Universidad Zaragoza
1542

Trabajo Fin de Grado

Simulación y programación de un Robot Industrial ABB
Simulation and programming of an Industrial Robot ABB

Autor/es

Pablo Molina Mata

Director/es

José Jesús Guerrero Campo

Escuela de Ingeniería y Arquitectura

2017

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

D./Dª. PABLO MOLINA MATA,

con nº de DNI 73007298 en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Simulación y programación de un Robot Industrial ABB

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, a 23 de Noviembre de 2017

Fdo: PABLO MOLINA MATA

Resumen

El presente proyecto describe el trabajo desarrollado para generar material docente, tanto teórico como práctico en forma de guías y de estaciones de simulación, para la asignatura Automatización Flexible y Robótica, del Grado de Tecnologías Industriales, y se ha desarrollado en el Laboratorio L0.06 del edificio Ada Byron de la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza.

El trabajo tiene como denominador común la lectura y comprensión de gran cantidad de documentación técnica en referencia al lenguaje de programación de robots RAPID, al Robot de ABB modelo IRB 120 disponible en dicho laboratorio, así como al software simulación de entornos de producción robotizados, Robot Studio. Como resultado del proyecto, se ha generado material docente para prácticas de programación de robots, así como la documentación de todo el trabajo realizado, que se ha dividido principalmente en tres bloques temáticos:

1. Introducción al robot industrial IRB 120, en la que se muestran sus principales características, así como al lenguaje de programación RAPID.
2. Explicación detallada de las opciones de Robot Studio como software de simulación, incluyendo la herramienta FlexPendant.
3. Creación de las estaciones de simulación, adaptadas a las necesidades de las prácticas de la asignatura, en que se incluyen y se desarrollan las principales funciones de Robot Studio y RAPID.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1 Contexto y motivación

La Robótica Industrial es una rama de la ingeniería que si bien se encuentra ya muy desarrollada e implantada en gran parte de las industrias de los países desarrollados, todavía se encuentra en un proceso de mejora, perfeccionamiento y aumento de sus capacidades.

El contexto en el que se desarrolla este proyecto parte de la reciente adquisición de un robot IRB 120 por parte del Departamento de Informática e Ingeniería de Sistemas para el laboratorio 0.06 del Edificio Ada Byron, en la Escuela de Ingeniería y Arquitectura. Más adelante en este proyecto se detallarán sus principales características.

En este contexto, la motivación de este proyecto tiene dos claras vertientes. Por un lado, el interés del proyectando en la robótica, particularizando en la robótica de carácter industrial, lo que hacía que este proyecto quedara perfectamente integrado en su formación.

Y por otro lado, el interés de generar nuevo contenido para futuros estudiantes de la Escuela, principalmente de prácticas de programación de robots, en esta asignatura o en otras de carácter similar, con el objetivo de mejorar su formación en un ámbito tan importante como este.

1.2 Objetivos y alcance del proyecto

El objetivo principal de este Trabajo de Fin de Grado, ha sido el apoyo a la docencia de la asignatura de Automatización Flexible y Robótica, del grado de Tecnologías Industriales, mediante la realización de Guías y Prácticas de Laboratorio con el simulador Robot Studio y de programación en RAPID, así como Trabajos de la asignatura.

Particularizando en este Trabajo, los objetivos son:

- Realización de una guía, resumida y adaptada para los alumnos, de programación en RAPID.

- Realización de una simulación sencilla, guiada paso a paso, en Robot Studio, para que los alumnos puedan conocer y comprender las nociones básicas de dicho simulador.
- Programar un ejemplo, mediante la herramienta FlexPendant que ofrece Robot Studio, que posteriormente será la empleada para programar en el Robot real.
- Creación de varias herramienta tipo “pinza”, en el simulador, para que pueda ser empleada posteriormente en las practicas.
- Creación de varias estaciones de simulación distintas, en las que se demuestra las capacidades del simulador, y que están adaptadas para ser utilizadas en la asignatura como prácticas o trabajos. Entre ellas se encuentran tanto prácticas meramente de simulación, como otras en las que se puede trabajar con el robot real, incluyendo en algunas, el trabajo coordinado con dos robots.

Respecto al alcance final del proyecto, que se irá desgranando a lo largo de los diferentes capítulos del mismo, se destacan los siguientes hitos:

- Se ha efectuado una intensiva lectura de los manuales y guías oficiales proporcionados por ABB, tanto del Lenguaje RAPID y el Robot IRB 120, como del simulador RobotStudio.
- Se han creado guías más asequibles y manejables para el alumno de dichos manuales, así como ejemplos guiados “paso a paso” para conseguir una rápida comprensión de ambos por su parte.
- Se ha simulado varios escenarios, adaptable a su realización en el laboratorio 0.06 del edificio Ada Byron, durante las prácticas.
- Se ha desarrollado una simulación con movimiento coordinado entre dos robots, basada en un reto de ABB, llamado “Fanta Can Challenge”.
- Respecto a la documentación del trabajo realizado, además de los manuales, se encuentra todo el contenido de esta memoria y sus apéndices. Así como todos los programas debidamente comentados.

1.3 Organización de los capítulos

Después de esta visión general de lo que es este proyecto, se va a explicar brevemente la forma en la que está estructurada la memoria.

En el Capítulo 2, Robot IRB 120 y Lenguaje RAPID, se explicará brevemente el robot de ABB, modelo IRB 120, así como introducir al Lenguaje de programación RAPID.

En el Capítulo 3, Simulación con RobotStudio, se introduce el simulador RobotStudio en su versión 6.05.01, que es la que se encuentra en el laboratorio así como la descripción de la herramienta FLexPendant.

En el Capítulo 4, Escenarios de Simulación para prácticas, se explicarán detalladamente en qué consisten y cómo se han desarrollado dichos escenarios.

En el Capítulo 5, Trabajo de simulación Multi-Robot, se explicará como ejecutar movimiento coordinado entre dos robots dentro de RobotStudio, así como explicar detalladamente cómo se ha desarrollado el escenario correspondiente.

Por último, el capítulo 6 incluye las conclusiones del proyecto.

Capítulo 2

Robot IRB 120 y Lenguaje RAPID

En este capítulo se explican las características principales del Robot de ABB modelo IRB 120, así como introducir al Lenguaje RAPID, el cual se explicará con un mayor detalle en el Anexo A. Con ello se pretende presentar el Robot a partir del cual se han creado las posteriores estaciones, así como facilitar la comprensión del código de las mismas.

2.1 Robot IRB 120

El Robot IRB 120 s miembro de la generación más reciente de ABB Robotics de robots industriales de 6 ejes. Consta de una carga útil de 3 kg, y se ha diseñado para industrias de fabricación que utilizan una automatización flexible basada en robot, o como es el caso, para uso académico debido a su pequeño tamaño y ergonomía.

El robot tiene una estructura abierta especialmente adaptada para un uso flexible y presenta unas grandes posibilidades de comunicación con sistemas externos.

Este robot está equipado con el controlador IRC5 y el software de control de robots RobotWare, el cual admite todos los aspectos del sistema de robot, tales como el control del movimiento, el desarrollo y le ejecución de programas, la comunicación, etc.

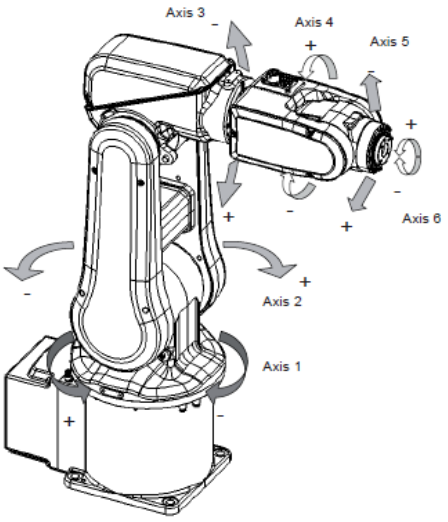

Figura 2.1 Ejes del Robot IRB 120

El Robot IRB 120 admite su montaje en el suelo, en posición invertida o en pared con cualquier ángulo, inclinado alrededor del eje X o Y. En el caso del Robot disponible en el laboratorio se encuentra como indica la Figura 2.2.

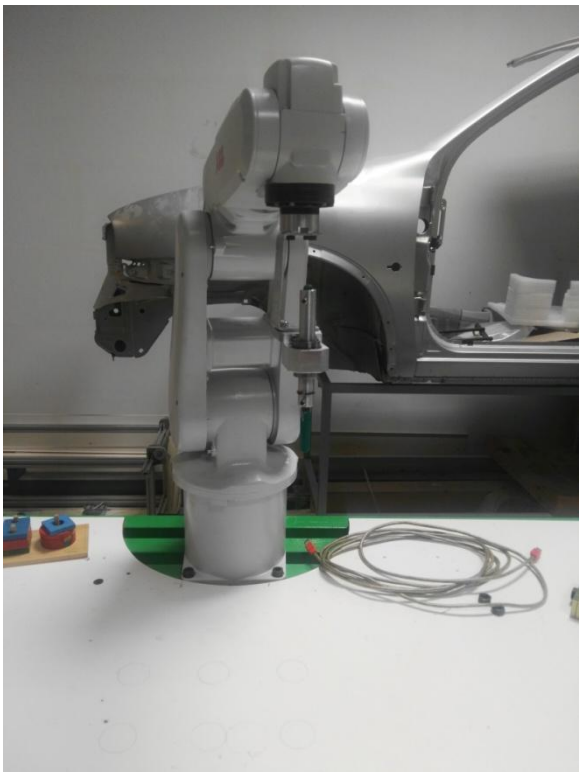

Figura 2.2 Robot IRB 120 montado en el Laboratorio 0.06

En lo referente a las conexiones con el usuario, mencionar que los cables están integrados en el robot y los conectores están situados en la carcasa del brazo superior, así como en la base, el conector UTOW01210SH05 (R3.CP/CS) y el conector UTOW71210PH06 (R1.CP/CS) respectivamente.

En la tabla 2.1 se muestran las características más relevantes del robot IRB 120 en lo referente a su número de ejes, carga máxima admisible, distancia alcanzable en horizontal, peso del robot, además de las velocidades de dichos ejes.

