	Se emplea para sincronizar varios programas de tarea en un punto especial del programa y esperar a los otros programas de tarea. Cuando todos los programas de tarea han alcanzado la instrucción "WaitSyncTask", prosigue su ejecución.

Tabla 17. Instrucción de MultiMove.

Este procedimiento necesita una variable de tipo syncident. Primero se moverá el brazo derecho, cogerá la maza derecha, y antes de llegar a su posición de reposo, el brazo izquierdo que estaba esperando a que el brazo derecho llegase a un punto determinado, comenzará a coger la maza izquierda. Todas estas

instrucciones aparecen reflejadas en la Figura 72 (brazo derecho) y en la Figura 73 (brazo izquierdo).

```
! Coger maza derecha
LOCAL PROC SyncCogeMaza()
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;

    AbrirPinza;

    MoveJ Offs(Target_10_MR,0,0,100),v300,z0,Pinza_YuMi_R\WObj:=Maza_R;
    MoveL Offs(Target_10_MR,0,0,0),v50,fine,Pinza_YuMi_R\WObj:=Maza_R;

    CerrarPinza;

    WaitSyncTask sync1,Tareas;

    MoveL Offs(Target_10_MR,0,0,50),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
    MoveL Offs(Target_10_MR,10,0,50),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
    MoveJ Punto,v300,fine,Pinza_YuMi_R;
ENDPROC
```

Figura 72. PROC SyncCojeMaza T_ROB_R.

```
! Coger maza izquierda
LOCAL PROC SyncCogeMaza()
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L;

    AbrirPinza;

    WaitSyncTask sync1,Tareas;

    MoveJ Offs(Target_10_ML,0,0,100),v300,z0,Pinza_YuMi_L\WObj:=Maza_L;
    MoveL Offs(Target_10_ML,0,0,0),v50,fine,Pinza_YuMi_L\WObj:=Maza_L;

    CerrarPinza;

    MoveL Offs(Target_10_ML,0,0,50),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
    MoveL Offs(Target_10_ML,10,0,50),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
    MoveJ Punto,v300,fine,Pinza_YuMi_L;
ENDPROC
```

Figura 73. PROC SyncCogeMaza T_ROB_L.

SyncDejaMaza

Este procedimiento se utiliza en la cuarta opción del programa, y sirve para que el robot vuelva a colocar las mazas en su sitio inicial. Las instrucciones de movimiento son similares a coger las mazas. Solo se necesitará una instrucción de MultiMove “WaitSyncTask” para que no colisionen los brazos.

El código de este procedimiento se puede ver en la Figura 74 (para el brazo derecho) y en la Figura 75 (para el brazo izquierdo).

```
! Dejar maza derecha
LOCAL PROC SyncDejaMaza()
    MoveJ Punto,v100,z0,Pinza_YuMi_R;
    MoveJ Offs(Target_10_MR,10,0,50),v300,z0,Pinza_YuMi_R\WObj:=Maza_R;
    MoveL Offs(Target_10_MR,0,0,50),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
    MoveL Offs(Target_10_MR,0,0,0),v50,fine,Pinza_YuMi_R\WObj:=Maza_R;

    AbrirPinza;

    WaitSyncTask sync4,Tareas;

    MoveL Offs(Target_10_MR,0,0,100),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;
ENDPROC
```

Figura 74. PROC SyncDejaMaza T_ROB_R.

```
! Dejar maza izquierda
LOCAL PROC SyncDejaMaza()
    MoveJ Punto,v300,fine,Pinza_YuMi_L;

    WaitSyncTask sync4,Tareas;

    MoveJ Offs(Target_10_ML,10,0,50),v300,z0,Pinza_YuMi_L\WObj:=Maza_L;
    MoveL Offs(Target_10_ML,0,0,50),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
    MoveL Offs(Target_10_ML,0,0,0),v50,fine,Pinza_YuMi_L\WObj:=Maza_L;

    AbrirPinza;

    MoveL Offs(Target_10_ML,0,0,100),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L;
ENDPROC
```

Figura 75. PROC SyncDejaMaza T_ROB_L.

SyncPulsarTeclas

En la Figura 76 y en la Figura 77 se muestra el código de los movimientos que se utilizan para que el robot golpee las láminas. Para ello se necesita que los brazos estén coordinados, por lo que se usarán dos instrucciones MultiMove por cada movimiento. Sin embargo, para golpear la primera y última nota se requiere de una única instrucción de este tipo, ya que los brazos no empiezan a moverse a la vez.

El brazo derecho comenzará golpeando las notas. Para la primera nota, este brazo no necesita ningún tipo de sincronización al comienzo del movimiento, ya que solo está ejecutándose este brazo, por lo que este procedimiento cuenta con una variable que indica el número de nota que se está ejecutando, denominada “paso”. Lo mismo sucede cuando se golpea la última nota. El último brazo que golpee la nota no necesitará la última instrucción de MultiMove. Ahora bien, no siempre será el mismo brazo el último en golpear la nota. Dependiendo de si el número de notas solicitadas es par o impar, acabará el brazo izquierdo o el brazo derecho, y dependiendo de eso un brazo tendrá o

no tendrá la última instrucción de sincronización. Todo esto se contempla en este procedimiento con sentencias “IF”.

En el esquema de la Figura 78 se representan los diferentes movimientos que se realizan cuando se llama a este procedimiento dentro de un bucle. En dicho esquema, se muestra de forma visual cuándo se utilizan las instrucciones MultiMove, así como el valor que tiene en cada momento la variable con el número de nota que se ha pulsado. Esto será útil para el momento en el que MATLAB lea esta variable. Si vale distinto de cero, generará el valor de esa nota, y si vale cero sabrá que la próxima vez que cambie su valor se habrá golpeado una nota nueva.

Además, en este procedimiento se utiliza un tipo de instrucción que realiza una búsqueda lineal usando el robot, denominada “SearchL”.

Para golpear una tecla, el robot realiza tres movimientos básicos:

- Desplazamiento a una posición base, un poco más alejado del xilófono que la posición de reposo para que no golpee ninguna nota no deseada.
- Desplazamiento a una posición con el valor del eje y correspondiente a la nota que se quiere golpear.
- Desplazamiento a otra posición en sentido lineal para buscar la posición en la que se encuentra la lámina del xilófono.

Durante el movimiento, el robot supervisa una variable declarada como tipo PERS, la cual actúa como señal digital de entrada. Cuando el valor de la señal cambia al valor solicitado, el robot lee inmediatamente la posición actual, permitiendo obtener las coordenadas de contorno del xilófono.

Al utilizar el bus de campo DeviceNet, obtenemos tiempo breves, dado que utiliza el cambio de estado como tipo de conexión.

Se utiliza como argumento [\Stop], con el que el robot se detiene lo antes posible, sin mantener el TCP en la trayectoria (paro rígido) cuando el valor de la señal de búsqueda cambia a activo. Sin embargo, el robot se mueve una distancia corta antes del paro y no regresa a la posición buscada, es decir, a la posición en la que cambió la señal.

El cambio del valor de la señal digital a supervisar se realiza por medio de interrupciones. Cuando la maza atraviesa el sensor de la tecla que va a golpear, este activa la señal de entrada del controlador de la nota correspondiente y se interrumpe el movimiento con una rutina TRAP que contiene la señal digital a supervisar igualada a uno.

Además, esta instrucción también necesita dos robtarget:

- Uno con la posición del TCP y de los ejes externos en el momento del disparo de la señal de búsqueda, denominado en este caso sp.
- Otro con el punto de destino de los ejes del robot y de los ejes externos, que en este caso dependerá de la nota solicitada.

El resto de los argumentos son los que se utilizan en todas las instrucciones de movimiento, como la velocidad, la herramienta y el objeto de trabajo.

```
! Pulsar teclas leyendo las notas de un fichero
LOCAL PROC SyncPulsarTeclas(num notas)
    IF paso<>1 THEN
        WaitSyncTask sync3,Tareas;
        puls_nota:=0;
        WaitTime 1;
    ENDIF

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v150,z0,ToolMaza\WObj:=Xilofono;

    puls_R:=FALSE;
    SearchL \Stop, puls_R, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    IF ( orden=1 AND paso=(l/2)) THEN
        MoveL Offs(Target_Nota,0,2*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
    ELSE
        MoveL Offs(Target_Nota,0,2*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
        puls_nota:=0;
        WaitSyncTask sync2,Tareas;
    ENDIF
    paso:=paso+1;
ENDPROC
```

Figura 76. PROC SyncPulsarTeclas T_ROB_R.

```
! Pulsar teclas leyendo las notas de un fichero
LOCAL PROC SyncPulsarTeclas(num notas)
    WaitSyncTask sync2,Tareas;
    WaitTime 1;

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v150,z0,ToolMaza\WObj:=Xilofono;

    puls_L:=FALSE;
    SearchL \Stop, puls_L, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    IF (orden=0 AND paso=((l-1)/2)) THEN
        MoveL Offs(Target_Nota,0,7*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
    ELSE
        MoveL Offs(Target_Nota,0,7*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
        WaitSyncTask sync3,Tareas;
    ENDIF
    paso:=paso+1;
ENDPROC
```

Figura 77. PROC SyncPulsarTeclas T_ROB_L.

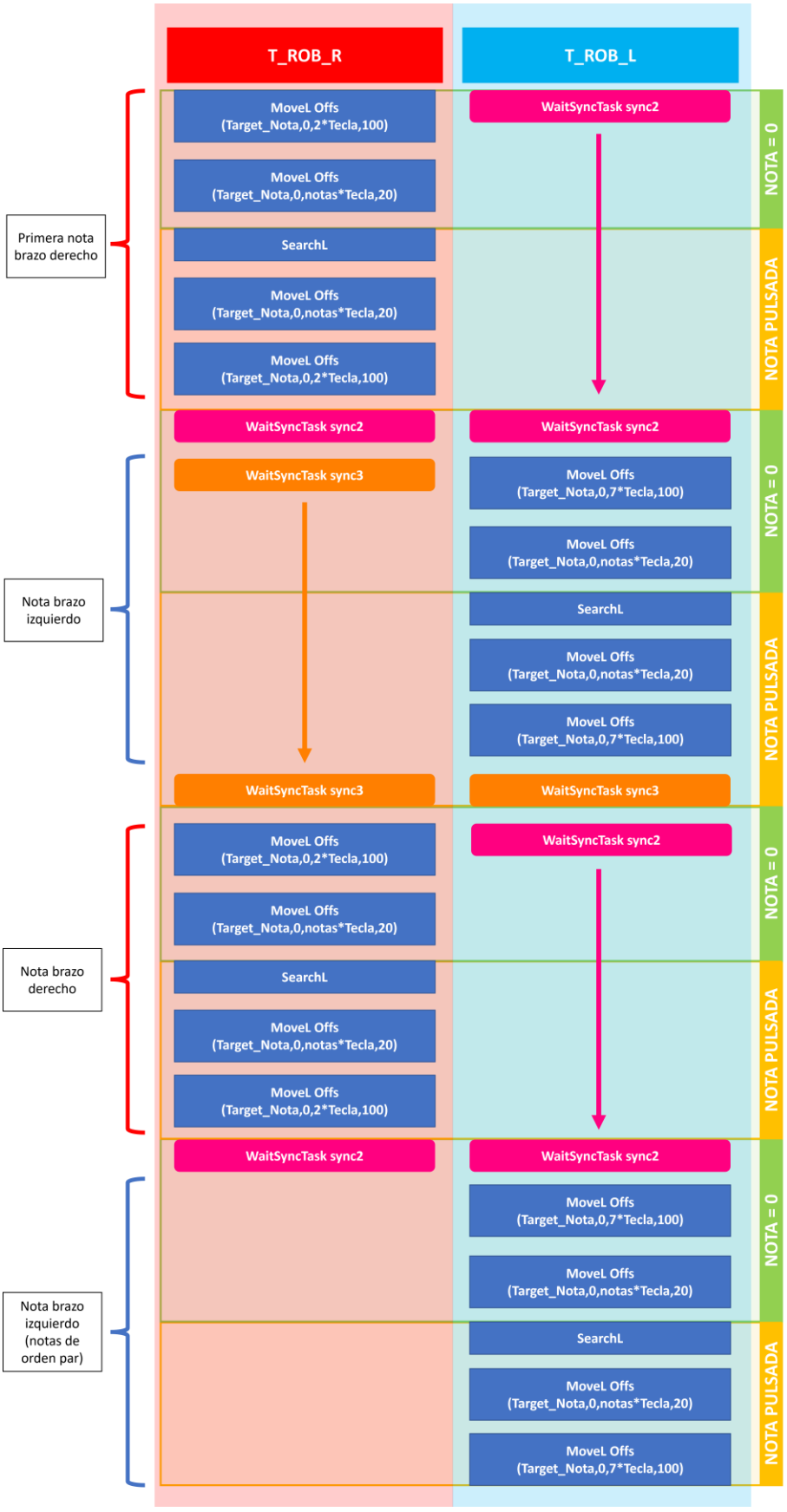


Figura 78. Esquema PROC SyncPulsarTeclas.

mainXilofono

Este procedimiento es el principal del robot, donde se puede encontrar el test con las diferentes opciones con las que cuenta el programa, como se muestra en la Figura 80 (brazo derecho) y en la Figura 81 (brazo izquierdo). En función de la entrada del controlador pulsada, se ejecutará una opción u otra al llamar a la rutina TRAP correspondiente, que cambia el valor de la variable "opcion".

Las opciones con las que cuenta el programa son las siguientes:

- **Opción 1:** Coger Mazas.
- **Opción 2:** Golpear las notas que van siendo solicitadas en tiempo real.
- **Opción 3:** Golpear las notas que se han leído previamente de un fichero. Para esta opción, el programa cuenta con un bucle WHILE que va leyendo un vector con todas las notas almacenadas del fichero. En función de si la posición de la nota es par o impar, esa nota será golpeada por el brazo izquierdo o por el brazo derecho.
- **Opción 4:** Dejar Mazas.
- **Opción 5:** Finalizar programa.
  Si se selecciona esta opción, como su propio nombre indica, el programa finalizará.

En la Figura 79 se ha definido el diagrama de bloques correspondiente a la programación del "main" de los módulos "Xilofono_R" y "Xilofono_L", utilizando los símbolos normalizados para los diagramas de bloques que se muestran en la Tabla 18.

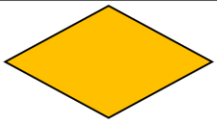
	Inicio o fin del programa.
	Acción o proceso.
	Toma de decisiones.
	Entrada de información.

Tabla 18. Símbolos normalizados para diagramas de bloques.