N° de ejes	6
Carga Máxima	3 kg
Distancia alcanzable (Horizontal)	580 mm

Velocidad del eje 1	250 °/s
Velocidad del eje 2	250 °/s
Velocidad del eje 3	250 °/s
Velocidad del eje 4	320 °/s
Velocidad del eje 5	320 °/s
Velocidad del eje 6	420 °/s

Tabla 2.1 Características principales del Robot IRB 120

2.2 Lenguaje RAPID

RAPID (Robotics Application Programming Interactive Dialogue) es un lenguaje de programación de robots, que permite escribir las instrucciones precisas para que el robot realice la rutina, o la acción que se desee.

Se trata de un lenguaje de programación de alto nivel, es decir, un lenguaje de programación con palabras y letras fácilmente comprensibles.

RAPID fue desarrollado por la empresa suiza ABB y es lenguaje que emplean sus robots, por tanto, en este trabajo (Anexo A), se va a proceder a describir sus características principales.

Este lenguaje de programación no distingue entre mayúsculas y minúsculas, además posee una serie de palabras clave, que tienen un significado dentro del mismo y por tanto no pueden ser empleados como identificadores de módulos, rutinas, datos y etiquetas.

Se permite emplear espacios en cualquier del código del programa, exceptuando dentro de un identificador, palabra reservada por RAPID, un valor numérico o en su defecto, una marca de sustitución.

Del mismo modo, los comentarios en este lenguaje empezarán con el signo de exclamación "!" y terminarán con un carácter de salto de línea.

En lo referente a los sistemas de coordenadas, se dispone de un sistema de coordenadas de la base, que el programa por defecto lo situará de forma que coincida con la base del Robot, y posteriormente se pueden crear “objetos de trabajo”.

Estos objetos de trabajo son sistemas de coordenadas relativos al sistema de coordenadas de usuario que se quiera, el cual no se define expresamente en el código del programa, sino que dentro de la declaración del objeto de trabajo se incluye la posición y orientación del ya mencionado sistema de coordenadas del usuario.

Por otro lado, RAPID permite controlar el flujo del programa mediante instrucciones como IF…ENDIF, FOR…ENDFOR, TASTE…CASE, etc… por lo se pueden crear códigos complejos y depurados si se desea.

El tratamiento de entradas y salidas por parte de este lenguaje, consta de las principales instrucciones como pueden ser SET, RESET, WAIT, etc… tanto de señales digitales como analógicas. Pero por otro lado, también se dispone de una gran cantidad de instrucciones duales, es decir, instrucciones que activan o desactivan una señal, pero que a su vez realizan otra función como mover el robot a lo largo de una trayectoria, por ejemplo.

Capítulo 3

Simulación con RobotStudio y FlexPendant

En este capítulo va a proceder a la explicación de las características de la herramienta de simulación RobotStudio, así como de una explicación más detallada del FlexPendant. El objetivo del mismo, es introducir a la principal herramienta objeto de estudio en este proyecto, así como la herramienta de manipulación del robot IRB 120 real.

3.1 Simulación con RobotStudio

El software de simulación RobotStudio, se trata, de una herramienta de ingeniería diseñada para configurar y programar robots de ABB. Dicha configuración y programación se puede realizar tanto en robots reales en el centro de producción, conectándolo a un controlador real, como robots virtuales, y su consiguiente controlador virtual, en el PC.

En este trabajo, se va a considerar únicamente la segunda función de RobotStudio, es decir la programación fuera de línea y la simulación de células de robot.

Esta herramienta, permite trabajar con un controlador fuera de línea, que constituye un controlador IRC5 virtual que se ejecuta localmente en el PC.

RobotStudio permite incluir en la estación robots industriales de ABB, herramientas disponibles en la biblioteca, crearte herramientas personalizadas, crear mecanismos que muevan al robot o a la herramienta, crear geometría compleja combinando diferentes tipos de elementos, importar geometría de programas como AutoCAD, SolidWorks, etc…

Por otro lado, a la hora de trabajar con el simulador, se puede realizar de dos modos. Existe un modo visual, es decir, crear objetivos, objetos de trabajo, rutinas, etc… mediante el empleo de funciones incluidas en las distintas pestañas de la interfaz. Una vez se ha creado lo mencionado previamente, RobotStudio permite “sincronizar la estación con RAPID”, es decir volcará la información que hayáis generado manualmente y generará código RAPID a partir de ella.

Sin embargo, esta herramienta permite trabajar del modo opuesto, es decir, escribir todo el código en RAPID primero, y “sincronizar RAPID con la estación”, con lo que se vuelca la información generada en el código a la estación simulada.

A continuación se va a proceder a aclarar y definir, los principales conceptos que se manejan en RobotStudio:

- **Herramienta:** Una herramienta, se trata de un objeto que puede montarse directa o indirectamente sobre el disco giratorio del robot, o montarse en una posición fija dentro del área de trabajo del robot. Todas las herramientas, deberán tener definido un TCP, Tool Center Point.
- **TCP:** El TCP, o punto central de la herramienta, es el punto respecto del cual se definen todas las posiciones del robot. Normalmente, dicho punto se define respecto de una posición de la brida giratoria del manipulador. A su vez, este punto constituye el origen del sistema de coordenadas de la herramienta, y puede estar definido tanto móvil como fijo.
- **Objeto de trabajo:** El objeto de trabajo, es un sistema de coordenadas que tiene asociadas determinadas propiedades específicas. Se emplea principalmente para simplificar la programación durante la edición de programas debido a los desplazamientos asociados a tareas, objetos, procesos y otros elementos concretos. El sistema de coordenadas del objeto debe de ser definido en dos bases de coordenadas: la base de coordenadas de usuario, y la base de coordenadas de objeto.
- **Objetivos:** Los objetivos son las posiciones objetivo del robot. Hay que definirlos asociados a un objeto de trabajo, y un TCP de una herramienta, en particular, además de indicar la configuración con la el robot llegará a dicha posición.
- **Trayectorias:** Las rutas o trayectorias definen los movimientos a realizar por el robot, y están definidas en mayor medida a partir de los objetivos.

Por otro lado, RobotStudio permite crear componentes inteligentes, SmartComponentes asociados, o no, a otros objetos de la estación, por lo que adquiere una mayor complejidad y capacidad de realizar estaciones industriales complejas. Más adelante en este trabajo, se va a profundizar en mayor medida en los componentes inteligentes.

Por otro lado, este simulador tiene una opción que permite trabajar con una herramienta de programación virtual FlexPendant, idéntica a la que se emplea para un robot industrial real. A continuación se explicará detalladamente dicha herramienta.

3.2 FlexPendant

El FlexPendant es una unidad de operador de mano que se usa para realizar muchas de las tareas implicadas en el manejo de un sistema de robot: ejecutar programas, mover el manipulador, modificar programas del robot, etc.

El FlexPendant ha sido diseñado para un funcionamiento continuo en entornos industriales agresivos, y posee de una pantalla táctil que se limpia con facilidad y es resistente al agua, el aceite y las salpicaduras de soladura calientes.

Esta herramienta dispone tanto de componente hardware como de software y es un ordenador completo por sí sólo. No obstante, forma parte del IRC5 y se conecta a dicho controlador mediante un cable y un conector integrados. Sin embargo, existe la opción de desconectar el FlexPendant en el modo automático y seguir trabajando sin él.

En la Figura 3.2.1 se muestra el FlexPendant con los distintos componentes que tiene, y en la Tabla 3.1 se indican los nombres de dichos componentes.

Figura 3.2.1 FlexPendant

A	Conector
B	Pantalla táctil
C	Botón de paro de emergencia
D	Joystick
E	Puerto USB
F	Sistema de agarre
G	Puntero
H	Pulsador de restablecimiento

Tabla 3.1 Componentes del FlexPendant

En los botones que aparecen en la Figura 3.2.1, pueden ser empleados para seleccionar una unidad mecánica, activar o desactivar el modo de movimiento, reorientación o lineal, activar/desactivar incrementos, retroceder un paso (ejecutar una instrucción hacia atrás), iniciar la ejecución del programa, avanzar un paso o detener la ejecución del programa.

Por otro lado, la pantalla táctil se explica más detalladamente en la Figura 3.2.2.

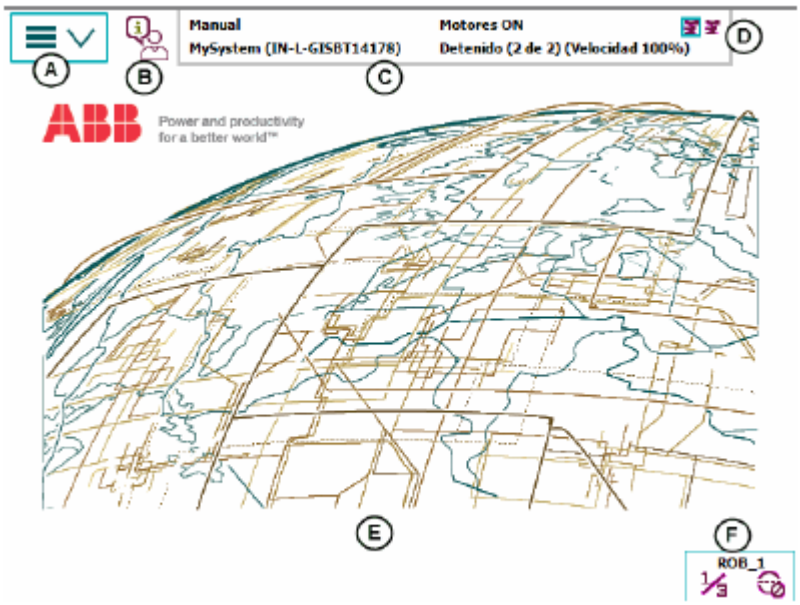

Figura 3.2.2 Pantalla táctil del FlexPendant

Las opciones disponibles en dicha pantalla táctil son las siguientes:

- **Menú ABB** (A): El menú ABB permite seleccionar entre varios elementos distintos, como las entradas y salidas, el movimiento, el editor de programas, los datos de programa, etc.
- **Ventana de operador** (B): La ventana del operador muestra mensajes de los programas del robot, y suelen aparecer cuando el programa requiere algún tipo de respuesta del operador para poder continuar.
- **Barra de estado** (C): La barra de estado muestra información importante acerca del estado del sistema, como por ejemplo el modo de funcionamiento, motores ON/OFF, el estado del programa, etc.
- **Botón Cerrar** (D): Al tocar el botón Cerrar se cierra la vista o aplicación que esté activa actualmente.
- **Barra de tareas** (E): La barra de tareas muestra todas las vistas abiertas a la vez, y se emplea para cambiar entre ellas.
- **Menú de configuración rápida** (F): El menú de configuración rápida contiene valores sobre el movimiento y la ejecución del programa. Y permite, entre otras cosas, cambiar las propiedades del movimiento de manera más rápida.