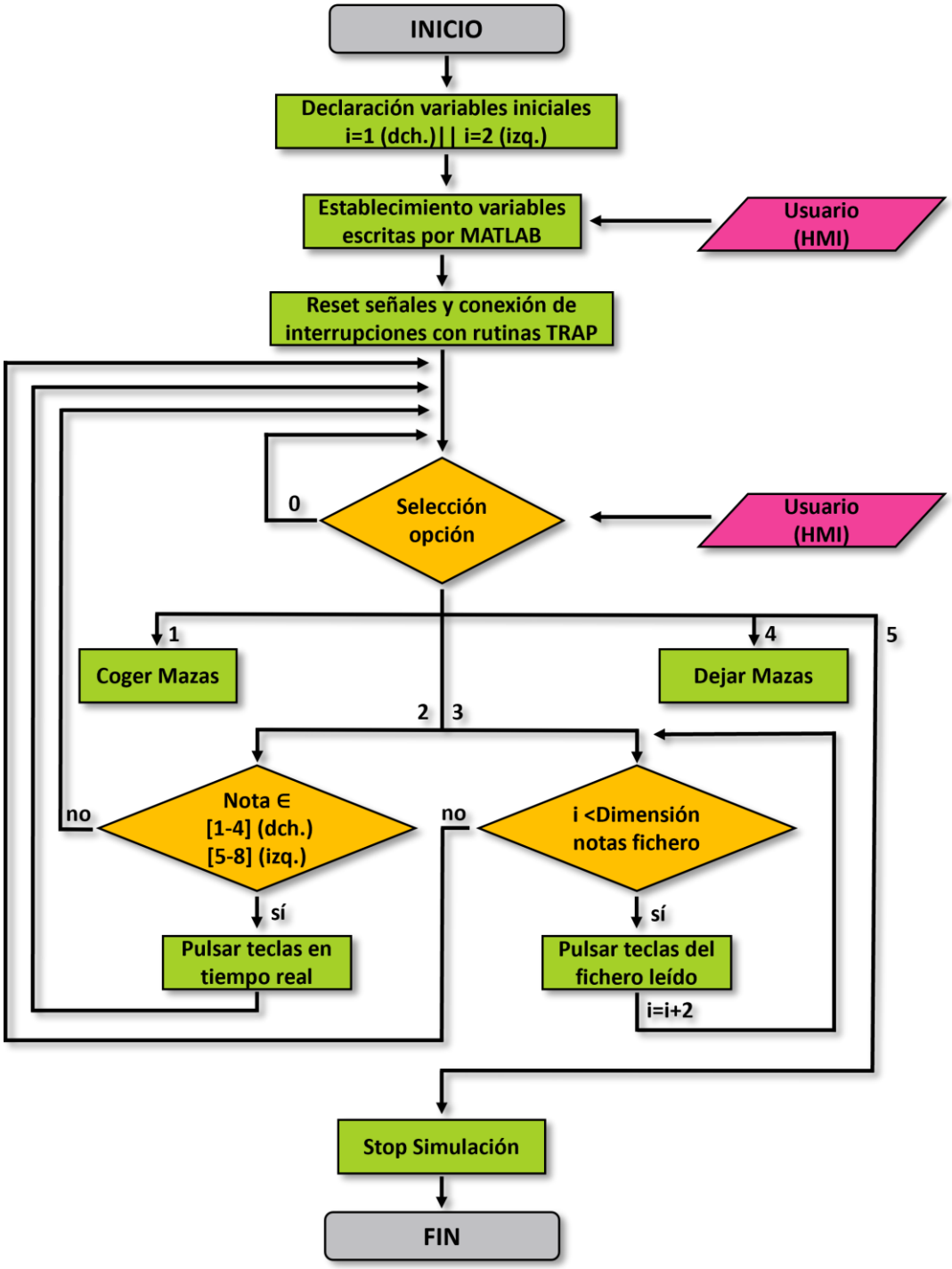


Figura 79. Diagrama de bloques del código del programa.

```
PROC mainXilofono_R()
    VAR num i:=1;
    VAR num n:=0;
    VAR num rango{4}:=[1,2,3,4];

    First;

    nosalir:=TRUE;
    opcion:=0;

    WHILE nosalir DO
        TEST opcion
        CASE 1:
            opcion:=0;
            SyncCogeMaza;
        CASE 2:
            ocupado:=0;
            opcion:=0;
            puls_nota:=0;
            FOR i FROM 1 TO 4 DO
                IF puls_nota_RT=rango{i} THEN
                    PulsarTeclas(puls_nota_RT);
                ENDIF
            ENDFOR
        CASE 3:
            opcion:=0;
            puls_nota:=0;
            WaitTime 1.5;
            l:=1;
            paso:=1;
            i:=1;
            n:=0;
            Escritura;
            WHILE i < Dim(puls_nota_num,1) AND puls_nota_num{i}<>0 DO
                IF puls_nota_num{i}<>0 THEN
                    n:=puls_nota_num{i};
                    SyncPulsarTeclas n;
                    i:=i+2;
                ENDIF
            ENDWHILE
            puls_nota:=0;
            MoveJ Punto,v100,z0,Pinza_YuMi_R;
        CASE 4:
            opcion:=0;
            SyncDejaMaza;
        CASE 5:
            opcion:=0;
            nosalir:=FALSE;
        DEFAULT:
        ENDTEST
    ENDWHILE

    TPWrite "FIN DEL PROGRAMA R";
    Stop;
ENDPROC
```

Figura 80. PROC mainXilofono_R T_ROB_R.

```
PROC mainXilofono_L()
    VAR num i:=2;
    VAR num n:=0;
    VAR num rango{4}:=[5,6,7,8];

    First;

    TPWrite "1.- Coger Mazas (DI_8)";
    TPWrite "2.- Leer notas en tiempo real (DI_9)";
    TPWrite "3.- Leer notas de un fichero (DI_10)";
    TPWrite "4.- Dejar Mazas (DI_11)";
    TPWrite "5.- Finalizar programa (DI_12)";

    nosalir:=TRUE;
    opcion:=0;

    WHILE nosalir DO
        TEST opcion
        CASE 1:
            opcion:=0;
            TPWrite "Coger mazas";
            SyncCogeMaza;
        CASE 2:
            ocupado:=0;
            opcion:=0;
            TPWrite "Leer notas en tiempo real";
            FOR i FROM 1 TO 4 DO
                IF puls_nota_RT=rango{i} THEN
                    PulsarTeclas(puls_nota_RT);
                ENDIF
            ENDFOR
        CASE 3:
            opcion:=0;
            l:=1;
            paso:=1;
            i:=2;
            n:=0;
            Escritura;
            TPWrite "Leer notas de un fichero";
            WHILE i < Dim(puls_nota_num,1) AND puls_nota_num{i}<>0 DO
                IF puls_nota_num{i}<>0 THEN
                    n:=puls_nota_num{i};
                    SyncPulsarTeclas n;
                    i:=i+2;
                ENDIF
            ENDWHILE
            MoveJ Punto,v100,z0,Pinza_YuMi_L;
        CASE 4:
            opcion:=0;
            TPWrite "Dejar mazas";
            SyncDejaMaza;
        CASE 5:
            TPWrite "Finalizar programa";
            opcion:=0;
            nosalir:=FALSE;
        DEFAULT:
        ENDTEST
    ENDWHILE

    TPWrite "FIN DEL PROGRAMA L";
    Stop;
ENDPROC
```

Figura 81. PROC mainXilofono_L T_ROB_L.

4.2.3. Rutinas TRAP

Una rutina TRAP es un conjunto de instrucciones que es disparado por una interrupción.

Este programa cuenta con trece rutinas TRAP. Ocho de estas rutinas sirven para cambiar el valor de la señal a supervisar por la instrucción SearchL y el valor de la variable que contiene el número de nota golpeada, en el caso de la tarea del brazo derecho, la cual será leída por MATLAB posteriormente para que el xilófono suene correctamente. Serán activadas por señales de entrada del controlador, activadas por el sensor colocado sobre las láminas. Las cinco rutinas restantes servirán para seleccionar la opción correcta que se ha

solicitado en el menú que contiene el procedimiento principal. Serán activadas por señales de entrada del controlador, activadas por el usuario a través de la interfaz de MATLAB. Si se activa la opción que finaliza el programa, se activará una rutina que parará el robot en el instante en el que es activada, y a continuación volverá a su posición base, a modo de seguridad.

```
LOCAL TRAP trap_1_DO
    puls_nota:=1;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_2_RE
    puls_nota:=2;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_3_MI
    puls_nota:=3;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_4_FA
    puls_nota:=4;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_5_SOL
    puls_nota:=5;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_6_LA
    puls_nota:=6;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_7_SI
    puls_nota:=7;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_8_DO
    puls_nota:=8;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_CogeMaza
    opcion:=1;
ENDTRAP

LOCAL TRAP trap_TiempoReal
    opcion:=2;
ENDTRAP

LOCAL TRAP trap_Fichero
    opcion:=3;
ENDTRAP

LOCAL TRAP trap_DejaMaza
    opcion:=4;
ENDTRAP

LOCAL TRAP trap_Fin
    opcion:=5;
    StopMove;
    StorePath;
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;
    RestoPath;
ENDTRAP
```

Figura 82. Rutinas TRAP T_ROB_R.

```
LOCAL TRAP trap_1_DO
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 1_DO";
ENDTRAP

LOCAL TRAP trap_2_RE
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 2_RE";
ENDTRAP

LOCAL TRAP trap_3_MI
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 3_MI";
ENDTRAP

LOCAL TRAP trap_4_FA
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 4_FA";
ENDTRAP

LOCAL TRAP trap_5_SOL
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 5_SOL";
ENDTRAP

LOCAL TRAP trap_6_LA
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 6_LA";
ENDTRAP

LOCAL TRAP trap_7_SI
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 7_SI";
ENDTRAP

LOCAL TRAP trap_8_DO
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 8_DO";
ENDTRAP

LOCAL TRAP trap_CogeMaza
    opcion:=1;
ENDTRAP

LOCAL TRAP trap_TiempoReal
    opcion:=2;
ENDTRAP

LOCAL TRAP trap_Fichero
    opcion:=3;
ENDTRAP

LOCAL TRAP trap_DejaMaza
    opcion:=4;
ENDTRAP
LOCAL TRAP trap_Fin
    opcion:=5;
    StopMove;
    StorePath;
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L
    RestoPath;
ENDTRAP
```

Figura 83. Rutinas TRAP T_ROB_L.

4.3. TERCERA FASE: Comunicación OPC UA

La arquitectura unificada OPC UA (Open Protocol Communication Unified Architecture) es un estándar de intercambio de datos para la comunicación industrial segura, fiable, independiente del fabricante e independiente de la plataforma. Permite el intercambio de datos seguro entre plataformas de hardware de distintos proveedores y entre sistemas operativos [23].

OPC UA incorpora el modelo de información orientado a objetos que aglutina las funcionalidades tradicionales de OPC (como acceso a datos, históricos, alarmas, eventos, condiciones…) y otras nuevas e innovadoras orientadas a los tipos de datos y métodos.

La característica de que la estructura del direccionamiento en los servidores OPC UA esté orientada a objetos y de que la interfaz para el acceso a dicha estructura sea totalmente genérica da lugar a que OPC UA se considere no solo una pasarela de comunicación entre dos equipos sino también un lenguaje de programación con capacidades de comunicación a través de redes [24].

Para poder leer y escribir variables de RobotStudio desde MATLAB, se ha desarrollado una comunicación OPC UA, tipo cliente-servidor, la cual se describe en el presente apartado. Dicha comunicación ha sido implementada mediante un servidor OPC ABB de RobotStudio y un cliente OPC de MATLAB.

En este caso, se busca escribir sobre las variables de RobotStudio que contienen las notas solicitadas, y leer las variables que contienen el número de la nota una vez ya golpeada para que pueda sonar correctamente desde MATLAB.

Para establecer la comunicación, se usa la aplicación ABB IRC5 OPC, la cual será la máquina que aloja el servidor OPC. Esta aplicación permite seleccionar el controlador con el que se conectará el servidor. Para ello, el controlador debe estar en estado iniciado en el PC.

Si se selecciona la opción “Scan” de la aplicación, aparece el nombre del controlador activo mediante un escaneo, permitiendo obtener el nombre del Alias con el que poder reconocer la conexión, como se puede ver en la Figura 84.

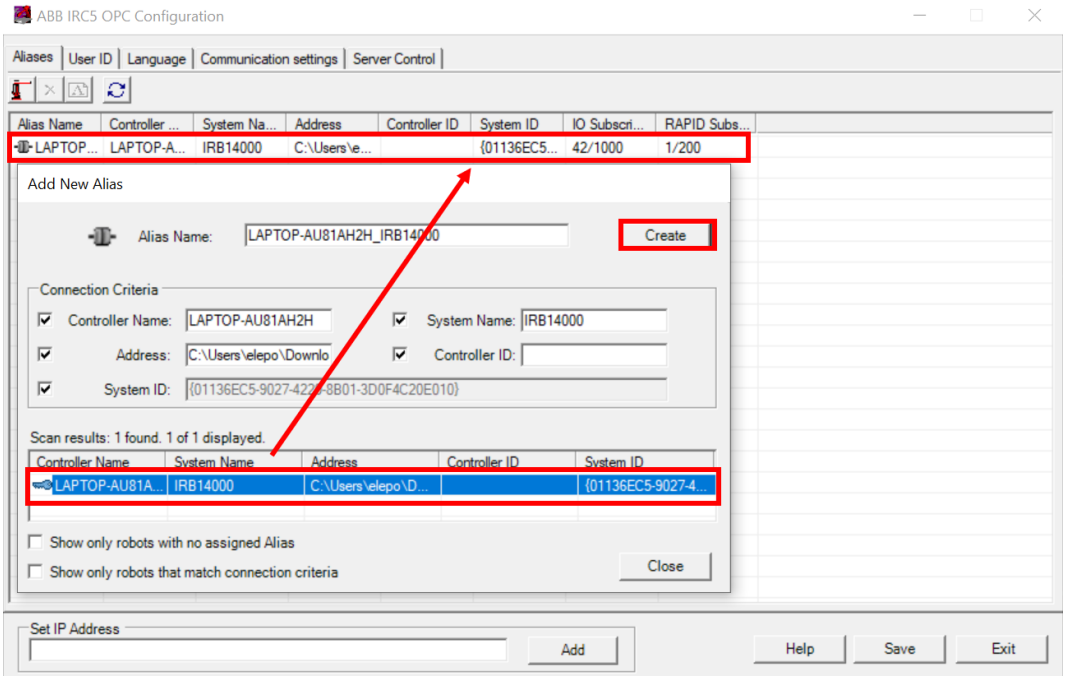


Figura 84. Añadir nuevo Alias.

Después de activar el servidor del controlador, ya se podría activar la comunicación.

El siguiente paso es, desde MATLAB, obtener el nombre de los elementos de acceso a datos. Para ello se usará el comando de MATLAB "opcDataAccessExplorer".

Dentro de esta herramienta se añade un Host, que corresponderá al nombre o la dirección IP de la máquina que aloja el servidor OPC, y se crea un cliente, como se puede ver en la Figura 85.

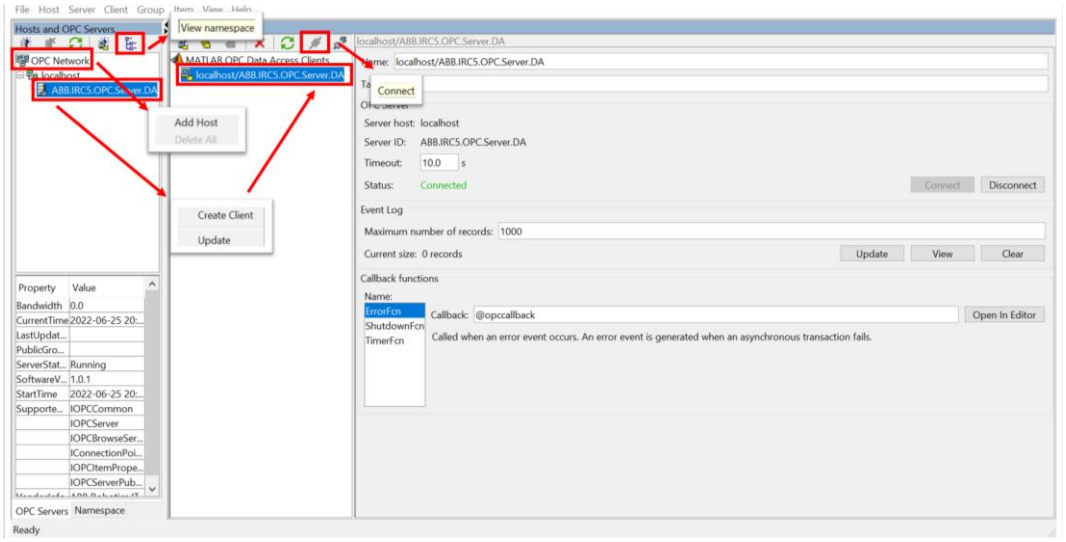


Figura 85. OPC Data Access Explorer.

Después se establece la conexión, y ya se puede acceder al espacio de nombres.

Todos los servidores OPC deben publicar un espacio de nombres, que consiste en una disposición del nombre de cada elemento del servidor (también conocido como ID de elemento) asociado con ese servidor. El espacio de nombres proporciona el mapa interno de cada dispositivo y ubicación que el servidor puede monitorear y/o actualizar [25].

Se selecciona el elemento del que se quiere obtener su ID y se añade a un grupo en el servidor para obtener su nombre dentro del servidor, como se puede ver en la Figura 86.

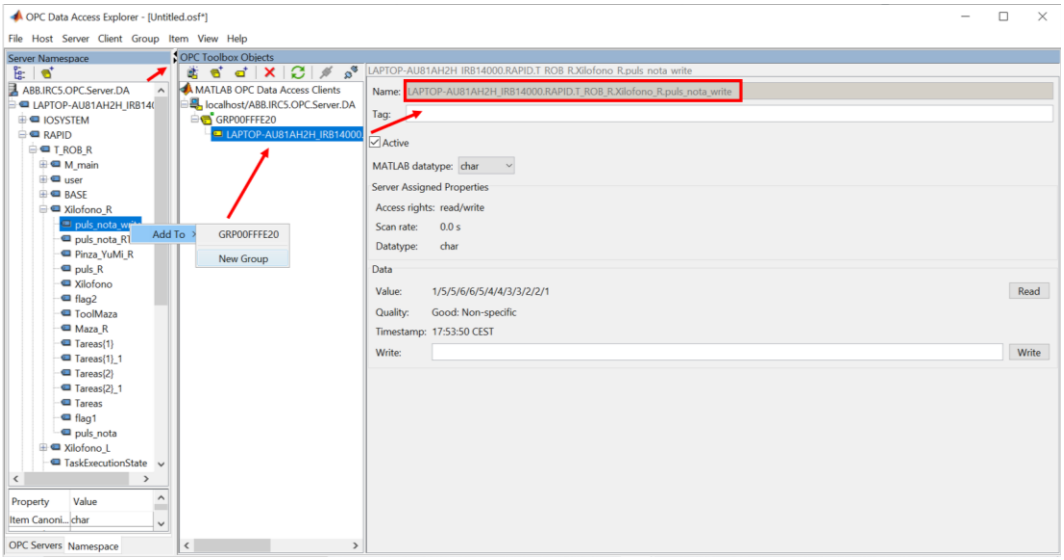


Figura 86. Identificador completo de una variable en el servidor OPC.

Una vez hecho esto, se programan los comandos suficientes en MATLAB para establecer la conexión, los cuales aparecerán dentro del HMI, y se activarán al seleccionar el botón correspondiente a la conexión. Para ello, se siguen los pasos recogidos en la Tabla 19.

Después ya se podrían utilizar las funciones de MATLAB “write” y “read” según se quiera escribir o leer.

Paso 1	Crear un objeto de cliente de acceso a datos de OPC. `app.da= opcda('localhost', 'ABB.IRC5.OPC.Server.DA');`
Paso 2	Conectarse al servidor de acceso a datos de OPC. `connect(app.da);`
Paso 3	Crear un grupo de objetos de acceso a datos OPC. `grp = addgroup(app.da);`
Paso 4	Explorar el espacio de nombres del servidor. `puls_input1=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_8']);`
Paso 5	Agregar elementos de acceso al grupo de datos OPC. `app.pulsItem_input1=additem(grp, puls_input1);`
Paso 6	Configurar las propiedades del grupo para el registro. `set(grp,'UpdateRate',0.001);`
Paso 7	Desconectar. `disconnect(app.da)`

Tabla 19. Pasos de comunicación OPC.

4.4. Manual de usuario del HMI

Para solicitar las notas de un modo más dinámico, se ha desarrollado una Interfaz Hombre-Máquina (HMI) con la que se puede controlar toda la simulación.

A continuación, se desarrolla el manual de la interfaz de usuario que se ha creado con la aplicación de MATLAB "AppDesigner".

4.4.1. Descripción

La presente interfaz de usuario se ha creado con el fin de poder solicitar las notas en tiempo real de una manera más sencilla, similar a la de tener un teclado donde poder pulsar las notas requeridas en cada momento, así como de poder escoger la acción que se quiera ejecutar, ya sea coger las mazas, leer las notas de un fichero que incluye el número de cada nota de una canción entera, o dejar las mazas en su posición inicial.

Para poder seleccionar cada una de las acciones que realiza el robot, es necesario que haya una comunicación entre la interfaz y el robot, por lo que existe un modo de conexión con el que se inicializa la comunicación, y un modo de desconexión, con el que finaliza tanto la comunicación como la simulación y se envía inmediatamente el robot a su posición de reposo, independientemente de lo que esté haciendo en ese momento, a modo de seguridad para el usuario.

4.4.2. Datos de entrada

Para poder utilizar esta interfaz de usuario al completo, es necesario que exista un fichero en formato TXT en la misma carpeta en la que se encuentra la interfaz. Dentro de este fichero están escritos el número correspondiente a cada nota de una canción, separadas por una barra. La interfaz se ha programado de tal manera que no pueda leer más de cuatro canciones, y cada secuencia de notas esté separada por una fila, en la que aparecerá el título de la canción escrita, como se puede ver en la Figura 87.

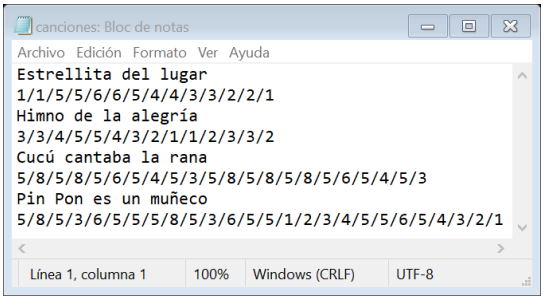


Figura 87. Archivo .txt con las notas de las canciones que se pueden solicitar.

4.4.3. Requerimientos

Para poder utilizar la interfaz de usuario es necesario contar con los módulos que se han programado para cada brazo del robot YuMi, así como de la estación y el controlador que se han creado para la realización de este trabajo en RobotStudio. Además, será necesario establecer la comunicación entre MATLAB y RobotStudio con el servidor creado a través de la aplicación "ABB IRC5 OPC Configuration" cuando el estado del controlador del robot esté en estado activo.

4.4.4. Presentación de cada herramienta

En la Figura 88 se muestra el resultado final de la ventana que aparece al simular la interfaz de usuario. En ella podemos ver todos los botones con los que cuenta esta aplicación.

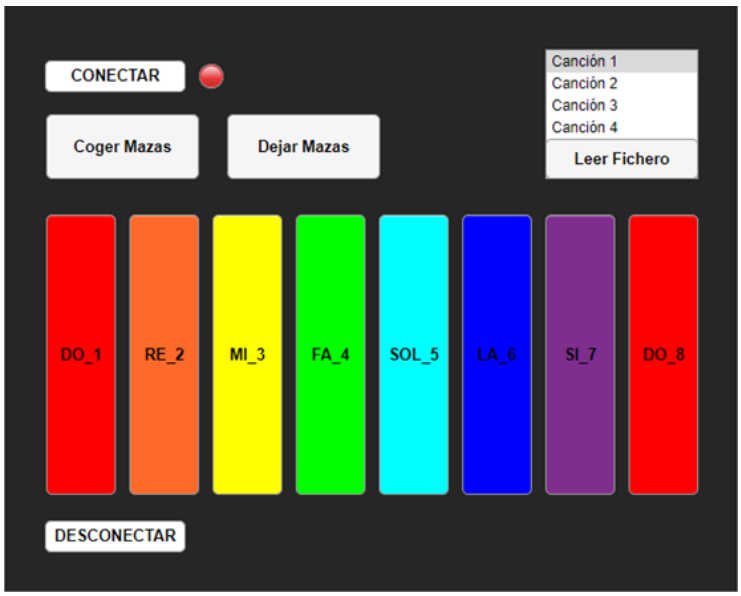


Figura 88. Interfaz de usuario (HMI).

En la Figura 89 aparecen todas las propiedades necesarias que se han definido en la interfaz. Estas propiedades pueden utilizarse en cualquier parte del código, actuando como variables globales del programa. Aquí se puede encontrar:

- El objeto de cliente de acceso a datos de OPC (da).
- La variable "stop", utilizada para parar la simulación.
- La lectura de datos de RobotStudio.
- Los diferentes elementos de acceso a datos OPC que se van a necesitar en las diferentes funciones del programa.

```
properties (Access = private)
    da
    stop

    pulsItem_input1
    pulsItem_input2
    pulsItem_input3
    pulsItem_input4
    pulsItem_input5

    notaItem_RT_R
    notaItem_RT_L
    ocupadoVal_R
    ocupadoVal_L

    notaItem_write_R
    notaItem_write_L
end
```

Figura 89. Propiedades HMI.

En la Figura 90 se muestra la función que genera el sonido de cada nota [26]. Se pasa como argumento el valor numérico de la nota que se quiere oír. Para distinguir las notas se define su frecuencia en Hercios y el período de la señal, y se almacenan en un vector. La posición en la que se encuentren las notas coincide con su valor numérico.

```
methods (Access = private)
    % Función que genera el sonido de las notas
    function results = sonido(app,nota)
        Fs=8000;
        Ts=1/Fs;
        t=[0:Ts:0.3];

        INICIO = 0;
        DO_1 = 261.626;
        RE_2 = 293.665;
        MI_3 = 329.628;
        FA_4 = 349.228;
        SOL_5 = 391.995;
        LA_6 = 440;
        SI_7 = 493.883;
        DO_8 = 523.251;

        vector_nota=[DO_1,RE_2,MI_3,FA_4,SOL_5,LA_6,SI_7,DO_8,INICIO];
        notes = [vector_nota(nota)];
        x = cos(2*pi*notes*t);
        sig = reshape(x',length(t),1);
        soundsc(sig,Fs)
    end
end
```

Figura 90. Función que genera el sonido de las notas.

A continuación, se explica la funcionalidad que tiene cada uno de los botones con los que cuenta esta interfaz.

Conectar

Este botón es el primero que debe ser pulsado para poder utilizar esta interfaz. Establece la comunicación con el robot e inicia la lectura de la variable que indica las notas que son golpeadas por el robot, la cual va a ser constante durante toda la simulación, ejecutándose en paralelo con el resto de funciones, las cuales actuarán como interrupciones. Por defecto, la luz que aparece al lado de este botón aparece de color rojo, pero cuando se pulsa este, luce de color verde, como se puede ver en la Figura 91, para saber en todo momento si estamos estableciendo una comunicación con el robot o no.

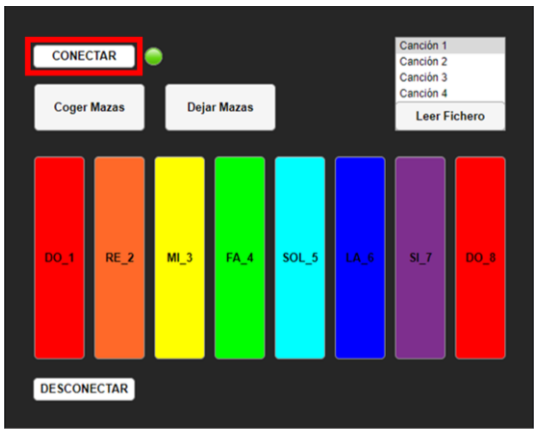


Figura 91. Conectar al servidor HMI.

En la Figura 92 se puede ver el código de la función propia de este botón, donde se siguen los pasos reflejados en la Tabla 19 para establecer una comunicación OPC.

```
        % Button pushed function: CONECTARButton
        function CONECTARButtonPushed(app, event)
            % Comenzar la comunicación creando todos los items que van a ser
            % utilizados y leer las notas que están siendo
            % pulsadas desde el robot
            app.da= opcda('localhost', 'ABB.IRC5.OPC.Server.DA');
            connect(app.da);
            grp = addgroup(app.da);
            set(grp,'UpdateRate',0.001);
            pc='LPSK_IRB14000.';
            app.Lamp.Color=[0.39,0.83,0.07];

            puls_input1=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_8']);
            puls_input2=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_9']);
            puls_input3=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_10']);
            puls_input4=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_11']);
            puls_input5=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_12']);

            % Leer
            nota_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.puls_nota']);

            % Pulsar teclas
            nota_RT_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.↙
puls_nota_RT']);
            nota_RT_L=serveritems(app.da,[pc,'RAPID.T_ROB_L.Xilofono_L.↙
puls_nota_RT']);
            ocupado_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.ocupado']);
            ocupado_L=serveritems(app.da,[pc,'RAPID.T_ROB_L.Xilofono_L.ocupado']);
            % Leer fichero
            nota_write_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.↙
puls_nota_write']);
            nota_write_L=serveritems(app.da,[pc,'RAPID.T_ROB_L.Xilofono_L.↙
puls_nota_write']);

            app.pulsItem_input1=additem(grp, puls_input1);
            app.pulsItem_input2=additem(grp, puls_input2);
            app.pulsItem_input3=additem(grp, puls_input3);
            app.pulsItem_input4=additem(grp, puls_input4);
            app.pulsItem_input5=additem(grp, puls_input5);

            notaItem_R=additem(grp, nota_R);

            app.notaItem_RT_R=additem(grp, nota_RT_R);
            app.notaItem_RT_L=additem(grp, nota_RT_L);
            ocupadoItem_R=additem(grp, ocupado_R);
            ocupadoItem_L=additem(grp, ocupado_L);

            app.notaItem_write_R= additem(grp, nota_write_R);
            app.notaItem_write_L= additem(grp, nota_write_L);

            app.stop=0;
            sonido(app,9)
            % Leer
            write(notaItem_R,0)
            registro=0;
            while app.stop==0
                app.ocupadoVal_R=read(ocupadoItem_R);
                app.ocupadoVal_L=read(ocupadoItem_L);
                notaVal_R = read(notaItem_R);
                if ((notaVal_R.Value)~=0) && registro==0
                    sonido(app,notaVal_R.Value)
                    registro=1;
                elseif notaVal_R.Value==0
                    registro=0;
                end
                pause(0.001)
            end
        end
```

Figura 92. Función del botón Conectar HMI.