Capítulo 4

Escenarios de simulación para prácticas

4.1 ESCENARIO PINTANOMBRE

En este escenario, el Robot IRB 120 recorrerá unas trayectorias, definidas por el usuario, que simularán las letras correspondientes al nombre que él elija escribir, en este caso será mi nombre, Pablo.

Se ha elegido este escenario, ya que existe la herramienta necesaria tanto en el simulador, como en el Laboratorio 0.06 del edificio Ada Byron, y es algo aplicable y realizable con el robot real en dicho laboratorio, si así lo quisiesen los profesores.

En este escenario se pretende que el alumno se familiarice con los distintos tipos de trayectorias que se pueden realizar en RAPID, lineales, circulares y articulares. Por otro lado, se pretende dar a conocer la interesante opción de rastrear los movimientos del TCP, así como las posibilidades que ello ofrece.

4.1.1 Creación del Escenario

Este escenario está formado por el Robot IRB 120 y por dos mesas, “Mesa_Robot” y “Mesa_Piezas”, las cuales tienen las siguientes dimensiones:

- Mesa_Robot: Se trata de una mesa circular de 400mm de diámetro y 350mm de altura, como muestra la Figura 4.1.1. Hará de soporte para el Robot IRB 120, y se ha tomado como modelo la mesa situada en el Laboratorio 0.06 del edificio Ada Byron.

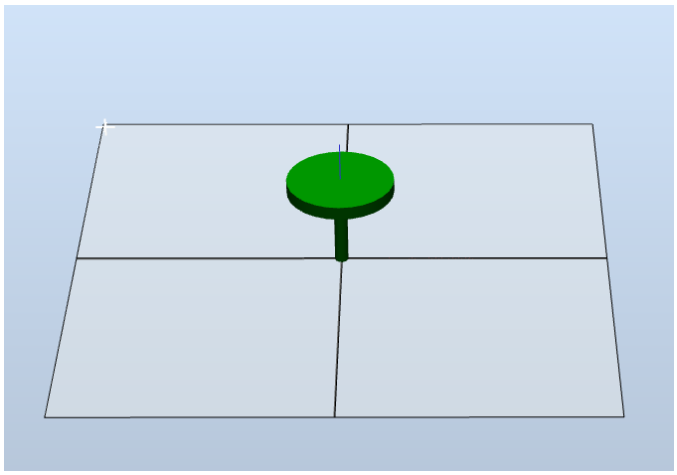

Figura 4.1.1 Mesa soporte del Robot

- Mesa_Piezas: Esta pieza se trata de una mesa rectangular de 1200x600 y con 350mm de altura, como muestra la Figura 4.1.2. A dicha pieza se le ha restado la mesa anterior, para que ambos objetos no colisionasen. Para la realización de esta pieza, se ha tomado como ejemplo la que se encuentra en el Laboratorio 0.06 del edificio Ada Byron, y sobre ella se pintarán las trayectorias correspondientes.

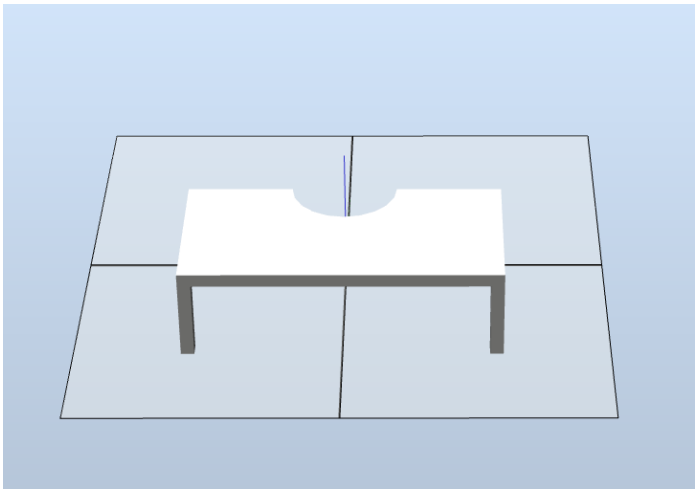

Figura 4.1.2 Mesa soporte de las piezas.

Ambas mesas se encuentran las librerías de los ordenadores del laboratorio, para que puedan ser usadas si fuese necesario.

El Robot IRB 120 se colocará en el centro de la Mesa_Robot, y se le conectará una herramienta genérica, “My_Pen”, que se encuentra en las bibliotecas proporcionadas por defecto por RobotStudio. Dicha herramienta, se muestra en la Figura 4.1.3.

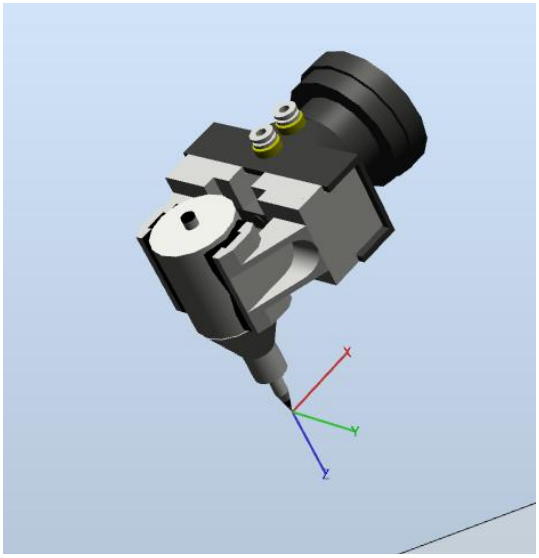

Figura 4.1.3 Herramienta “MyPen”.

4.1.2 Creación de objetos de trabajo y posiciones

En el caso de este escenario en particular, se va a definir los puntos necesarios para la representación de 5 letras, P, A, B, L y O.

Estas posiciones, o “RobotTarget”, se han definido a partir de sistema de coordenadas de objeto, llamado “Mesa_Piezas, que se encuentra localizado en la esquina superior derecha de la pieza, en coordenadas globales [600,0,0]. Además no se ha realizado giro alguno en ningún eje, por tanto su orientación expresada en cuaternios será [1,0,0,0].

De este modo, los puntos definidos para cada letra serían los siguientes, ilustrados de la Tabla 4.1.1 a la Tabla 4.1.5.

Letra	Nombre	Posición	Orientación	Configuración
"P"	P1	[-100,-200,0]	[0.258819045,0,-0.965925826,0]	[-1,1,-2,0]
	P2	[-100,-400,0]	[0.258819045,0,-0.965925826,0]	[-1,0,-2,0]
	P3	[-100,-300,0]	[0.258819045,0,-0.965925826,0]	[-1,1,-2,0]
	P4	[-150,-350,0]	[0.258819045,0,-0.965925826,0]	[-1,1,-2,0]

Tabla 4.1.1 Posiciones de trabajo en la letra P

Letra	Nombre	Posición	Orientación	Configuración
"A"	A1	[-250,-250,0]	[0.25,-0.25,-0.933012702,-0.066987298]	[-1,-1,0,1]
	A2	[-350,-450,0]	[0.25,-0.25,-0.933012702,-0.066987298]	[-1,-1,0,1]
	A3	[-450,-250,0]	[0.25,-0.25,-0.933012702,-0.066987298]	[-1,-1,-1,1]
	A4	[-400,-350,0]	[0.25,-0.25,-0.933012702,-0.066987298]	[-1,-1,0,1]
	A5	[-300,-350,0]	[0.25,-0.25,-0.933012702,-0.066987298]	[-1,-1,0,1]

Tabla 4.1.2 Posiciones de trabajo en la letra A

Letra	Nombre	Posición	Orientación	Configuración
“B”	B1	[-550,-300,0]	[0.183012702,-0.683012702,-0.683012702,-0.183012702]	[-1,0,0,1]
	B2	[-550,-400,0]	[0.183012702,-0.683012702,-0.683012702,-0.183012702]	[-1,0,0,1]
	B3	[-550,-500,0]	[0.183012702,-0.683012702,-0.683012702,-0.183012702]	[-1,0,0,1]
	B4	[-600,-450,0]	[0.183012702,-0.683012702,-0.683012702,-0.183012702]	[-1,0,0,1]
	B5	[-600,-350,0]	[0.183012702,-0.683012702,-0.683012702,-0.183012702]	

Tabla 4.1.3 Posiciones de trabajo en la letra B

Letra	Nombre	Posición	Orientación	Configuración
“L”	L1	[-750,-250,0]	[0,0.965925826,0,0.258819045]	[-2,0,-1,1]
	L2	[-750,-450,0]	[0.183012702,-0.683012702,-0.683012702,-0.183012702]	[-2,-1,-2,0]
	L3	[-850,-250,0]	[0,0.965925826,0,0.258819045]	[-2,0,-1,1]

Tabla 4.1.4 Posiciones de trabajo en la letra L

Letra	Nombre	Posición	Orientación	Configuración
"O"	O1	[-1000,-200,0]	[0,0.965925826,0,0.258819045]	[-2,0,-1,1]
	O2	[-900,-300,0]	[0,0.965925826,0,0.258819045]	[-2,0,-1,1]
	O3	[-1000,-400,0]	[0,0.965925826,0,0.258819045]	[-2,0,-1,1]
	O4	[-1100,-300,0]	[0,0.965925826,0,0.258819045]	[-2,0,-1,1]

Tabla 4.1.5 Posiciones de trabajo en la letra O

4.1.4 Creación de trayectorias y procedimientos

Una vez definidos estos puntos, las trayectorias que se definen son las correspondientes a cada letra, incluyendo en todas ellas los puntos que se han definido para cada una, en el orden adecuado, en la Figura 4.1.4 se ilustran dichas trayectorias.