Coger Mazas

El botón denominado “Coger Mazas”, el cual se puede ver recuadrado en rojo en la Figura 93, se pulsa para llamar al procedimiento en el que están

programados los movimientos del robot para poder coger las mazas cuando se encuentran en su posición inicial.

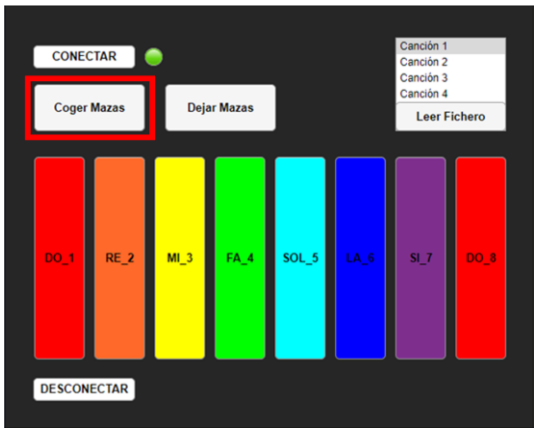


Figura 93. Coger Mazas HMI.

En la Figura 94 se puede ver el código de la función propia de este botón, donde se activa y desactiva la señal de entrada del controlador correspondiente a la primera opción que aparece en el procedimiento principal del programa.

```
% Button pushed function: CogerMazasButton
function CogerMazasButtonPushed(app, event)
    % Activar entrada del controlador corespondiente a coger las
    % mazas
    write(app.pulsItem_input1,'1')
    pause(0.1)
    write(app.pulsItem_input1,'0')
end
```

Figura 94. Función del botón CogerMazas HMI.

Pulsar notas en tiempo real

Para poder pulsar cualquier nota que se quiera solicitar en tiempo real, se han creado ocho botones, uno para cada tecla del xilófono, del mismo color que éstas y con el nombre de la nota a la que corresponden para semejar la apariencia del xilófono real, y para darle una apariencia mucho más visual y llamativa, como se puede ver en la Figura 95. Cada vez que se pulse un botón, el brazo del robot al que corresponda esa nota golpeará la lámina correspondiente. Si se pulsan notas entre *do* y *fa*, se moverá el brazo derecho, mientras que si se pulsan notas de *sol* a *do* se moverá el brazo izquierdo, por motivo de la cercanía a estas.

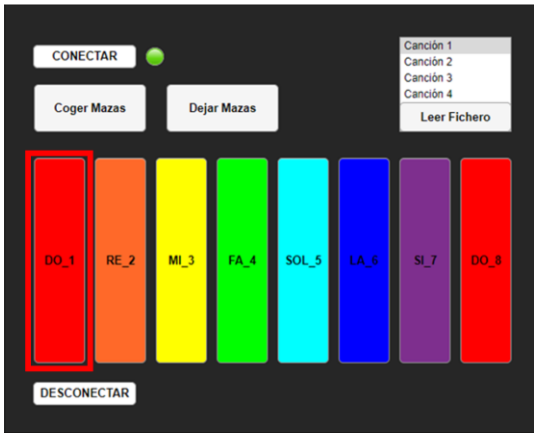


Figura 95. Pulsar notas en tiempo real HMI.

En la Figura 96 se puede ver el código de la función correspondiente al botón de la primera nota, *do*. Se ha programado de tal manera que solo solicite una nota si el brazo izquierdo está libre, es decir, que solo se active la señal del controlador correspondiente a la rutina TRAP que selecciona la segunda opción del procedimiento principal si el brazo izquierdo está libre. Después de 0,1 segundos, esta entrada se desactivará para poder volver a pulsarla a continuación si se desea. La variable definida en RobotStudio para las notas solicitadas en tiempo real se escribe siempre con el valor correspondiente de la nota seleccionada.

```
        % Button pushed function: DO_1Button
        function DO_1ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_R,1)
            if app.ocupadoVal_L.Value==0 % solo solicita esta nota si el brazo↙
izquierdo no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end
```

Figura 96. Función del botón DO_1 HMI.

Leer Fichero

Para poder leer las notas del fichero, primero se debe seleccionar la canción deseada de entre las cuatro posibles en el menú, si es que se quiere que el robot toque otra canción diferente a la seleccionada por defecto, y después pulsar el botón en el que pone “Leer Fichero”. Tanto el menú como el botón aparecen encuadrados en rojo en la Figura 97.

Figura 97. Escoger canción y Leer Fichero HMI.

La Figura 98 muestra el código de la función correspondiente a este botón, donde se puede ver la llamada al valor del menú de la lista de canciones seleccionada, y en función de éste, se leerá una fila u otra del archivo TXT. Una vez leído, se escribe sobre la variable definida en RobotStudio la cadena de caracteres correspondiente al número de notas de la canción seleccionada, y se activa la entrada del controlador que llama al procedimiento para golpear las notas leídas del fichero.

```
% Button pushed function: LeerFicheroButton
function LeerFicheroButtonPushed(app, event)
    % Leer del fichero la canción seleccionada en el menú
    menu= app.ListBox.Value;

    if strcmp(menu, 'Canción 1')
        cancion = 2;
    elseif strcmp(menu, 'Canción 2')
        cancion = 4;
    elseif strcmp(menu, 'Canción 3')
        cancion = 6;
    elseif strcmp(menu, 'Canción 4')
        cancion = 8;
    end

    % Escribir canción
    A = importdata('canciones.txt');
    write(app.notaItem_write_R,A(cancion))
    write(app.notaItem_write_L,A(cancion))

    write(app.pulsItem_input3,'1')
    pause(0.1)

    write(app.pulsItem_input3,'0')
end
```

Figura 98. Función del botón Leer Fichero HMI.

Dejar Mazas

El botón correspondiente a dejar las mazas se puede ver recuadrado en rojo en la Figura 99, el cual se pulsa para llamar al procedimiento en el que están programados los movimientos del robot para poder dejar las mazas en su posición inicial.

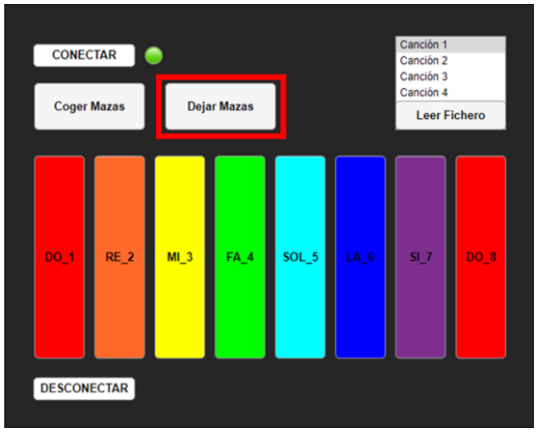


Figura 99. Dejar Mazas HMI.

En la Figura 100 se muestra el código de la función propia de este botón, donde se activa y desactiva la señal de entrada del controlador correspondiente a la cuarta opción que aparece en el procedimiento principal del programa.

```
% Button pushed function: DejarMazasButton
function DejarMazasButtonPushed(app, event)
    % Activar entrada del controlador corespondiente a dejar las mazas
    write(app.pulsItem_input4,'1')
    pause(0.1)
    write(app.pulsItem_input4,'0')
end
```

Figura 100. Función del botón Dejar Mazas HMI.

Desconectar

El botón de desconectar, el cual se puede ver en la Figura 101, es el último que debe ser pulsado para poder finalizar la simulación y desconectar la comunicación con el robot. Al ser pulsado, la luz volverá a ser de color rojo, indicando que ya no existe comunicación.

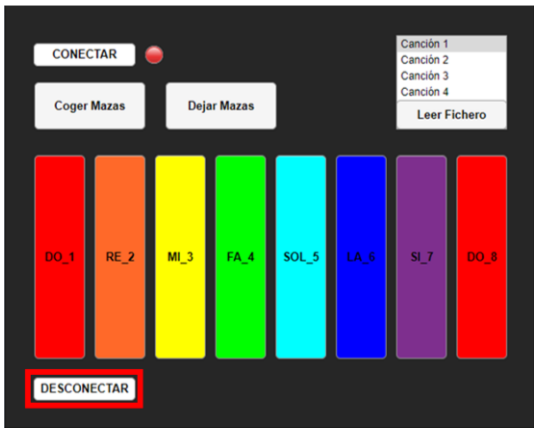


Figura 101. Desconectar al servidor HMI.

La Figura 102 muestra el código de la función correspondiente al botón "DESCONECTAR", donde se puede ver que se iguala la variable "stop" a uno para finalizar la lectura de las notas que son golpeadas, se activa la entrada del controlador correspondiente a la rutina TRAP que contiene la última opción del procedimiento principal para finalizar la simulación, y se desconecta la variable denominada "da", utilizada para crear un objeto de cliente de acceso a datos de OPC.

```
    % Button pushed function: DESCONECTARButton
    function DESCONECTARButtonPushed(app, event)
        % Dejar de leer las notas y finalizar la comunicación
        app.stop=1;
        write(app.pulsItem_input5,'1')
        pause(0.1)
        write(app.pulsItem_input5,'0')
        disconnect(app.da)
        app.Lamp.Color=[0.90,0.21,0.21];
    end
end
```

Figura 102. Función del botón Desconectar HMI.

4.4.5. Paso a paso

A continuación, se enumeran los pasos a seguir a la hora de utilizar esta interfaz de usuario, una vez se ha iniciado el servidor con la aplicación "ABB IRC5 OPC Configuration", y el controlador del robot IRB14000 está en estado activo.

PASO 1: Iniciar la simulación del robot desde RobotStudio.

PASO 2: Iniciar la simulación de la interfaz de usuario desde MATLAB.

PASO 3: Pulsar el botón "CONECTAR" para iniciar la comunicación.

PASO 4: Pulsar el botón "Coger Mazas" para que el robot coja las mazas situadas sobre el soporte de mazas, en la posición inicial.

PASO 5: Dependiendo de si se quiere que el robot toque una canción de las incluidas en el menú, o si se quiere que toque las notas que se van solicitando en tiempo real, se seguirá uno de los siguientes pasos:

- **PASO 5.1:** Para leer las notas del fichero:
  - **PASO 5.1.1:** Seleccionar una de las canciones que aparecen en el menú.
  - **PASO 5.1.2:** Pulsar el botón "Leer Fichero" para que el robot comience a tocar la canción seleccionada.

- **PASO 5.2:** Pulsar cualquiera de los botones denominados con el nombre de la nota a la que se corresponde ese botón, en función de la nota que se quiera que toque el robot.

PASO 6: Pulsar el botón "Dejar Mazas" para que el robot deje las mazas sobre el soporte de mazas, en la posición inicial.

PASO 7: Pulsar el botón "DESCONECTAR" para que el robot vuelva a su posición base y se finalice la comunicación.

CAPÍTULO 5

Resultados

Tras la explicación del desarrollo de este proyecto y la exposición de cómo se ha llevado a cabo, en este capítulo se presenta el resultado del trabajo realizado, y la ejecución de los programas, con algunas de las capturas donde se pueden ver los movimientos del robot, obtenidas durante la ejecución del programa, mostrando tanto la simulación del robot como la interfaz de usuario.

El programa comienza en un escenario en el que las mazas se encuentran en su soporte. Primero se pulsa el botón "CONECTAR" de la interfaz y se inicia la comunicación entre MATLAB y RobotStudio.

A continuación, se pulsa el botón "Coger Mazas", y el robot comienza a moverse. Primero el brazo derecho coge su maza correspondiente, y cuando llega a un punto determinado de la tarea, el brazo izquierdo comienza a moverse para coger la otra maza sin que se produzcan colisiones entre los dos brazos. A la hora de coger la maza se realizan cinco movimientos:

- El primero, sin seguir ninguna trayectoria determinada, llega a un punto cien milímetros por encima del extremo de la maza.
- Después, para retirar la maza del soporte, realiza tres movimientos lineales:
    - uno para bajar a una posición donde alcance la maza y poder cerrar la pinza,
    - y otros dos para sacarla del soporte.

Seguidamente, vuelve a su posición de reposo con la maza agarrada por la herramienta.

En la Figura 103 se pueden ver capturas de la simulación donde el robot realiza algunos de estos movimientos.

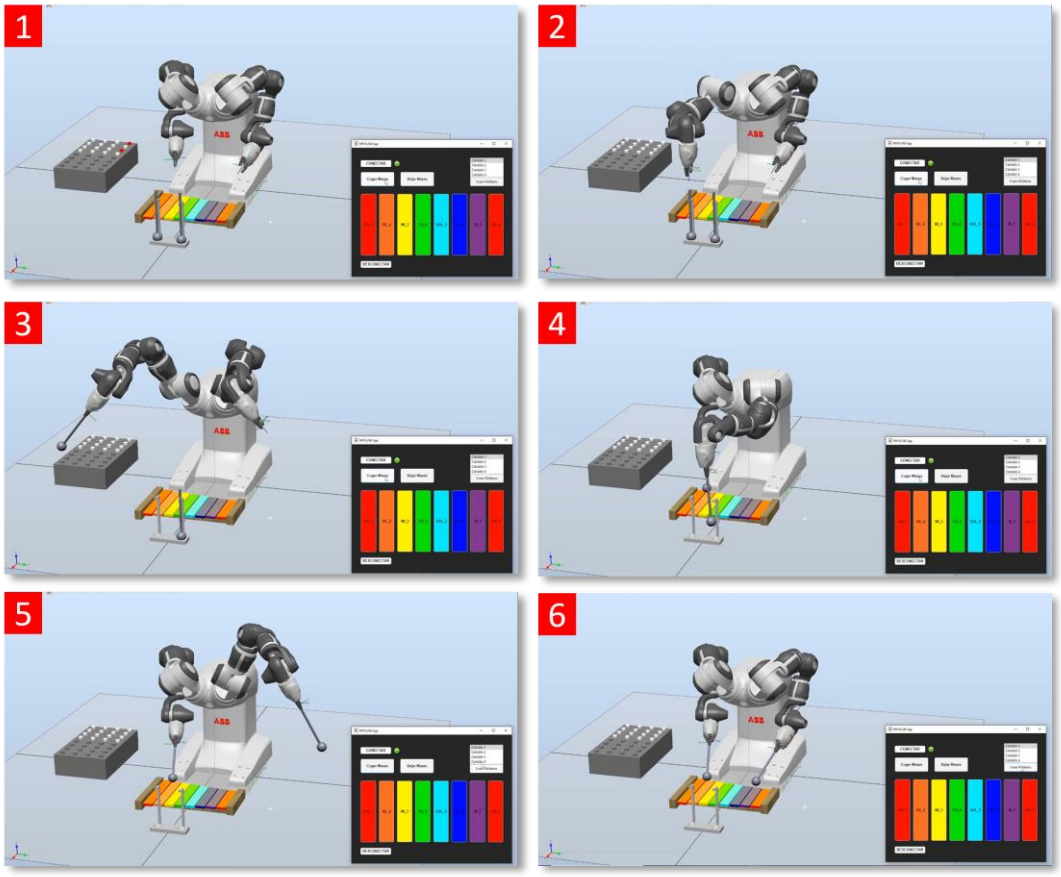


Figura 103. Simulación CogerMazas.

En la Figura 104 se pueden ver capturas del movimiento del robot cuando se selecciona la primera canción del menú, y se pulsa desde la interfaz de usuario el botón “Leer Fichero”. En esta opción del programa, los brazos del robot se van alternando de manera sincronizada para que no se produzcan colisiones ni se golpeen dos láminas a la vez. Cuando una maza golpea una lámina, el sensor que se encuentra en la parte superior de ésta se activa, haciendo que la tecla cambie de color. Para que el robot llegue a pulsar una tecla, éste realiza tres movimientos lineales:

- uno para llegar a un punto veinte milímetros por encima de la lámina que se desea pulsar,
- otro para buscar el momento en el que se activa el sensor, lo que significará que ya se ha golpeado la lámina,
- y otro para volver al punto que está veinte milímetros por encima de la lámina golpeada.

Después de esto volverá a una posición que se utilizará como base a la hora de golpear las láminas del xilófono.

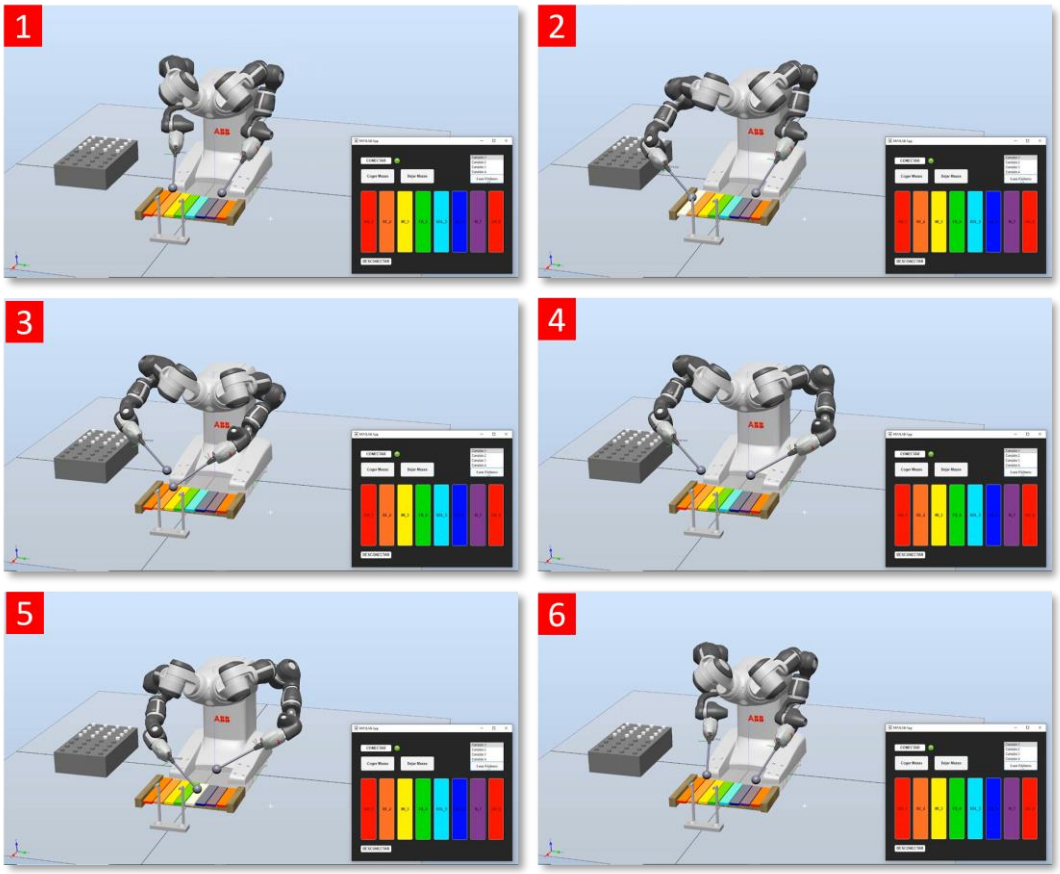


Figura 104. Simulación LeerFichero.

En la Figura 105 se recoge una serie de capturas de la simulación cuando se pulsa alguno de los botones multicolores de la interfaz, los cuales representan cada una de las láminas del xilófono. El movimiento del robot será el mismo que en la Figura 104, con la diferencia de que ahora las láminas más cercanas al brazo derecho serán golpeadas por éste, y las más cercanas al brazo izquierdo las golpeará el brazo izquierdo, según se van solicitando las notas en tiempo real.

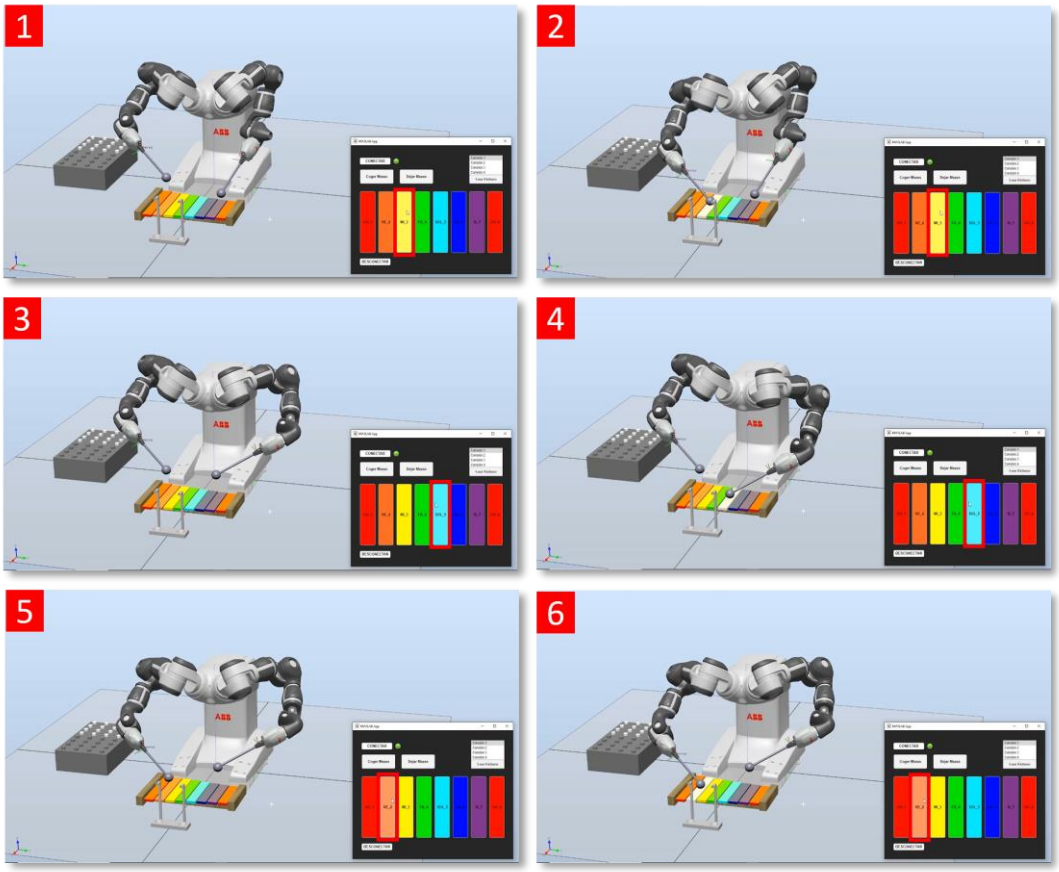


Figura 105. Simulación Pulsar notas en tiempo real.

Por último, para volver a dejar las mazas en su sitio, se pulsa el botón “Dejar Mazas” y el robot realiza movimientos análogos a cuando coge las mazas, pero en sentido contrario, como se puede ver en la Figura 106.

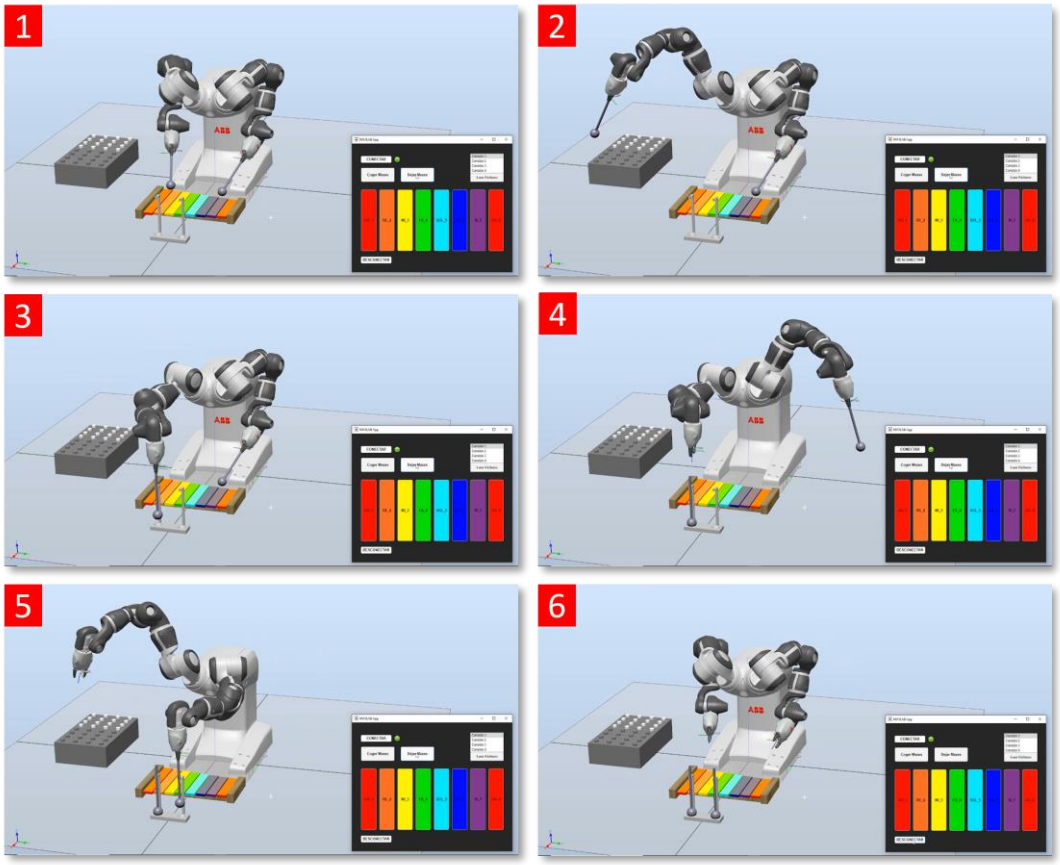


Figura 106. Simulación DejarMazas.

CAPÍTULO 6

Conclusiones y líneas futuras

En este capítulo se exponen las conclusiones finales del trabajo realizado, incluyendo un análisis del grado de cumplimiento de los objetivos. Finalmente, se plantean futuras líneas de trabajo destinadas a llevar la simulación a la vida real o a ampliar el rango de posibilidades del programa.

6.1. Conclusiones

Analizando los resultados alcanzados, se podría decir que se han cumplido los objetivos propuestos de manera satisfactoria, ya que el robot realiza los movimientos requeridos para golpear las láminas del xilófono, y se establece correctamente la comunicación bidireccional con la interfaz de usuario, haciendo que el programa sea más dinámico e interactivo, y se genere el sonido de las notas en el momento en el que la lámina es golpeada.

En la realización de este proyecto se han desarrollado conocimientos en diferentes campos de la ingeniería, cumpliendo de esta manera uno de los objetivos requeridos en este trabajo.

Además, se ha conseguido darle a la robótica una aplicación artística, creativa e innovadora.

En definitiva, las conclusiones alcanzadas tras la finalización del proyecto son las que se enumeran a continuación:

1. Se creó una estación en el entorno de simulación RobotStudio, incluyendo todos los componentes necesarios para que el robot YuMi fuese capaz de tocar un xilófono. Estos componentes son los siguientes:
   - Piezas:
     - Mazas
     - Soporte para mazas
   - Componentes inteligentes:
     - Pinza YuMi, capaz de coger cualquier objeto.
     - Xilófono, incluyendo sensores planos en cada una de sus láminas, las cuales cambian de color en el momento en el que son golpeadas.
     - Botonera, con dieciséis botones capaces de activar las dieciséis entradas del controlador, y dieciséis bombillas

capaces de cambiar de color cuando se activa alguna señal de salida del controlador, simulando que las bombillas lucen.

2. Se programó en lenguaje RAPID el código necesario para definir los movimientos e instrucciones a realizar por el robot. En este código se incluyó el siguiente menú de acciones que puede realizar el robot:

   - Opción 1: Coger Mazas.
   - Opción 2: Golpear las notas que van siendo solicitadas en tiempo real.
   - Opción 3: Golpear las notas que se han leído previamente de un fichero.
   - Opción 4: Dejar Mazas.
   - Opción 5: Finalizar programa.

3. Se estableció una comunicación entre RobotStudio y MATLAB utilizando el protocolo OPC UA, con el fin de leer y escribir desde MATLAB variables declaradas en RobotStudio.

4. Se desarrolló con MATLAB una Interfaz Hombre-Máquina con la que se controla toda la simulación de un modo más dinámico.

5. Se implementaron conocimientos en diferentes campos de la ingeniería, como son la robótica y las comunicaciones industriales.

6. Se consiguió emplear la robótica en una aplicación artística, creativa e innovadora.

6.2. Líneas futuras

En cuanto a las líneas futuras de trabajo, se pueden identificar importantes alternativas de programación que supondrían una ampliación de las funcionalidades actuales del programa, ya que este trabajo es un primer intento de programar un robot con el fin de tocar un xilófono. Estas alternativas son las siguientes:

- Realizar la simulación con un robot de verdad, conectando el xilófono con la tarjeta de señales del robot, o utilizando un xilófono independiente a las entradas del controlador del robot.

  Esto no se ha podido llevar a cabo debido a que, actualmente, la Universidad de Valladolid no cuenta con un robot colaborativo YuMi, pero se pretende comprar uno para poder trabajar con él, siendo este proyecto un anticipo para ese futuro robot colaborativo.

Las ventajas que se encontrarían al usar un robot colaborativo serían las siguientes:

- Se puede posicionar de forma manual al punto deseado, sin necesidad de llevarle únicamente a un punto con instrucciones RAPID a través de RobotStudio. Esto puede ser muy útil a la hora de definir las coordenadas de los puntos de la estación, ya que se pueden conseguir los valores buscados llevando al robot directamente al punto objetivo.

- Es más seguro. Los robots colaborativos están equipados con sensores para reducir su velocidad o detenerse en caso de entrar en contacto con alguien. Si se mueve el robot con la mano, se puede alejar del usuario en caso de que haya algún fallo en el programa o una persona se encuentre demasiado cerca de él, a diferencia del robot actual con el que cuenta la Universidad, el IRB120, que no puede ser utilizado por un alumno sin supervisión por seguridad.

- Para el caso del presente trabajo, suponiendo que la sensibilidad del robot fuese suficiente para detectar el golpe con la tecla, el robot no tendría que haber recibido la señal del xilófono, ya que un robot colaborativo puede detectar que está golpeando algo. Sin embargo, en simulación un sensor no detecta esfuerzos. Se pueden leer esfuerzos sobre el robot real, pero no sobre el robot simulado.