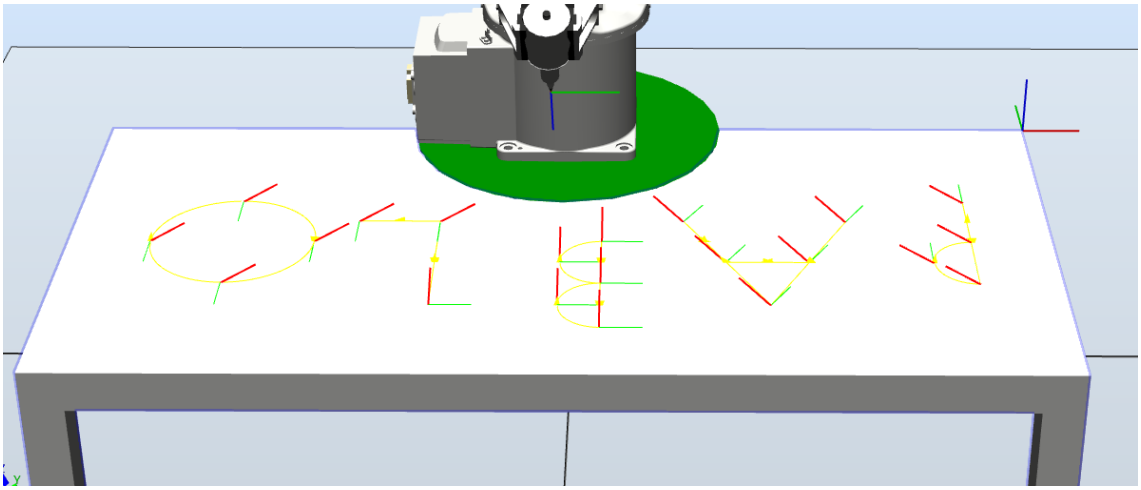

Figura 4.1.4 Trayectorias de las letras

En ellas se incluyen movimientos articulares para las aproximaciones a las letras, así como una combinación de movimientos lineales y circulares en caso de que sean necesarios. La precisión escogida para ellos ha sido la de "fine", para lograr una gran exactitud a la hora de reproducir las letras.

Por otro lado, se ha creado un punto de reposo, “Pos_Reposo”, al que acudirá el Robot una vez haya “Pintado” el nombre.

El procedimiento “Main”, mientras tanto, incluye todas las trayectorias ya mencionadas, así como dos instrucciones, “ConfL\on” y “SingArea\Wrist”, que facilitan la ejecución de movimiento de la forma precisa, incluso al pasar por una singularidad.

El simulador RobotStudio, tiene una función que permite marcar las trayectorias que sigue el TCP, llamada “Rastreo del TCP”. Por tanto, para simular que el robot “pinte” las trayectorias mencionadas previamente, se activará dicha función a la hora de realizar la simulación.

El problema de esta función, es que sigue todas las trayectorias que realiza el TCP, incluyendo las trayectorias de aproximación, que simulan el “levantar el bolígrafo del papel”. Como es lógico, estas trayectorias no se quieren pintar.

La solución elegida ha sido, la creación de una señal llamada “pinta”, que se activará cuando se quiera que pinte la trayectoria y se desactivará cuando no.

Esto es posible, ya que dicha función de RobotStudio permite activar el rastreo del TCP con la activación de señales, y por tanto pintará de un color las letras (azul), y de otro color las ya mencionadas trayectorias de aproximación (blanco).

Una vez realizada la simulación, el resultado final sería el mostrado en la Figura 4.1.5.

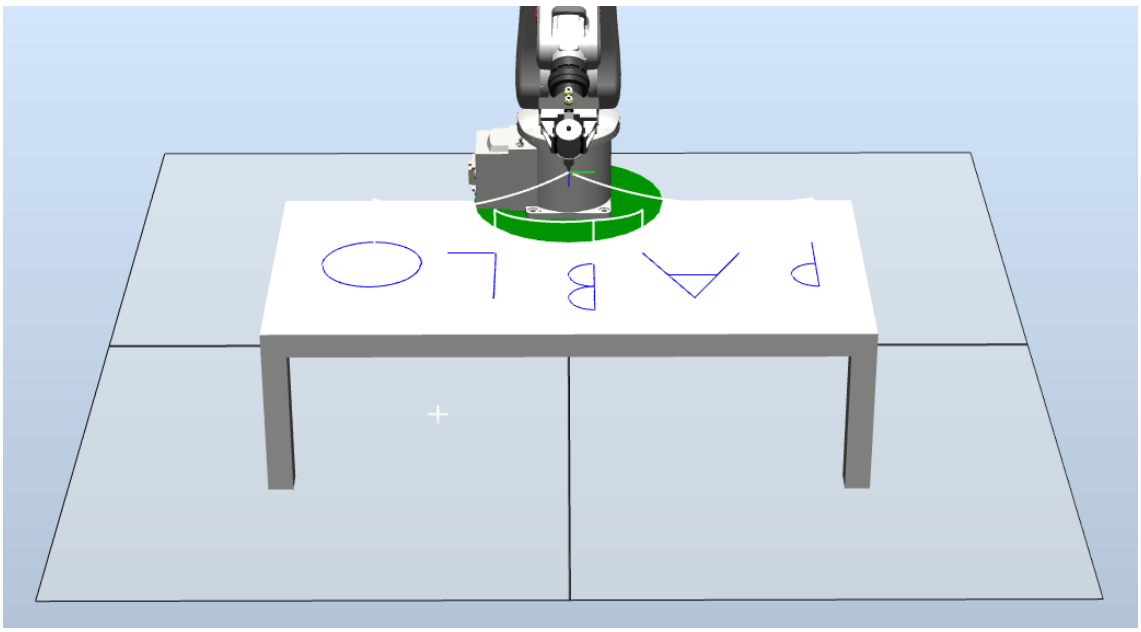

Figura 4.1.5 Nombre pintado

4.2 CREACIÓN HERRAMIENTA PINZA

RobotStudio, no tiene una herramienta tipo “pinza” entre sus librerías, por tanto, y dado el tipo de estaciones que se han creado en este trabajo, así como las prácticas que se han pensado para la asignatura, ha sido necesario crear varias herramientas de este tipo, combinando distintos tamaños, tanto de placas como de base (Figura 4.2.1).

A continuación se va a explicar la creación de una de ellas, la que se ha elegido para la realización del resto de escenarios, pero para el resto se ha seguido el mismo procedimiento.

4.2.1 Creación del mecanismo

Para comenzar, se ha creado la geometría de la misma, que en este caso consta de:

- Una primera pieza cilíndrica de 40mm de diámetro y 5mm de altura. Se ha escogido ese diámetro para hacerlo coincidir con la dimensión del eslabón 6 del Robot IRB 120.

- A continuación se ha creado encima de la misma, una pieza cilíndrica de mayor tamaño, de 30mm de diámetro y 20mm de altura.

- Para concluir lo que sería la base de la herramienta, se ha generado encima tetraedro de 50x100x25mm, sobre el que irán colocados los eslabones móviles.

Una vez se ha generado la geometría, se han agrupado las tres piezas que componen la base de la herramienta para tener una mayor comodidad al trabajar como una única pieza y se ha definido el origen local de la pieza como el centro de la primera pieza cilíndrica (Figura 4.2.2).

- Por último, en los extremos de esta última pieza, se han creado dos placas de 50x10x100mm, que constituirán las pinzas de la herramienta (Figura 4.2.3).

Actualmente, se tiene un conjunto de piezas, pero todavía no se comportan como una herramienta, para ello existe la opción de “Crear mecanismo”, en la cual indicaremos qué mecanismo queremos, si un robot, una herramienta, un eje externo, un dispositivo o un transportador.

Una vez seleccionada la opción de “herramienta”, se han indicado los eslabones que forman parte de dicha herramienta. En este caso:

- La “base” creada previamente. En este caso, se activará la opción de “Seleccionar como eslabón base”.
- La “placa 1” y la “placa 2”.

A continuación, se han creado los ejes de la herramienta. En este caso, esta herramienta constará de dos ejes prismáticos cuyo eslabón principal será el eslabón base creado con anterioridad, mientras que los eslabones secundarios serán ambas placas.

Los dos ejes tendrán de límite de movimiento mínimo 0mm, y de límite de movimiento máximo 65mm, ya que se quiere que lleguen justo hasta la mitad, pero sin tocarse.

Para conseguir que se muevan a la vez, el segundo eje no se ha seleccionado como activo y por tanto, se activará cuando se active el primero y por otro lado, en la pestaña dependencias, se ha creado una dependencia del eje 2, respecto del eje 1, con un factor de 1, es decir, cada milímetro que se mueva el eje 1 se moverá el eje 2.

Para concluir con esta primera parte de creación de herramienta, se le ha dado un nombre a la misma, "Pinza_analógica" en este caso, y se ha creado el sistema de coordenadas de la pinza. En este caso, al tratarse de una herramienta para coger objetos, el "eje z" de la misma ha de ir hacia dentro.

Ya se ha creado la herramienta como tal, pero para que dicha herramienta actúe de la forma que se quiere, coja los objetos y los deje, es necesario dotarla de "inteligencia".

4.2.2 Introducción a los SmartComponents

Antes de comenzar con la explicación del SmartComponent que se va a emplear para la creación de la pinza, se va a introducir a los mismos, ya que son una parte muy importante de este simulador y permite crear una gran variedad de situaciones distintas.

Dichos SmartComponents, o Componentes inteligentes, son elementos asociados a los sólidos, herramientas o robots de la estación. Estos elementos tienen un comportamiento controlado por señales y propiedades específicas del sistema., es decir, "dotan de inteligencia".

Los "SC", abreviación que se va a emplear a partir de ahora, tienen un conjunto de componentes subordinados, los cuales pueden emplearse para construir SC con un comportamiento más complejo. Están divididos en seis categorías, que son:

- **Señales y propiedades**: Dentro de esta categoría se encuentran elementos que permiten modificar una señal, o una propiedad del sistema a programar, como pueden ser Puertas Lógicas, Contadores, Temporizadores, Expresiones Matemáticas, Convertidores, Comparadores y Repetidores, entre otros.