Además, estas ventajas de usar un robot colaborativo podrían sacarse partido para otros muchos proyectos como, por ejemplo, uno que tenga como objetivo programar un robot colaborativo que diese de comer a una persona mayor, con la seguridad de que el robot va a actuar en función de la posición en la que está situada esa persona, y se pueda alejar el robot con la mano en caso de que haya algún fallo en el programa, o se quiera cambiar de posición.

- Colocar el robot en una cinta o transportador para que su área de trabajo sea mayor, y así pueda llegar a tocar un xilófono mucho más grande y con mayor número de escalas.

- Otra posibilidad de enfocar este trabajo, aunque de una manera mucho más ambiciosa, podría ser desarrollar un algoritmo de inteligencia artificial que cuente con una base de datos con numerosas canciones para que el robot sea capaz de empezar su propia composición musical a partir de una primera línea rítmica.

Bibliografía

[1] S.-M. FM, «Historia de la robótica: de Arquitas de Tarento al Robot da Vinci.,» 03 2007. [En línea]. Available: https://scielo.isciii.es/pdf/aue/v31n3/v31n3a02.pdf. [Último acceso: 05 07 2022].

[2] S. B. Niku, An introduction to robotics: analysis, control, applications. (2nd edition), John Wiley & Sons Inc., 2010.

[3] A. Barrientos, L. F. Peñin, C. Balaguer y R. A. Santoja, Fundamentos de Robótica (2º edición), MacGraw-Hill, 2007.

[4] P. Corke, Robotics, Vision and Control, Fundamental Algorithms in MATLAB., Springer Tract in Avance Robotic., 2013.

[5] c. ETI, «Robótica.,» [En línea]. Available: http://www.etitudela.com/profesores/rpm/rpm/downloads/robotica.pdf. [Último acceso: 06 07 2022].

[6] Femeval, «Robots Industriales y Cobots en prevención de riesgos laborales.,» [En línea]. Available: https://www.femeval.es/dam/jcr:fd091e5f-3c97-42fd-9981-680e8232a645/GUIA_ROBOTS.pdf. [Último acceso: 06 07 2022].

[7] Centro tecnológico de Automoción de Galicia, «CTAG. La industria del Futuro.,» [En línea]. Available: http://issga.xunta.gal/export/sites/default/recursos/descargas/documentacion/material-formativo/relatorios/2017_05_CO_Industria_4.0_Alonso.pdf. [Último acceso: 06 07 2022].

[8] ABB, «ABB data sheet. YuMi, un futuro automatizado juntos.,» 2015. [En línea]. Available: https://library.e.abb.com/public/0236bf1e2983477db86fe424e45b1134/Data%20sheet%20ES.pdf. [Último acceso: 22 02 2022].

[9] G. M. Garrido, «Trabajos Fin de Grado UVa. Simulación de una célula robotizada para la fabricación de piezas mecanizadas en un entorno industrial, utilizando RobotStudio.,» 2015. [En línea]. Available:

https://uvadoc.uva.es/handle/10324/13173. [Último acceso: 26 02 2022].

[10] J. A. Á. Herrero, «Trabajos Fin de Grado UVa. Modelado de una célula robótica con fines educativos usando el programa Robot-Studio.,» Septiembre 2015. [En línea]. Available: https://uvadoc.uva.es/handle/10324/14441. [Último acceso: 01 03 2022].

[11] C. J. Jiménez, «Trabajos Fin de Grado UVa. Diseño de un sistema robótico educativo para jugar al ajedrez con robots industriales.,» 01 07 2019. [En línea]. Available: https://uvadoc.uva.es/handle/10324/36877. [Último acceso: 26 02 2022].

[12] V. L. Granado, «Trabajos Fin de Máster UVa. Diseño, simulación y fabricación de una estación robotizada universitaria.,» 07 2020. [En línea]. Available: http://uvadoc.uva.es/handle/10324/41294. [Último acceso: 26 02 2022].

[13] R. S. García, «Trabajos Fin de Grado UVa.,» 15 06 2021. [En línea]. Available: https://uvadoc.uva.es/handle/10324/47254. [Último acceso: 26 02 2022].

[14] ABB, «ROBOTSTUDIO. Descarga RobotStudio con RobotWare y PowerPacs.,» 2022. [En línea]. Available: https://new.abb.com/products/robotics/es/robotstudio/descargas. [Último acceso: 01 07 2022].

[15] «Monografías. Introducción a la utilización de RobotStudio ABB.,» 2015. [En línea]. Available: https://www.monografias.com/trabajos102/introduccion-utilizacion-robotstudio-abb/introduccion-utilizacion-robotstudio-abb. [Último acceso: 02 07 2022].

[16] MathWorks, «MATLAB. Matemáticas. Gráficas. Programación.,» 2022. [En línea]. [Último acceso: 04 07 2022].

[17] infoPLC, «ABB Robot IRC5 OPC UA Server.,» 08 07 2021. [En línea]. Available: https://www.infoplc.net/descargas/46-abb-robotica/3288-abb-robot-irc5-opc-ua-server. [Último acceso: 04 07 2022].

[18] SIEMENS, «De camino a la digitalización, gracias a la comunicación abierta y neutral de plataforma con OPC UA.,» 2022. [En línea]. Available:

https://new.siemens.com/pe/es/products/automatizacion/comunicaciones-industriales/opc-ua.html. [Último acceso: 04 07 2022].

[19] atvise, «¿Qué Es OPC UA? Descubre La Arquitectura Unificada.,» 21 01 2021. [En línea]. Available: https://atvise.vesterbusiness.com/news/que-es-opc-ua-arquitectura-unificada/. [Último acceso: 05 07 2022].

[20] ABB, «Manual del operador RobotStudio 5.13. ID de documento: 3HAC032104-005 Revisión: C,» 2010.

[21] ABB, «Manual de referencia técnica. Instrucciones, funciones y tipos de datos de RAPID. RobotWare 5.13. ID de documento: 3HAC16581-5. Revisión: J.,» 2010.

[22] ABB, «ROBOTICS. Application Manual. MultiMove. RobotWare 6.13. Document ID: 3HAC050961-001. Revision: G.,» 2021.

[23] MathWorks, «OPC UA,» [En línea]. Available: https://es.mathworks.com/discovery/opc-ua.html. [Último acceso: 29 06 2022].

[24] M. Rodríguez, «INESEM,» 22 02 2016. [En línea]. Available: https://www.inesem.es/revistadigital/gestion-integrada/protocolo-opc-ua-caracteristicas-y-aplicaciones/. [Último acceso: 26 06 2022].

[25] MathWorks, «OPC Toolbox. User's Guide. MATLAB & SIMULINK. R2021a.,» 2021.

[26] MathWorks, «MATLAB Answers,» 21 09 2012. [En línea]. Available: https://es.mathworks.com/matlabcentral/answers/48717-create-a-piece-of-music-using-matlab. [Último acceso: 09 06 2022].