- **Primitivos paramétricos**: Estos componentes subordinados son los necesarios cuando se quiere crear de forma automática sólidos y/o líneas, así como generar copias de componentes gráficos ya existentes.
- **Sensores**: Aquí se encuentran los componentes, o sensores, que se pueden emplear para detectar objetos. Existen varios tipos de sensores, como pueden ser sensores de colisión, de línea, de superficie, volumétricos, de posición u objeto más cercano.
- **Acciones**: Entre estos componentes se incluyen los que realizan acciones a objetos, o entre objetos, como pueden ser conectarse, desconectarse, copiar un objeto, eliminarlo o hacer visible/invisible.
- **Manipuladores**: Este apartado reúne los componentes necesarios para mover un objeto de forma lineal, hacer rotar un objeto, moverse a lo largo de una curva o posicionarlo en un lugar determinado. Por otro lado, también se incluyen los que permiten mover los ejes de un mecanismo, herramienta o robot, a una posición elegida.
- **Otros**: En este último apartado se encuentran una gran variedad de complementos subordinados, que realizan varias funciones distintas, como puede ser: Representar una cola de objetos, generación de un número aleatorio, detención de la simulación, reproducción de sonido, etc.

Se quiere añadir que el idioma por defecto para la edición de los SC es el inglés, independientemente del idioma de la aplicación, aunque puede ser modificado dentro del mismo.

Con esto se quiere mostrar la cantidad de opciones que permite esta herramienta de RobotStudio, y aclarar cómo se ha realizado el SC de la herramienta pinza.

4.2.3 Creación del SmartComponent

Para ello, se va a crear un “SC”, de nombre “SC_PinzaAnalogica”. Este componente inteligente, tendrá una “LogicGate”, un “PlaneSensor”, dos “JointMover”, un “Attacher”, un “Detacher, un “JointSensor”, una “Expression” y un “Converter”.

Además tendrá una entrada y una salida digital, “cierra” (DI), para indicar que se inicie el proceso de cerrar la pinza y “pillada”(DO), para indicar que se tiene la pinza, y una salida analógica “AnchoPieza” (AO), para indicar el tamaño que tiene la pieza que se está cogiendo.

En la Figura 4.2.4 Se indica las conexiones que se realizan entre los distintos componentes, así como con sus entradas y sus salidas. No obstante, para aclarar la función de cada uno de ellos, se va a explicar uno a uno:

- **LogicGate**: Esta puerta lógica, será una puerta NOT, la cual tendrá un único Input, que será la señal digital de entrada "Cierra", y un único Output, que estará conectado al JointSensor "AbrePinza".

- **PlaneSensor**: Este PlaneSensor, se ha colocado en la "placa 1", pero 0.01 mm separado hacia el interior de la pinza, para que cuando se active dicho sensor no detecte la "placa 1", sino que detecte la pieza que nos interesa.

  Este sensor se activará cuando la señal digital de entrada, "cierra", pase a estar activa, y detectará la pieza que en ese momento esté cruzando el plano.

  Una vez activada, enviará una señal al "JointMover CierraPinza", para que deje de cerrarse. Además, enviará otra señal al "Attacher" para que se ejecute en ese instante.

- **JointMover**: Este SmartComponent tendrá dos JointMover, "CierraPinza" y "AbrePinza", que como sus nombres indican, ejecutarán las acciones de cerrar ambas placas o abrirlas.

  Para ello, en sus parámetros se ha seleccionado como mecanismo al que pertenecen el "SC_PinzaAnalogica", que el movimiento tenga una duración de 1 segundo, y en el primero que desplace el eje 1 hasta la posición 55mm, mientras que en el segundo desplazará dicho eje hasta 0mm.

  "CierraPinza" se ejecutará cuando la señal digital de entrada, "cierra", pase a estar activa. Por otro lado, como se ha comentado previamente, cuando el sensor detecte un objeto, este JointMover detendrá el movimiento del eje.

- **Attacher**: Es el elemento encargado de unir, o "pegar", la pieza que se quiere coger a las placas de la pinza. Para ello, se ha unido, el objeto detectado por el sensor al "child", o pieza a unir, del attacher y como "parent", es decir pieza a la que se quiere unir otra, la placa 1 de la herramienta.

  Por otro lado, como se ha explicado anteriormente, este componente se ejecutará, como es lógico, cuando el sensor haya detectado un objeto. Y además, cuando se haya ejecutado, activará la señal digital de salida, "pillada", con la que se indicará que se ya se ha producido la unión.

- **Detacher**: Este componente se encarga de lo opuesto al anterior, es decir, rompe la unión entre los dos elementos, por tanto, el "child" de este componente se ha unido con el "child" del attacher, para asegurarnos de que es la misma pieza.

  Este componente se activará, mientras se esté ejecutando la acción de abrir la pinza, ya que en ese momento es cuando queremos que dejen de estar unidos.

- **JointSensor**: Se trata del elemento encargado de medir la distancia que se mueven los ejes, y por tanto estará actualizándose mientras se están ejecutando las acciones de abrir y cerrar las pinzas. Su salida estará conectada al componente de expresión matemática.

- **Expression**: Este componente, se trata de una expresión matemática que tiene como variable la salida del JointSensor, y como fin medir el tamaño de la pieza que se está cogiendo.

$$Expression = 90 - (JointSensor * 2000)$$

  Dicha fórmula tiene esa forma ya que, el JointSensor, únicamente mide el movimiento de un eje y éste únicamente se mueve de 0 a 45, y además, está medido en micras.

- **Converter**: El resultado de la expresión, hay que convertirlo en señal analógica, y para ello se ha utilizado un Converter. La salida de este componente se configura como salida analógica y se conecta directamente con la señal analógica de salía, AO.

Con esto se ha terminado la herramienta tipo pinza que se necesitaba para completar las demás estaciones, y por tanto para realizar las prácticas.

Se trata de un fichero ".RSLIB", y se ha guardado con el nombre de "SC_PinzaAnalogica" en las librerías de usuario disponibles en los ordenadores del Laboratorio 0.06 del edificio Ada Byron.

4.3 ESCENARIO DEMOENSAMBLADO

En este escenario, el robot IRB 120 cogerá las piezas indicadas con la herramienta pinza, y las moverá siguiendo trayectorias lineales a su posición de destino. Dichas piezas, colocadas correctamente en el destino, simulan la unión de varias piezas mediante pernos.

El escenario “Demoensamblado”, se ha creado con la intención de utilizarlo como práctica de simulación. En la Figura 4.3.1 se muestra la estación completa, con las trayectorias, sistemas de coordenadas y objetos creados para la misma.

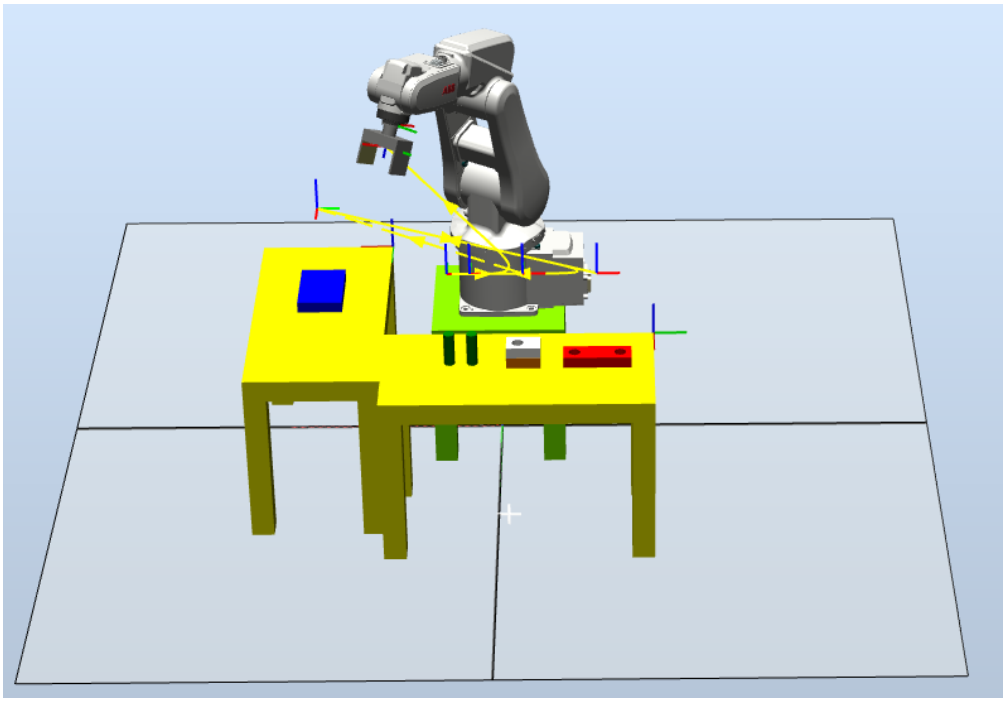

Figura 4.3.1 Estación Demoensamblado

Por otro lado, en dicho escenario, se pretende trabajar con la activación de señales, los tiempos de espera, la utilización y diferenciación entre puntos de esquina y puntos de paso, así como con la precisión del Robot en sus movimientos.

4.3.1Creación del escenario

Este escenario, está compuesto por una mesa sobre la que irá colocado el robot IRB 120, y dos mesas auxiliares de mayor tamaño, sobre las que se colocarán las piezas, y se montarán en su posición correspondiente, como muestra la Figura 4.3.2.

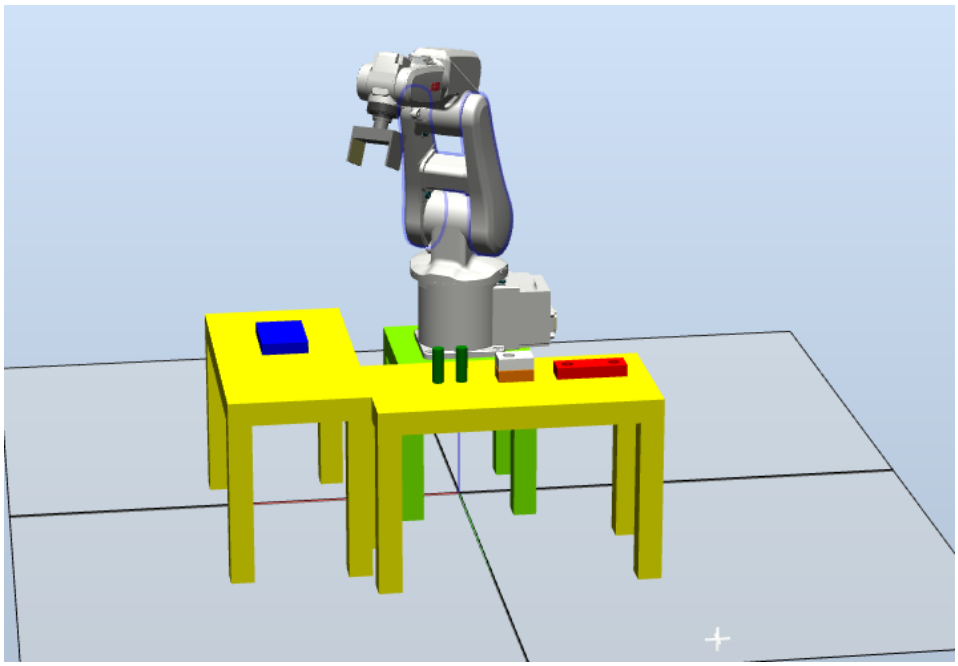

Figura 4.3.2 Creación de la estación

La mesa del robot, “Mesa_Robot”, se trata de una mesa cuadrada, de 300x300x50mm en el extremo superior (Figura 4.3.3).