ANEXO A

En este anexo se incluye el código desarrollado para la programación del robot en lenguaje RAPID.

```
MODULE Xilofono_R
    CONST jointtarget Inicio:=[[0,-130,30,0,40,0],[-135,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PERS tooldata Pinza_YuMi_R:=[TRUE,[[0,0,114.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    PERS tooldata ToolMaza:=[TRUE,[[0,0,314.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

    PERS wobjdata Maza_R:=[FALSE,TRUE,"",[[600,-75,15],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    PERS wobjdata Xilofono:=[FALSE,TRUE,"",[[200,-174,30],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget Punto:=[[89.387953825,-156.622113832,160],
[0.066010723,-0.842420916,-0.111214903,-0.523068666],[0,0,0,4],[-101.964426653,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget Target_10_MR:=[[25,25,210],[0,0,1,0],[0,0,0,0],[-101.964427132,9E+09,9E+09,9E
+09,9E+09,9E+09]];
    CONST robtarget Target_Nota:=[[132.5,-24,19.5],
[0.320796829,-0.781743202,-0.266041626,-0.463884484],[1,-1,1,4],[-111.912014269,9E+09,9E+09,9E+09,9E
+09,9E+09]];

    ! Variables de interrupción
    VAR intnum Idi0;
    VAR intnum Idi1;
    VAR intnum Idi2;
    VAR intnum Idi3;
    VAR intnum Idi4;
    VAR intnum Idi5;
    VAR intnum Idi6;
    VAR intnum Idi7;
    VAR intnum Idi8;
    VAR intnum Idi9;
    VAR intnum Idi10;
    VAR intnum Idi11;
    VAR intnum Idi12;

    ! VARIABLES Tasks y syncident (Sincronización)
    PERS tasks Tareas{2}:=[["T_ROB_L"],["T_ROB_R"]];
    VAR syncident sync1;
    VAR syncident sync2;
    VAR syncident sync3;
    VAR syncident sync4;

    ! Distancia entre teclas
    CONST num Tecla:=44;

    ! Variables de interrupción
    PERS bool nosalir;
    VAR num opcion;

    ! Variables escritas desde MATLAB
    PERS string puls_nota_write:="3/1/6/8/6/5"; ! Notas leídas del fichero
    PERS num puls_nota_RT:=1; ! Nota pulsada en tiempo real
```

```
! Variables leídas por MATLAB
PERS num ocupado:=0; ! Indica si el brazo izquierdo está ocupado o no
PERS num puls_nota:=0; ! Nota leída en todo momento por MATLAB

PERS bool puls_R:=FALSE; ! Variable a supervisar por SearchL
VAR num paso:=1;
VAR robtarget sp;
VAR string puls_nota_wr{100}; ! Notas leídas del fichero en string
VAR num puls_nota_num{100}; ! Notas leídas del fichero en num
CONST string notas{8}:=["1","2","3","4","5","6","7","8"];
VAR num orden:=0;
VAR num l:=1;

PROC mainXilofono_R()
    VAR num i:=1;
    VAR num n:=0;
    VAR num rango{4}:=[1,2,3,4];

    First;

    nosalir:=TRUE;
    opcion:=0;

    WHILE nosalir DO
        TEST opcion
        CASE 1:
            opcion:=0;
            SyncCogeMaza;
        CASE 2:
            ocupado:=0;
            opcion:=0;
            puls_nota:=0;
            FOR i FROM 1 TO 4 DO
                IF puls_nota_RT=rango{i} THEN
                    PulsarTeclas(puls_nota_RT);
                ENDIF
            ENDFOR
        CASE 3:
            opcion:=0;
            puls_nota:=0;
            WaitTime 1.5;
            l:=1;
            paso:=1;
            i:=1;
            n:=0;
            Escritura;
            WHILE i < Dim(puls_nota_num,1) AND puls_nota_num{i}<>0 DO
                IF puls_nota_num{i}<>0 THEN
                    n:=puls_nota_num{i};
                    SyncPulsarTeclas n;
                    i:=i+2;
```

```
                ENDIF
            ENDWHILE
            puls_nota:=0;
            MoveJ Punto,v100,z0,Pinza_YuMi_R;
        CASE 4:
            opcion:=0;
            SyncDejaMaza;
        CASE 5:
            opcion:=0;
            nosalir:=FALSE;
        DEFAULT:
        ENDTEST
    ENDWHILE

    TPWrite "FIN DEL PROGRAMA R";
    Stop;
ENDPROC

LOCAL PROC First()
    puls_nota:=0;

    ! Cancelamos las interrupciones
    IDelete Idi0;
    IDelete Idi1;
    IDelete Idi2;
    IDelete Idi3;
    IDelete Idi4;
    IDelete Idi5;
    IDelete Idi6;
    IDelete Idi7;
    IDelete Idi8;
    IDelete Idi9;
    IDelete Idi10;
    IDelete Idi11;
    IDelete Idi12;

    ! Conectar var intnum con trap
    CONNECT Idi0 WITH trap_1_DO;
    CONNECT Idi1 WITH trap_2_RE;
    CONNECT Idi2 WITH trap_3_MI;
    CONNECT Idi3 WITH trap_4_FA;
    CONNECT Idi4 WITH trap_5_SOL;
    CONNECT Idi5 WITH trap_6_LA;
    CONNECT Idi6 WITH trap_7_SI;
    CONNECT Idi7 WITH trap_8_DO;
    CONNECT Idi8 WITH trap_CogeMaza;
    CONNECT Idi9 WITH trap_TiempoReal;
    CONNECT Idi10 WITH trap_Fichero;
    CONNECT Idi11 WITH trap_DejaMaza;
    CONNECT Idi12 WITH trap_Fin;
```

```
    ISignalDI DI_0, 1, Idi0;
    ISignalDI DI_1, 1, Idi1;
    ISignalDI DI_2, 1, Idi2;
    ISignalDI DI_3, 1, Idi3;
    ISignalDI DI_4, 1, Idi4;
    ISignalDI DI_5, 1, Idi5;
    ISignalDI DI_6, 1, Idi6;
    ISignalDI DI_7, 1, Idi7;
    ISignalDI DI_8, 1, Idi8;
    ISignalDI DI_9, 1, Idi9;
    ISignalDI DI_10, 1, Idi10;
    ISignalDI DI_11, 1, Idi11;
    ISignalDI DI_12, 1, Idi12;

    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;

ENDPROC

LOCAL TRAP trap_1_DO
    puls_nota:=1;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_2_RE
    puls_nota:=2;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_3_MI
    puls_nota:=3;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_4_FA
    puls_nota:=4;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_5_SOL
    puls_nota:=5;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_6_LA
    puls_nota:=6;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_7_SI
    puls_nota:=7;
    puls_R:=TRUE;
```

```
ENDTRAP

LOCAL TRAP trap_8_DO
    puls_nota:=8;
    puls_R:=TRUE;
ENDTRAP

LOCAL TRAP trap_CogeMaza
    opcion:=1;
ENDTRAP

LOCAL TRAP trap_TiempoReal
    opcion:=2;
ENDTRAP

LOCAL TRAP trap_Fichero
    opcion:=3;
ENDTRAP

LOCAL TRAP trap_DejaMaza
    opcion:=4;
ENDTRAP

LOCAL TRAP trap_Fin
    opcion:=5;
    StopMove;
    StorePath;
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;
    RestoPath;
ENDTRAP

! Crear vector de números con las notas que se leen del fichero
LOCAL PROC Escritura()
    VAR string frag;
    VAR string puls_nota_w{100};
    VAR num j;
    VAR num h;
    VAR num k;
    ! Se llena un vector con únicamente los números de cada nota
    FOR j FROM 1 TO StrLen(puls_nota_write) DO
        frag:= StrPart(puls_nota_write,j,1);
        puls_nota_w{j}:=frag;
        IF puls_nota_w{j}<>"/" THEN
            puls_nota_wr{j}:=puls_nota_w{j};
        ENDIF
    ENDFOR
    ! Se pasa el vector de las notas de string a num
    FOR h FROM 1 TO Dim(puls_nota_w,1) DO
        FOR k FROM 1 TO Dim(notas,1) DO
            IF puls_nota_w{h}=notas{k} THEN
                puls_nota_num{l}:=k;
```

```
                l:=l+1; ! Devuelve el total de notas solicitadas
            ENDIF
        ENDFOR
    ENDFOR
    ! Se calcula el orden del número total de notas, par o impar
    IF ((l-1) MOD 2) <>0 THEN
        orden:=1;
    ENDIF
ENDPROC

! Pulsar teclas en tiempo real
LOCAL PROC PulsarTeclas(num notas)
    ocupado:=1;
    MoveL Offs(Target_Nota,0,2*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    puls_R:=FALSE;
    SearchL \Stop, puls_R, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    ocupado:=0;
    MoveL Offs(Target_Nota,0,2*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
ENDPROC

! Pulsar teclas leyendo las notas de un fichero
LOCAL PROC SyncPulsarTeclas(num notas)
    IF paso<>1 THEN
        WaitSyncTask sync3,Tareas;
        puls_nota:=0;
        WaitTime 1;
    ENDIF

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v150,z0,ToolMaza\WObj:=Xilofono;

    puls_R:=FALSE;
    SearchL \Stop, puls_R, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    IF ( orden=1 AND paso=(l/2)) THEN
        MoveL Offs(Target_Nota,0,2*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
    ELSE
        MoveL Offs(Target_Nota,0,2*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
        puls_nota:=0;
        WaitSyncTask sync2,Tareas;
    ENDIF
    paso:=paso+1;
ENDPROC

! Coger maza derecha
```

```
    LOCAL PROC SyncCogeMaza()
        MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;

        AbrirPinza;

        MoveJ Offs(Target_10_MR,0,0,100),v300,z0,Pinza_YuMi_R\WObj:=Maza_R;
        MoveL Offs(Target_10_MR,0,0,0),v50,fine,Pinza_YuMi_R\WObj:=Maza_R;

        CerrarPinza;

        WaitSyncTask sync1,Tareas;

        MoveL Offs(Target_10_MR,0,0,50),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
        MoveL Offs(Target_10_MR,10,0,50),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
        MoveJ Punto,v300,fine,Pinza_YuMi_R;
    ENDPROC

    ! Dejar maza derecha
    LOCAL PROC SyncDejaMaza()
        MoveJ Punto,v100,z0,Pinza_YuMi_R;
        MoveJ Offs(Target_10_MR,10,0,50),v300,z0,Pinza_YuMi_R\WObj:=Maza_R;
        MoveL Offs(Target_10_MR,0,0,50),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
        MoveL Offs(Target_10_MR,0,0,0),v50,fine,Pinza_YuMi_R\WObj:=Maza_R;

        AbrirPinza;

        WaitSyncTask sync4,Tareas;

        MoveL Offs(Target_10_MR,0,0,100),v50,z0,Pinza_YuMi_R\WObj:=Maza_R;
        MoveAbsJ Inicio,v300,fine,Pinza_YuMi_R;
    ENDPROC

    ! Abre la pinza
    LOCAL PROC AbrirPinza()
        Set DO_14; ! Abrir y desattacher
        WaitTime 2;
        Reset DO_14;
    ENDPROC

    ! Cerrar Pinza
    LOCAL PROC CerrarPinza()
        Set DO_15;  ! Activar los sensores de attacher
        WaitTime 2;
        Reset DO_15;
    ENDPROC
ENDMODULE
```

```
MODULE Xilofono_L
    CONST jointtarget Inicio:=[[0,-130,30,0,40,0],[135,9E+09,9E+09,9E+09,9E+09,9E+09]];

    PERS tooldata Pinza_YuMi_L:=[TRUE,[[0,0,114.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    PERS tooldata ToolMaza:=[TRUE,[[0,0,314.2],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];

    PERS wobjdata Maza_L:=[FALSE,TRUE,"",[[600,25,15],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    PERS wobjdata Xilofono:=[FALSE,TRUE,"",[[200,-174,30],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget Punto:=[[89.387953825,156.622113832,160],
[0.066010723,0.842420916,-0.111214903,0.523068666],[0,0,0,4],[101.964426653,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget Target_10_ML:=[[25,25,210],[0,0,1,0],[0,0,0,0],[101.96443128,9E+09,9E+09,9E
+09,9E+09,9E+09]];
    CONST robtarget Target_Nota:=[[132.5,-24,19.5],
[0.320796829,0.781743202,-0.266041626,0.463884484],[-1,1,-1,4],[111.912014269,9E+09,9E+09,9E+09,9E
+09,9E+09]];

    ! Variables de interrupción
    VAR intnum Idi0;
    VAR intnum Idi1;
    VAR intnum Idi2;
    VAR intnum Idi3;
    VAR intnum Idi4;
    VAR intnum Idi5;
    VAR intnum Idi6;
    VAR intnum Idi7;
    VAR intnum Idi8;
    VAR intnum Idi9;
    VAR intnum Idi10;
    VAR intnum Idi11;
    VAR intnum Idi12;

    ! VARIABLES Tasks y syncident (Sincronización)
    PERS tasks Tareas{2}:=[["T_ROB_L"],["T_ROB_R"]];
    VAR syncident sync1;
    VAR syncident sync2;
    VAR syncident sync3;
    VAR syncident sync4;

    ! Distancia entre teclas
    CONST num Tecla:=44;

    ! Variables de interrupción
    PERS bool nosalir;
    VAR num opcion;

    ! Variables escritas desde MATLAB
    PERS string puls_nota_write:="3/1/6/8/6/5"; ! Notas leídas del fichero
    PERS num puls_nota_RT:=1; ! Nota pulsada en tiempo real
```

```
! Variables leídas por MATLAB
PERS num ocupado:=0;

PERS bool puls_L:=FALSE; ! Variable a supervisar por SearchL
VAR num paso:=1;
VAR robtarget sp;
VAR string puls_nota_wr{100}; ! Notas leídas del fichero en string
VAR num puls_nota_num{100}; ! Notas leídas del fichero en num
CONST string notas{8}:=["1","2","3","4","5","6","7","8"];
VAR num orden:=0;
VAR num l:=1;

PROC mainXilofono_L()
    VAR num i:=2;
    VAR num n:=0;
    VAR num rango{4}:=[5,6,7,8];

    First;

    TPWrite "1.- Coger Mazas (DI_8)";
    TPWrite "2.- Leer notas en tiempo real (DI_9)";
    TPWrite "3.- Leer notas de un fichero (DI_10)";
    TPWrite "4.- Dejar Mazas (DI_11)";
    TPWrite "5.- Finalizar programa (DI_12)";

    nosalir:=TRUE;
    opcion:=0;

    WHILE nosalir DO
        TEST opcion
        CASE 1:
            opcion:=0;
            TPWrite "Coger mazas";
            SyncCogeMaza;
        CASE 2:
            ocupado:=0;
            opcion:=0;
            TPWrite "Leer notas en tiempo real";
            FOR i FROM 1 TO 4 DO
                IF puls_nota_RT=rango{i} THEN
                    PulsarTeclas(puls_nota_RT);
                ENDIF
            ENDFOR
        CASE 3:
            opcion:=0;
            l:=1;
            paso:=1;
            i:=2;
            n:=0;
            Escritura;
            TPWrite "Leer notas de un fichero";
```

```
            WHILE i < Dim(puls_nota_num,1) AND puls_nota_num{i}<>0 DO
                IF puls_nota_num{i}<>0 THEN
                    n:=puls_nota_num{i};
                    SyncPulsarTeclas n;
                    i:=i+2;
                ENDIF
            ENDWHILE
            MoveJ Punto,v100,z0,Pinza_YuMi_L;
        CASE 4:
            opcion:=0;
            TPWrite "Dejar mazas";
            SyncDejaMaza;
        CASE 5:
            TPWrite "Finalizar programa";
            opcion:=0;
            nosalir:=FALSE;
        DEFAULT:
        ENDTEST
    ENDWHILE

    TPWrite "FIN DEL PROGRAMA L";
    Stop;
ENDPROC

LOCAL PROC First()

    ! Cancelamos las interrupciones
    IDelete Idi0;
    IDelete Idi1;
    IDelete Idi2;
    IDelete Idi3;
    IDelete Idi4;
    IDelete Idi5;
    IDelete Idi6;
    IDelete Idi7;
    IDelete Idi8;
    IDelete Idi9;
    IDelete Idi10;
    IDelete Idi11;
    IDelete Idi12;

    ! Conectar var intnum con trap
    CONNECT Idi0 WITH trap_1_DO;
    CONNECT Idi1 WITH trap_2_RE;
    CONNECT Idi2 WITH trap_3_MI;
    CONNECT Idi3 WITH trap_4_FA;
    CONNECT Idi4 WITH trap_5_SOL;
    CONNECT Idi5 WITH trap_6_LA;
    CONNECT Idi6 WITH trap_7_SI;
    CONNECT Idi7 WITH trap_8_DO;
    CONNECT Idi8 WITH trap_CogeMaza;
```

```
    CONNECT Idi9 WITH trap_TiempoReal;
    CONNECT Idi10 WITH trap_Fichero;
    CONNECT Idi11 WITH trap_DejaMaza;
    CONNECT Idi12 WITH trap_Fin;

    ISignalDI DI_0, 1, Idi0;
    ISignalDI DI_1, 1, Idi1;
    ISignalDI DI_2, 1, Idi2;
    ISignalDI DI_3, 1, Idi3;
    ISignalDI DI_4, 1, Idi4;
    ISignalDI DI_5, 1, Idi5;
    ISignalDI DI_6, 1, Idi6;
    ISignalDI DI_7, 1, Idi7;
    ISignalDI DI_8, 1, Idi8;
    ISignalDI DI_9, 1, Idi9;
    ISignalDI DI_10, 1, Idi10;
    ISignalDI DI_11, 1, Idi11;
    ISignalDI DI_12, 1, Idi12;

    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L;

ENDPROC

LOCAL TRAP trap_1_DO
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 1_DO";
ENDTRAP

LOCAL TRAP trap_2_RE
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 2_RE";
ENDTRAP

LOCAL TRAP trap_3_MI
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 3_MI";
ENDTRAP

LOCAL TRAP trap_4_FA
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 4_FA";
ENDTRAP

LOCAL TRAP trap_5_SOL
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 5_SOL";
ENDTRAP

LOCAL TRAP trap_6_LA
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 6_LA";
```

```
ENDTRAP

LOCAL TRAP trap_7_SI
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 7_SI";
ENDTRAP

LOCAL TRAP trap_8_DO
    puls_L:=TRUE;
    TPWrite "Choque con la tecla 8_DO";
ENDTRAP

LOCAL TRAP trap_CogeMaza
    opcion:=1;
ENDTRAP

LOCAL TRAP trap_TiempoReal
    opcion:=2;
ENDTRAP

LOCAL TRAP trap_Fichero
    opcion:=3;
ENDTRAP

LOCAL TRAP trap_DejaMaza
    opcion:=4;
ENDTRAP
LOCAL TRAP trap_Fin
    opcion:=5;
    StopMove;
    StorePath;
    MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L;
    RestoPath;
ENDTRAP

! Crear vector de números con las notas que se leen del fichero
LOCAL PROC Escritura()
    VAR string frag;
    VAR string puls_nota_w{100};
    VAR num j;
    VAR num h;
    VAR num k;
    ! Se llena un vector con únicamente los números de cada nota
    FOR j FROM 1 TO StrLen(puls_nota_write) DO
        frag:= StrPart(puls_nota_write,j,1);
        puls_nota_w{j}:=frag;
        IF puls_nota_w{j}<>"/" THEN
            puls_nota_wr{j}:=puls_nota_w{j};
            TPWrite puls_nota_w{j};
        ENDIF
    ENDFOR
```

```
    ! Se pasa el vector de las notas de string a num
    FOR h FROM 1 TO Dim(puls_nota_w,1) DO
        FOR k FROM 1 TO Dim(notas,1) DO
            IF puls_nota_w{h}=notas{k} THEN
                puls_nota_num{l}:=k;
                l:=l+1; ! Devuelve el total de notas solicitadas
            ENDIF
        ENDFOR
    ENDFOR
    ! Se calcula el orden del número total de notas, par o impar
    IF ((l-1) MOD 2) <>0 THEN
        orden:=1;
    ENDIF
ENDPROC

! Pulsar teclas en tiempo real
LOCAL PROC PulsarTeclas(num notas)
    ocupado:=1;
    MoveL Offs(Target_Nota,0,7*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    puls_L:=FALSE;
    SearchL \Stop, puls_L, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    ocupado:=0;
    MoveL Offs(Target_Nota,0,7*Tecla,100),v200,z0,ToolMaza\WObj:=Xilofono;
ENDPROC

! Pulsar teclas leyendo las notas de un fichero
LOCAL PROC SyncPulsarTeclas(num notas)
    WaitSyncTask sync2,Tareas;
    WaitTime 1;

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v150,z0,ToolMaza\WObj:=Xilofono;

    puls_L:=FALSE;
    SearchL \Stop, puls_L, sp, Offs(Target_Nota,0,notas*Tecla,0), v100, ToolMaza\WObj:=Xilofono;

    MoveL Offs(Target_Nota,0,notas*Tecla,20),v200,z0,ToolMaza\WObj:=Xilofono;

    IF (orden=0 AND paso=((l-1)/2)) THEN
        MoveL Offs(Target_Nota,0,7*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
    ELSE
        MoveL Offs(Target_Nota,0,7*Tecla,100),v150,z0,ToolMaza\WObj:=Xilofono;
        WaitTime 1;
        WaitSyncTask sync3,Tareas;
    ENDIF
    paso:=paso+1;
ENDPROC
```

```
    ! Coger maza izquierda
    LOCAL PROC SyncCogeMaza()
        MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L;

        AbrirPinza;

        WaitSyncTask sync1,Tareas;

        MoveJ Offs(Target_10_ML,0,0,100),v300,z0,Pinza_YuMi_L\WObj:=Maza_L;
        MoveL Offs(Target_10_ML,0,0,0),v50,fine,Pinza_YuMi_L\WObj:=Maza_L;

        CerrarPinza;

        MoveL Offs(Target_10_ML,0,0,50),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
        MoveL Offs(Target_10_ML,10,0,50),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
        MoveJ Punto,v300,fine,Pinza_YuMi_L;
    ENDPROC

    ! Dejar maza izquierda
    LOCAL PROC SyncDejaMaza()
        MoveJ Punto,v300,fine,Pinza_YuMi_L;

        WaitSyncTask sync4,Tareas;

        MoveJ Offs(Target_10_ML,10,0,50),v300,z0,Pinza_YuMi_L\WObj:=Maza_L;
        MoveL Offs(Target_10_ML,0,0,50),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
        MoveL Offs(Target_10_ML,0,0,0),v50,fine,Pinza_YuMi_L\WObj:=Maza_L;

        AbrirPinza;

        MoveL Offs(Target_10_ML,0,0,100),v50,z0,Pinza_YuMi_L\WObj:=Maza_L;
        MoveAbsJ Inicio,v300,fine,Pinza_YuMi_L;
    ENDPROC

    ! Abre la pinza
    LOCAL PROC AbrirPinza()
        Set DO_12; ! Abrir y desattacher
        WaitTime 2;
        Reset DO_12;
    ENDPROC

    ! Cerrar Pinza
    LOCAL PROC CerrarPinza()
        Set DO_13;  ! Activar los sensores de attacher
        WaitTime 2;
        Reset DO_13;
    ENDPROC
ENDMODULE
```