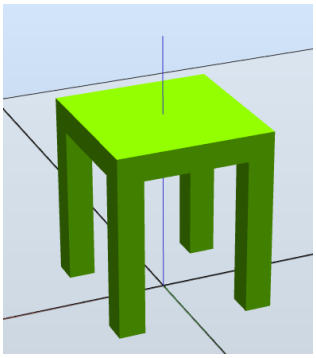

Figura 4.3.3 Mesa soporte del Robot

Las dos mesas auxiliares amarillas, son de mayor tamaño, 300x600x50 la “Mesa_1” (Figura 4.3.4) y 600x300x50 la “Mesa_2” (Figura 4.3.5).

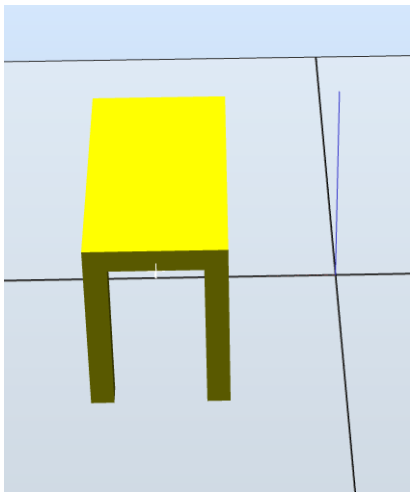

Figura 4.3.4 Mesa soporte para ensamblaje

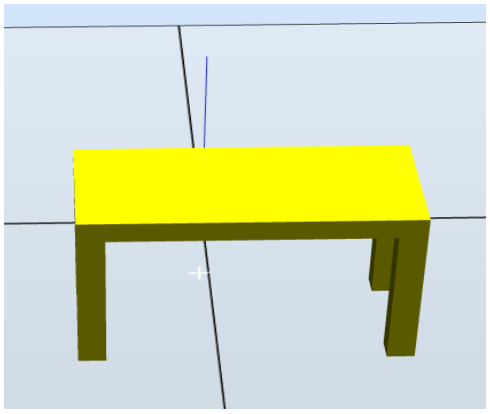

Figura 4.3.5 Figura Mesa soporte para piezas

Por último, se han creado las piezas que se van a ensamblar, las cuales tienes las siguientes dimensiones:

- **Junta** (Figura 4.3.6): Esta pieza tiene 150x50x25mm, como dimensiones principales. Los agujeros para los pernos son de 25mm de diámetro, y su centro se encuentra separado 25mm de los laterales.

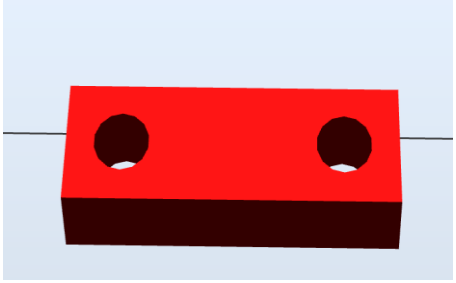

Figura 4.3.6 Junta

- **Piezas 1 y 2** (Figura 4.3.7): Ambas tienen 75x50x25mm de dimensiones principales. La diferencia entre ambas es la orientación que tienen, ya que una se ha colocado para que tenga el agujero para el perno a la izquierda, y sin embargo, la otra lo tendrá en el lado derecho. Ambos centros se encontrarán a 25mm del lateral y tendrán 25mm de diámetro.

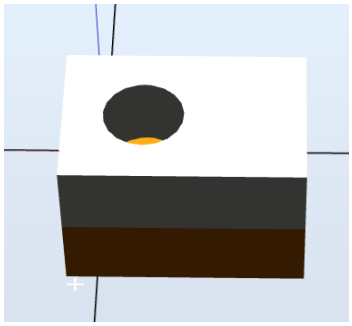

Figura 4.3.7 Piezas 1 y 2.

- Pernos 1 y 2 (Figura 4.3.8): Estos pernos, son dos cilindros macizos de 24mm de diámetro y 75 mm de altura.

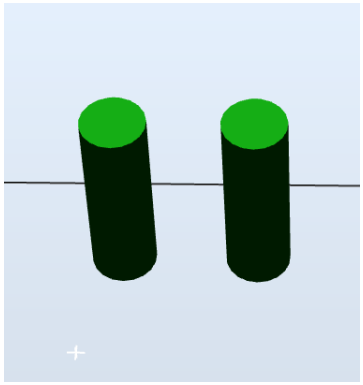

Figura 4.3.8 Pernos 1 y 2.

- “Piezagrande” (Figura 4.3.9): Esta pieza, es la única inmóvil, ya que va a representar una plataforma sobre la que se ensamblarán los elementos anteriores. Se trata de un tetraedro de 100x150x25mm.

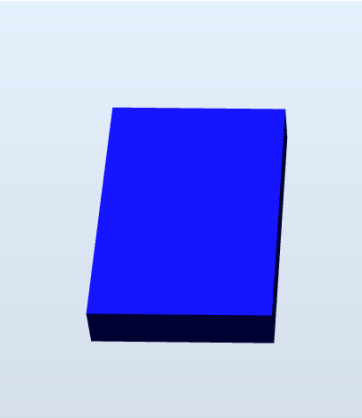

Figura 4.3.9 Pieza soporte para ensamblado

5.3.2 Creación de objetos de trabajo y posiciones

Una vez se ha creado el escenario, se han definido los objetos de trabajo que se van a emplear, y a partir de ellos, se han definido las posiciones a las que acudirá el robot en sus respectivas trayectorias.

En este escenario, se han definido dos objetos de trabajo, o "wobjdata", "Mesa_1" y "Mesa_2". Ambos, son objetos de trabajo que no están sostenidos por el robot, por tanto constan de un sistema de coordenadas de usuario fijo. La posición de los sistemas de coordenadas de usuario y de objeto de ambos objetos, con respecto al sistema de coordenadas de la base del Robot, es la que sigue:

- **"Mesa_1"**: Su sistema de coordenadas de usuario se encuentra en [250,-150,50], y con una orientación, expresada como cuaternio, de [1,0,0,0], es decir, la misma orientación que la base. El sistema de coordenadas del objeto se ha definido en el mismo punto.
- **"Mesa_2"**: Su sistema de coordenadas de usuario se encuentra en [-350,250,50], y con una orientación, expresada como cuaternio, de [0.707106781,0,0,0.707106781], es decir, tendrá un giro de 90 grados en el eje Z con respecto a la base. De nuevo, el sistema de coordenadas del objeto se ha definido en el mismo punto.

Por defecto, en RobotStudio y en RAPID, se genera un objeto de trabajo llamado "wobj0", que se encuentra en la base del robot y con la misma orientación que su sistema de coordenadas.

Para una mayor aclaración se dispone del Anexo A, apartado A.1.1, en el cuál se explica con precisión la definición de los objetos de trabajo, y la dependencia relativa de los distintos sistemas de coordenadas que se utilizan.

Como se ha comentado previamente, a partir de estos objetos de trabajo, se definen las posiciones de robot, o "robtargets". Por tanto en la siguiente Tabla 4.3.1 se muestran las posiciones de esos puntos, respecto a su objeto de trabajo:

Objeto de trabajo	Nombre	Posición	Orientación	Configuración
Mesa_1	Piezagrande	[150,225,225]	[0.707106781,0,0,0.707106781]	[0,0,-1,0]
Mesa_2	Junta	[125,-125,225]	[0.707106781,0,0,0.707106781]	[1,0,-1,0]
	Piezas	[125,-285.617,225]	[0.707106781,0,0,0.707106781]	[1,0,-1,0]
	Perno1	[125,-400,225]	[0.707106781,0,0,0.707106781]	[0,0,-2,0]
	Perno2	[125,-450,225]	[0.707106781,0,0,0.707106781]	[0,0,-2,0]

Tabla 4.3.1 Posiciones de trabajo en Demoensamblado

5.3.3 Lógica de la estación

Para la realización de esta estación, se ha empleado la herramienta pinza descrita previamente en este trabajo, con su correspondiente SC.

Además, se han creado 3 señales distintas, "cerrar_pinza", "pillada" y "AnchoPieza", que se explican a continuación:

- cerrar_pinza: Es una señal digital de salida, que se conectará a la señal digital de entrada "cierra" perteneciente al "SC_Pinza_Analogica".
- "pillada": Es una señal digital de entrada, que se conectará a la señal digital de salida "pillada", que de nuevo pertenece al "SC_Pinza_Analogica".
- "AnchoPieza: Es una señal analógica de entrada, conectada a la señal analógica de salida del "SC_Pinza_Analogica".

En la Figura 4.3.10 se muestran estas conexiones de manera gráfica.

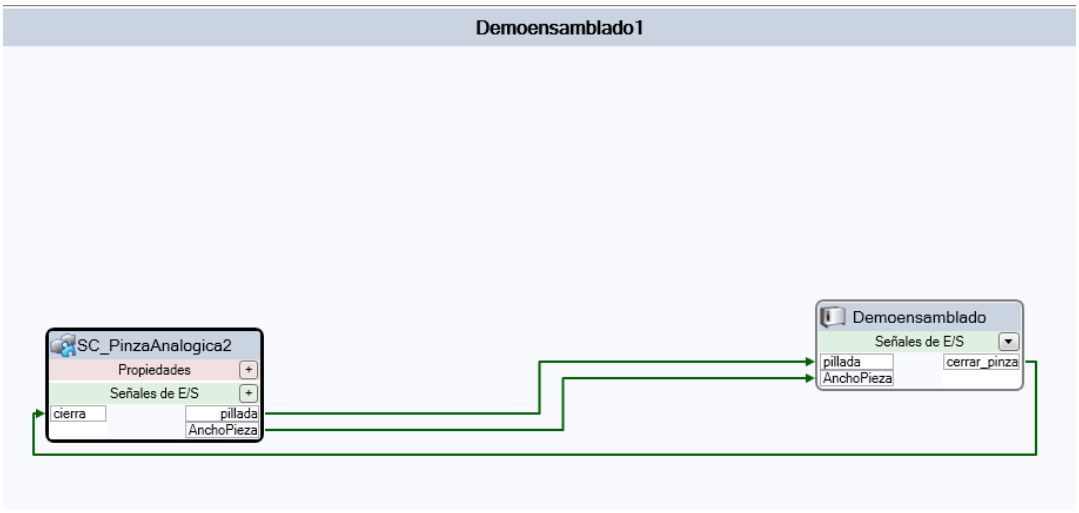

Figura 4.3.10 Lógica de la Estación

Por otro lado, a la hora de realizar la simulación, se ha activado un detector de colisiones. Dicho detector, se activará y coloreará en rojo las piezas que estén colisionando, si hubiese alguna. Por otro lado, este detector también tiene activada la opción de detectar "casi colisiones", con una distancia mínima de 3 mm. El color indicador de "casi colisión" será el amarillo, y como es lógico se activará a la hora de introducir los pernos en su posición final.

4.3.4 Creación de trayectorias y procedimientos

Para conseguir montar y desmontar las piezas, se ha realizado mediante doce procedimientos, entre los que se encuentran el procedimiento principal, o "main", así como montar y desmontar cada uno de los elementos de ensamblado y por último ir a una posición de reposo, situada en una posición cómoda para el robot.

El procedimiento "main" incluye dos comandos, como "ConfL \on" y "SingArea\Wrist", que ambos ayudan a la correcta realización de las trayectorias, siguiendo las configuraciones y orientaciones indicadas. En el Anexo A, apartado A.2.4, se encuentra una explicación más detalla de lo que realizan ambos comandos. Por otro lado, en este procedimiento se incluirán las llamadas al resto de procedimientos.

Los otros procedimientos de montaje y desmontaje, incluyen trayectorias articulares para los movimientos de desplazamiento de piezas hasta los puntos de aproximación, así como trayectorias lineales para acudir desde los puntos de aproximación hasta los puntos de destino finales. Estos puntos de aproximación, son puntos relativos a los puntos de destino, que se

encuentran a poca distancia en el eje Z, y sirven para generar trayectorias lineales, más lentas y con mayor precisión, para que el robot lleve cada pieza a su destino final. Para llevar al robot a posiciones relativas a una posición de destino, se ha empleado la función “offs”, que se encuentra más detallada en el Anexo A, apartado A.1.3.

Por tanto, la secuencia de montado y desmontado, quedaría como indican las Figura 4.3.11 y 4.3.12, respectivamente.

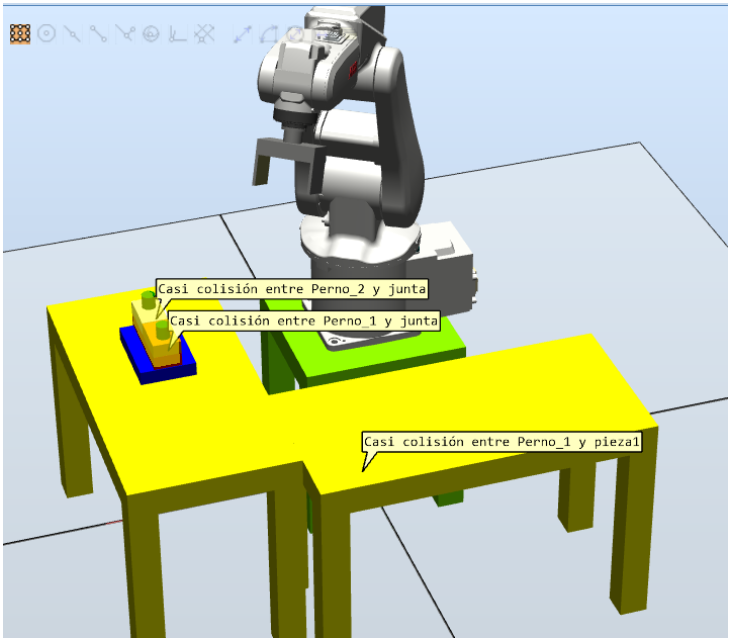

Figura 4.3.11 Ensamblaje montado

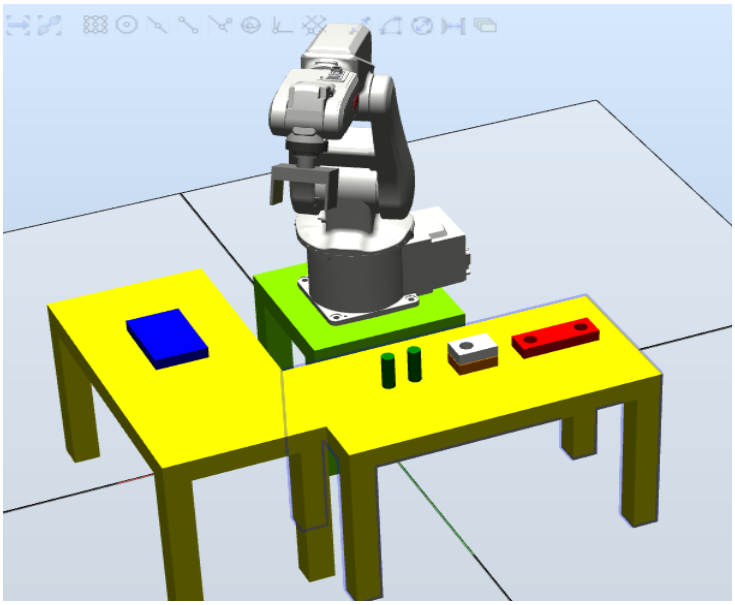

Figura 4.3.12 Ensamblaje desmontado

4.4 Estación “Demotren”

En este escenario, el Robot IRB 120 cogerá las piezas indicadas mediante la herramienta pinza, y las colocará de la forma correcta en su posición, simulando la recreación de un vagón de tren.

Se ha escogido esta estación como práctica, ya que de nuevo se tiene que trabajar con la activación de señales, los tiempos de espera, la utilización y diferenciación de los puntos de esquina y puntos de paso, así como mostrar la precisión del robot en sus movimientos.

Además de lo anterior, dado que se dispone del “vagón de tren” original en el laboratorio físicamente, Figura 4.4.1, en el momento en el que se disponga de una herramienta tipo pinza real, los alumnos podrán reproducir esta simulación de forma segura y controlada en el laboratorio, si el profesor así lo quiere.

Figura 4.4.1 Maqueta del tren

4.4.1 Creación del escenario

Para la creación del escenario, se ha colocado el Robot IRB 120 sobre la ya mencionada “Mesa_Robot” y las piezas que se van a explicar a continuación, sobre la “Mesa_Piezas”, ambas importadas desde las bibliotecas de usuario. Por tanto el escenario inicial sería el indicado en la Figura 4.4.2.

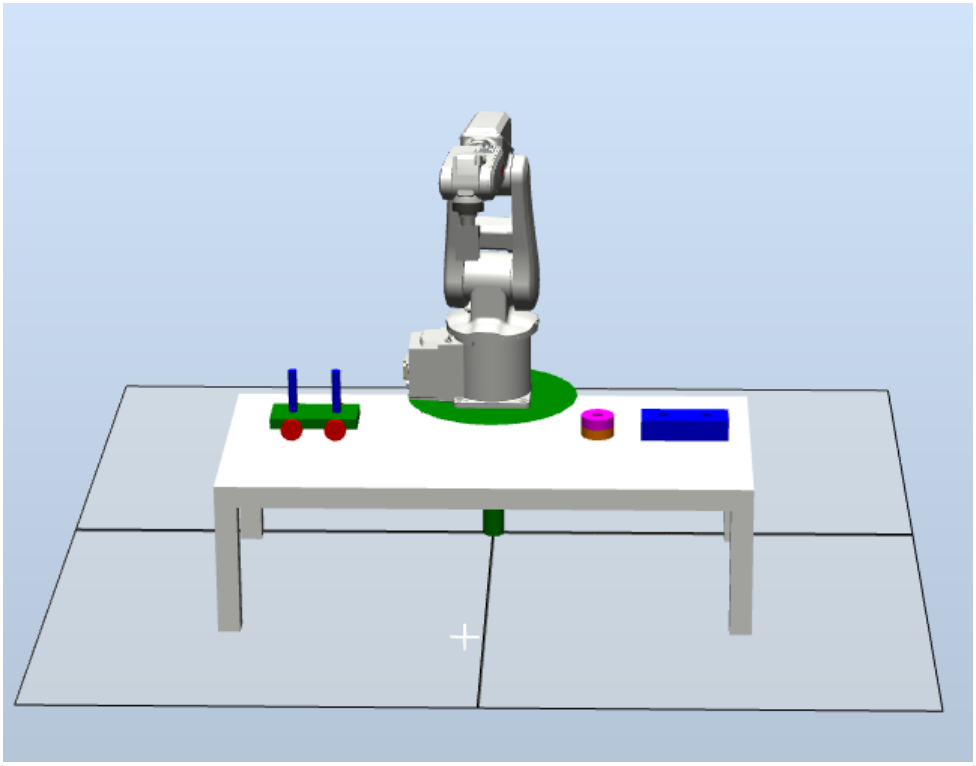

Figura 4.4.2 Creación del escenario

Las piezas que se van a crear, encima de la correspondiente mesa, serán las siguientes:

- **Pieza** (Figura 4.4.3): Se trata de un objeto rectangular, con dimensiones 200x75x50mm. Los dos agujeros que se observan, tienen diámetro 24mm, para que pueda ser introducida en el tren.