ANEXO B

En este anexo se incluye el código desarrollado para la programación de la interfaz de usuario con la aplicación “AppDesigner” de MATLAB.

```
classdef Elena_Pozas_TFG_exported < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure           matlab.ui.Figure
        Lamp               matlab.ui.control.Lamp
        DESCONECTARButton  matlab.ui.control.Button
        DejarMazasButton   matlab.ui.control.Button
        LeerFicheroButton  matlab.ui.control.Button
        DO_8Button         matlab.ui.control.Button
        SI_7Button         matlab.ui.control.Button
        LA_6Button         matlab.ui.control.Button
        SOL_5Button        matlab.ui.control.Button
        FA_4Button         matlab.ui.control.Button
        MI_3Button         matlab.ui.control.Button
        RE_2Button         matlab.ui.control.Button
        DO_1Button         matlab.ui.control.Button
        CogerMazasButton   matlab.ui.control.Button
        CONECTARButton     matlab.ui.control.Button
        ListBox            matlab.ui.control.ListBox
    end


    properties (Access = private)
        da
        stop

        pulsItem_input1
        pulsItem_input2
        pulsItem_input3
        pulsItem_input4
        pulsItem_input5

        notaItem_RT_R
        notaItem_RT_L
        ocupadoVal_R
        ocupadoVal_L

        notaItem_write_R
        notaItem_write_L
    end

    methods (Access = private)
        % Función que genera el sonido de las notas
        function results = sonido(app,nota)
            Fs=8000;
            Ts=1/Fs;
            t=[0:Ts:0.3];

            INICIO = 0;
            DO_1 = 261.626;
            RE_2 = 293.665;
            MI_3 = 329.628;
            FA_4 = 349.228;
            SOL_5 = 391.995;
```

```
            LA_6 = 440;
            SI_7 = 493.883;
            DO_8 = 523.251;

            vector_nota=[DO_1,RE_2,MI_3,FA_4,SOL_5,LA_6,SI_7,DO_8,INICIO];
            notes = [vector_nota(nota)];
            x = cos(2*pi*notes*t);
            sig = reshape(x',length(t),1);
            soundsc(sig,Fs)
        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: CONECTARButton
        function CONECTARButtonPushed(app, event)
            % Comenzar la comunicación creando todos los items que van a ser
            % utilizados y leer las notas que están siendo
            % pulsadas desde el robot
            app.da= opcda('localhost', 'ABB.IRC5.OPC.Server.DA');
            connect(app.da);
            grp = addgroup(app.da);
            set(grp,'UpdateRate',0.001);
            pc='LPSK_IRB14000.';
            app.Lamp.Color=[0.39,0.83,0.07];

            puls_input1=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_8']);
            puls_input2=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_9']);
            puls_input3=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_10']);
            puls_input4=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_11']);
            puls_input5=serveritems(app.da,[pc,'IOSYSTEM.IOSIGNALS.DI_12']);

            % Leer
            nota_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.puls_nota']);

            % Pulsar teclas
            nota_RT_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.↙
puls_nota_RT']);
            nota_RT_L=serveritems(app.da,[pc,'RAPID.T_ROB_L.Xilofono_L.↙
puls_nota_RT']);
            ocupado_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.ocupado']);
            ocupado_L=serveritems(app.da,[pc,'RAPID.T_ROB_L.Xilofono_L.ocupado']);
            % Leer fichero
            nota_write_R=serveritems(app.da,[pc,'RAPID.T_ROB_R.Xilofono_R.↙
puls_nota_write']);
            nota_write_L=serveritems(app.da,[pc,'RAPID.T_ROB_L.Xilofono_L.↙
puls_nota_write']);

            app.pulsItem_input1=additem(grp, puls_input1);
            app.pulsItem_input2=additem(grp, puls_input2);
            app.pulsItem_input3=additem(grp, puls_input3);
            app.pulsItem_input4=additem(grp, puls_input4);
            app.pulsItem_input5=additem(grp, puls_input5);
```

```
            notaItem_R=additem(grp, nota_R);

            app.notaItem_RT_R=additem(grp, nota_RT_R);
            app.notaItem_RT_L=additem(grp, nota_RT_L);
            ocupadoItem_R=additem(grp, ocupado_R);
            ocupadoItem_L=additem(grp, ocupado_L);

            app.notaItem_write_R= additem(grp, nota_write_R);
            app.notaItem_write_L= additem(grp, nota_write_L);

            app.stop=0;
            sonido(app,9)
            % Leer
            write(notaItem_R,0)
            registro=0;
            while app.stop==0
                app.ocupadoVal_R=read(ocupadoItem_R);
                app.ocupadoVal_L=read(ocupadoItem_L);
                notaVal_R = read(notaItem_R);
                if ((notaVal_R.Value)~=0) && registro==0
                    sonido(app,notaVal_R.Value)
                    registro=1;
                elseif notaVal_R.Value==0
                    registro=0;
                end
                pause(0.001)
            end
        end

        % Button pushed function: CogerMazasButton
        function CogerMazasButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a coger las
            % mazas
            write(app.pulsItem_input1,'1')
            pause(0.1)
            write(app.pulsItem_input1,'0')
        end

        % Button pushed function: DO_1Button
        function DO_1ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_R,1)
            if app.ocupadoVal_L.Value==0 % solo solicita esta nota si el brazo↙
izquierdo no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: RE_2Button
        function RE_2ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
```

```
            % notas en tiempo real
            write(app.notaItem_RT_R,2)
            if app.ocupadoVal_L.Value==0 % solo solicita esta nota si el brazo↙
izquierdo no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: MI_3Button
        function MI_3ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_R,3)
            if app.ocupadoVal_L.Value==0 % solo solicita esta nota si el brazo↙
izquierdo no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: FA_4Button
        function FA_4ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_R,4)
            if app.ocupadoVal_L.Value==0 % solo solicita esta nota si el brazo↙
izquierdo no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: SOL_5Button
        function SOL_5ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_L,5)
            if app.ocupadoVal_R.Value==0 % solo solicita esta nota si el brazo derecho↙
no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: LA_6Button
        function LA_6ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_L,6)
            if app.ocupadoVal_R.Value==0 % solo solicita esta nota si el brazo derecho↙
```

```
no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: SI_7Button
        function SI_7ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_L,7)
            if app.ocupadoVal_R.Value==0 % solo solicita esta nota si el brazo derecho↙
no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: DO_8Button
        function DO_8ButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a leer las
            % notas en tiempo real
            write(app.notaItem_RT_L,8)
            if app.ocupadoVal_R.Value==0 % solo solicita esta nota si el brazo derecho↙
no está ocupado
                write(app.pulsItem_input2,'1')
                pause(0.1)
                write(app.pulsItem_input2,'0')
            end
        end

        % Button pushed function: LeerFicheroButton
        function LeerFicheroButtonPushed(app, event)
            % Leer del fichero la canción seleccionada en el menú
            menu= app.ListBox.Value;

            if strcmp(menu, 'Canción 1')
                cancion = 2;
            elseif strcmp(menu, 'Canción 2')
                cancion = 4;
            elseif strcmp(menu, 'Canción 3')
                cancion = 6;
            elseif strcmp(menu, 'Canción 4')
                cancion = 8;
            end

            % Escribir canción
            A = importdata('canciones.txt');
            write(app.notaItem_write_R,A(cancion))
            write(app.notaItem_write_L,A(cancion))

            write(app.pulsItem_input3,'1')
            pause(0.1)
```

```
            write(app.pulsItem_input3,'0')
        end

        % Button pushed function: DejarMazasButton
        function DejarMazasButtonPushed(app, event)
            % Activar entrada del controlador corespondiente a dejar las mazas
            write(app.pulsItem_input4,'1')
            pause(0.1)
            write(app.pulsItem_input4,'0')
        end

        % Button pushed function: DESCONECTARButton
        function DESCONECTARButtonPushed(app, event)
            % Dejar de leer las notas y finalizar la comunicación
            app.stop=1;
            write(app.pulsItem_input5,'1')
            pause(0.1)
            write(app.pulsItem_input5,'0')
            disconnect(app.da)
            app.Lamp.Color=[0.90,0.21,0.21];
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [0.149 0.149 0.149];
            app.UIFigure.Position = [100 100 599 480];
            app.UIFigure.Name = 'MATLAB App';

            % Create ListBox
            app.ListBox = uilistbox(app.UIFigure);
            app.ListBox.Items = {'Canción 1', 'Canción 2', 'Canción 3', 'Canción 4'};
            app.ListBox.Position = [443 338 125 106];
            app.ListBox.Value = 'Canción 1';

            % Create CONECTARButton
            app.CONECTARButton = uibutton(app.UIFigure, 'push');
            app.CONECTARButton.ButtonPushedFcn = createCallbackFcn(app,↙
@CONECTARButtonPushed, true);
            app.CONECTARButton.BackgroundColor = [1 1 1];
            app.CONECTARButton.FontSize = 13;
            app.CONECTARButton.FontWeight = 'bold';
            app.CONECTARButton.Position = [34 409 115 26];
            app.CONECTARButton.Text = 'CONECTAR';

            % Create CogerMazasButton
            app.CogerMazasButton = uibutton(app.UIFigure, 'push');
            app.CogerMazasButton.ButtonPushedFcn = createCallbackFcn(app,↙
@CogerMazasButtonPushed, true);
```

```
            app.CogerMazasButton.FontSize = 13;
            app.CogerMazasButton.FontWeight = 'bold';
            app.CogerMazasButton.Position = [35 338 125 53];
            app.CogerMazasButton.Text = 'Coger Mazas';

            % Create DO_1Button
            app.DO_1Button = uibutton(app.UIFigure, 'push');
            app.DO_1Button.ButtonPushedFcn = createCallbackFcn(app, @DO_1ButtonPushed,↙
true);
            app.DO_1Button.BackgroundColor = [1 0 0];
            app.DO_1Button.FontSize = 13;
            app.DO_1Button.FontWeight = 'bold';
            app.DO_1Button.Position = [35 80 57 229];
            app.DO_1Button.Text = 'DO_1';

            % Create RE_2Button
            app.RE_2Button = uibutton(app.UIFigure, 'push');
            app.RE_2Button.ButtonPushedFcn = createCallbackFcn(app, @RE_2ButtonPushed,↙
true);
            app.RE_2Button.BackgroundColor = [1 0.4118 0.1608];
            app.RE_2Button.FontSize = 13;
            app.RE_2Button.FontWeight = 'bold';
            app.RE_2Button.Position = [103 80 57 229];
            app.RE_2Button.Text = 'RE_2';

            % Create MI_3Button
            app.MI_3Button = uibutton(app.UIFigure, 'push');
            app.MI_3Button.ButtonPushedFcn = createCallbackFcn(app, @MI_3ButtonPushed,↙
true);
            app.MI_3Button.BackgroundColor = [1 1 0];
            app.MI_3Button.FontSize = 13;
            app.MI_3Button.FontWeight = 'bold';
            app.MI_3Button.Position = [171 80 57 229];
            app.MI_3Button.Text = 'MI_3';

            % Create FA_4Button
            app.FA_4Button = uibutton(app.UIFigure, 'push');
            app.FA_4Button.ButtonPushedFcn = createCallbackFcn(app, @FA_4ButtonPushed,↙
true);
            app.FA_4Button.BackgroundColor = [0 1 0];
            app.FA_4Button.FontSize = 13;
            app.FA_4Button.FontWeight = 'bold';
            app.FA_4Button.Position = [239 80 57 229];
            app.FA_4Button.Text = 'FA_4';

            % Create SOL_5Button
            app.SOL_5Button = uibutton(app.UIFigure, 'push');
            app.SOL_5Button.ButtonPushedFcn = createCallbackFcn(app,↙
@SOL_5ButtonPushed, true);
            app.SOL_5Button.BackgroundColor = [0 1 1];
            app.SOL_5Button.FontSize = 13;
            app.SOL_5Button.FontWeight = 'bold';
            app.SOL_5Button.Position = [307 80 57 229];
            app.SOL_5Button.Text = 'SOL_5';
```

```
            % Create LA_6Button
            app.LA_6Button = uibutton(app.UIFigure, 'push');
            app.LA_6Button.ButtonPushedFcn = createCallbackFcn(app, @LA_6ButtonPushed,↙
true);
            app.LA_6Button.BackgroundColor = [0 0 1];
            app.LA_6Button.FontSize = 13;
            app.LA_6Button.FontWeight = 'bold';
            app.LA_6Button.Position = [375 80 57 229];
            app.LA_6Button.Text = 'LA_6';

            % Create SI_7Button
            app.SI_7Button = uibutton(app.UIFigure, 'push');
            app.SI_7Button.ButtonPushedFcn = createCallbackFcn(app, @SI_7ButtonPushed,↙
true);
            app.SI_7Button.BackgroundColor = [0.4941 0.1843 0.5569];
            app.SI_7Button.FontSize = 13;
            app.SI_7Button.FontWeight = 'bold';
            app.SI_7Button.Position = [443 80 57 229];
            app.SI_7Button.Text = 'SI_7';

            % Create DO_8Button
            app.DO_8Button = uibutton(app.UIFigure, 'push');
            app.DO_8Button.ButtonPushedFcn = createCallbackFcn(app, @DO_8ButtonPushed,↙
true);
            app.DO_8Button.BackgroundColor = [1 0 0];
            app.DO_8Button.FontSize = 13;
            app.DO_8Button.FontWeight = 'bold';
            app.DO_8Button.Position = [511 80 57 229];
            app.DO_8Button.Text = 'DO_8';

            % Create LeerFicheroButton
            app.LeerFicheroButton = uibutton(app.UIFigure, 'push');
            app.LeerFicheroButton.ButtonPushedFcn = createCallbackFcn(app,↙
@LeerFicheroButtonPushed, true);
            app.LeerFicheroButton.FontSize = 13;
            app.LeerFicheroButton.FontWeight = 'bold';
            app.LeerFicheroButton.Position = [443 338 125 33];
            app.LeerFicheroButton.Text = 'Leer Fichero';

            % Create DejarMazasButton
            app.DejarMazasButton = uibutton(app.UIFigure, 'push');
            app.DejarMazasButton.ButtonPushedFcn = createCallbackFcn(app,↙
@DejarMazasButtonPushed, true);
            app.DejarMazasButton.FontSize = 13;
            app.DejarMazasButton.FontWeight = 'bold';
            app.DejarMazasButton.Position = [183 338 125 53];
            app.DejarMazasButton.Text = 'Dejar Mazas';

            % Create DESCONECTARButton
            app.DESCONECTARButton = uibutton(app.UIFigure, 'push');
            app.DESCONECTARButton.ButtonPushedFcn = createCallbackFcn(app,↙
@DESCONECTARButtonPushed, true);
            app.DESCONECTARButton.BackgroundColor = [1 1 1];
            app.DESCONECTARButton.FontSize = 13;
            app.DESCONECTARButton.FontWeight = 'bold';
```

```
            app.DESCONECTARButton.Position = [34 33 115 26];
            app.DESCONECTARButton.Text = 'DESCONECTAR';

            % Create Lamp
            app.Lamp = uilamp(app.UIFigure);
            app.Lamp.Position = [160 412 20 20];
            app.Lamp.Color = [0.902 0.2118 0.2118];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = Elena_Pozas_TFG_exported

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```