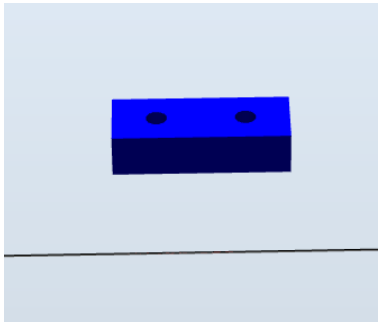

Figura 4.4.3 Pieza

- **Cilindro1 y 2** (Figura 4.4.4): Los cilindros constan de 75mm de diámetro externo, así como de 24mm de diámetro interno, para que puedan ser introducidos en el tren. Por otro lado, ambos tienen 25mm de espesor.

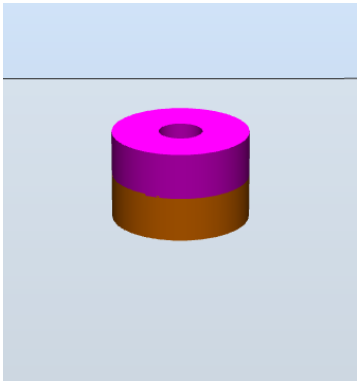

Figura 4.4.4 Cilindros 1 y 2

- **Tren** (Figura 4.4.5): El tren, se ha creado a partir de la unión de varios elementos. Las cuatro ruedas son cuatro cilindros macizos de 50mm de diámetro, la base del tren se trata de un rectángulo de 200x100x25mm y por último los dos postes son dos cilindros de 20mm de diámetro y 100mm de altura.

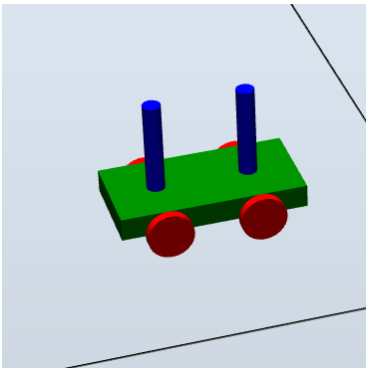

Figura 4.4.5 Tren

4.2.2 Creación de objetos de trabajo y posiciones

Una vez se ha creado el escenario, se han definido los objetos de trabajo a partir de los cuáles se han creado las posiciones del robot posteriores.

En este caso, se han definido dos objetos de trabajo, “Sistema_Piezas y Sistema_tren”, que se han definido de forma que son objetos de trabajo no sostenidos por el robot, y por tanto tienen un sistema de coordenadas de usuario fijo. La posición y la orientación de dicho sistema de coordenadas, así como la del sistema de coordenadas de objeto, con respecto a la base del robot, es la siguiente:

- **Sistema_piezas**: Su sistema de coordenadas de usuario se encuentra en la posición $[600,0,0]$, mientras que la orientación no cambia con respecto a la base del robot,

y por tanto, expresada en forma de cuaternio, sería [1,0,0,0]. Por otro lado, el sistema de coordenadas de objeto está definido en el mismo punto.

- **Sistema_tren**: En este caso, su sistema de coordenadas de usuario se encuentra en el otro extremo de la "Mesa_Piezas", en la posición [-600,0,0] y con un giro de 90º en el eje Z con respecto a la base de coordenadas del robot. Por tanto, expresado en forma de cuaternio será, [0.707106781,0,0,0.707106781]. De nuevo, el sistema de coordenadas de objeto se ha definido en el mismo punto.

Como en estaciones anteriores, en RobotStudio y en RAPID, se genera un objeto de trabajo llamado "wobj0", que se encuentra en la base del robot y con la misma orientación que su sistema de coordenadas.

Para una mayor aclaración se dispone del Anexo A, apartado A.1.1, en el cuál se explica con precisión la definición de los objetos de trabajo, y la dependencia relativa de los distintos sistemas de coordenadas que se utilizan.

Una vez definidos estos objetos de trabajo, las posiciones de robot, o "robottargets", que se definen a partir de ellos son los siguientes, como indica la Tabla 4.4.1

Objeto de trabajo	Nombre	Posición	Orientación	Configuración
Sistema_piezas	Cilindros	[-350,-262.5,137.5]	[1,0,0,0]	[-1,0,-1,0]
	Piezas	[-150,-262.5,125]	[1,0,0,0]	[-1,0,-1,0]
Sistema_tren	Tren	[-250,-200,75]	[0.707106781,0,0,-0.707106781]	[-2,0,-2,0]

Tabla 4.4.1 Posiciones de trabajo en Demotren

4.2.3 Lógica de la estación

Para la realización de esta estación, se ha empleado la herramienta pinza descrita previamente en este trabajo, con su correspondiente SC.

Además, se han creado 3 señales distintas, "cerrar_pinza", "pillada" y "AnchoPieza", que se explican a continuación:

- cerrar_pinza: Es una señal digital de salida, que se conectará a la señal digital de entrada “cierra” perteneciente al “SC_Pinza_Analogica”.
- “pillada”: Es una señal digital de entrada, que se conectará a la señal digital de salida “pillada”, que de nuevo pertenece al “SC_Pinza_Analogica”.
- “AnchoPieza: Es una señal analógica de entrada, conectada a la señal analógica de salida del “SC_Pinza_Analogica”.

En la Figura 4.4.6 se da una explicación gráfica de cómo serían dichas conexiones.

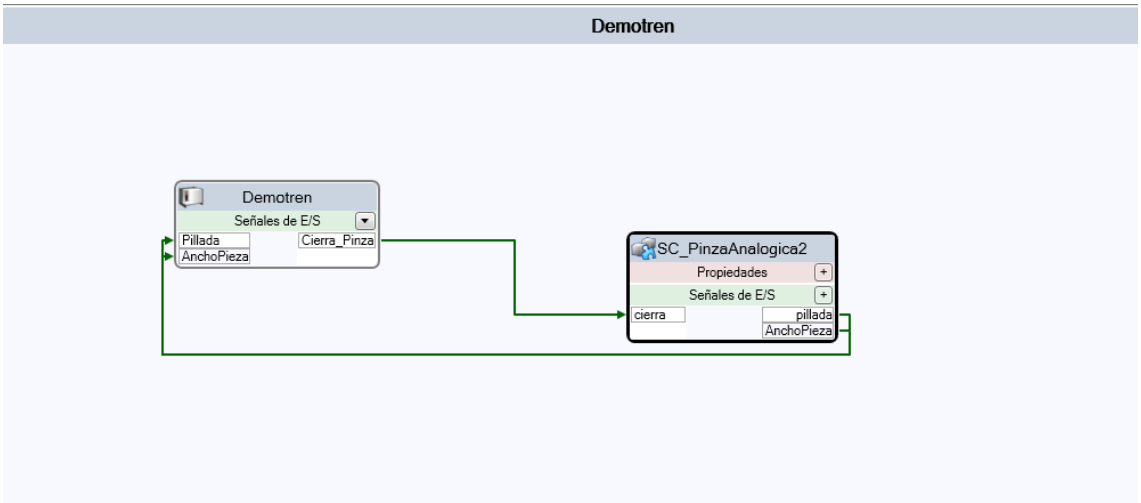

Figura 4.4.6 Lógica de la estación Demotren

Por otro lado, a la hora de realizar la simulación de nuevo, se ha activado un detector de colisiones. Dicho detector, se activará y coloreará en rojo las piezas que estén colisionando, si hubiese alguna. Por otro lado, del mismo modo que en la estación anterior, este detector también tiene activada la opción de detectar “casi colisiones”, con una distancia mínima de 3 mm. El color indicador de “casi colisión” será el amarillo, y como es lógico se activará a la hora de introducir la pieza y los cilindros en la maqueta del tren.

4.2.4 Creación de trayectorias y procedimientos

Para conseguir montar y desmontar las piezas, se ha realizado mediante doce procedimientos, entre los que se encuentran el procedimiento principal, o “main”, así como montar y desmontar cada uno de los elementos de ensamblado y por último ir a una posición de reposo, situada en una posición cómoda para el robot.

El procedimiento “main” incluye dos comanos, como “ConfL \on” y “SingArea\Wrist”, que ambos ayudan a la correcta realización de las trayectorias, siguiendo las configuraciones y

orientaciones indicadas. En el Anexo A, apartado A.2.4, se encuentra una explicación más detalla de lo que realizan ambos comandos. Por otro lado, en este procedimiento se incluirán las llamadas al resto de procedimientos.

Los otros procedimientos de montaje y desmontaje, incluyen trayectorias articulares para los movimientos de desplazamiento de piezas hasta los puntos de aproximación, así como trayectorias lineales para acudir desde los puntos de aproximación hasta los puntos de destino finales. Estos puntos de aproximación, son puntos relativos a los puntos de destino, que se encuentran a poca distancia en el eje Z, y sirven para generar trayectorias lineales, más lentas y con mayor precisión, para que el robot lleve cada pieza a su destino final. Para llevar al robot a posiciones relativas a una posición de destino, se ha empleado la función “offs”, que se encuentra más detallada en el Anexo A, apartado A.1.3.

Por tanto, la secuencia de montado y desmontado, quedaría como indican las Figura 4.4.7 y 4.4.8, respectivamente.

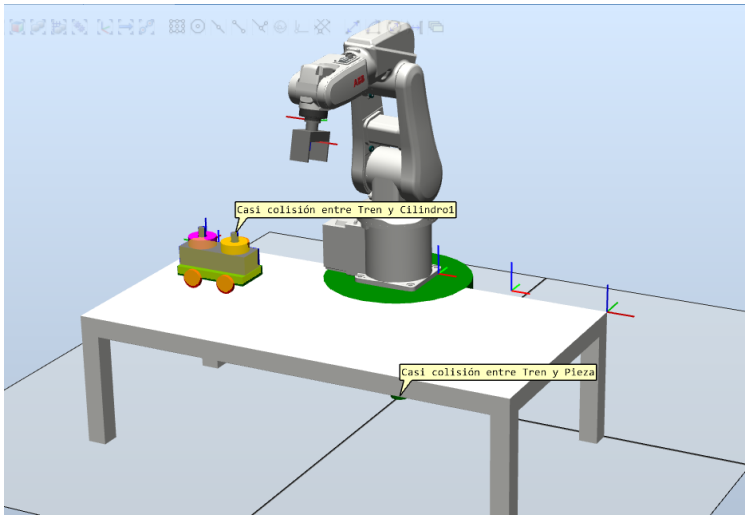

Figura 4.4.7 Tren montado

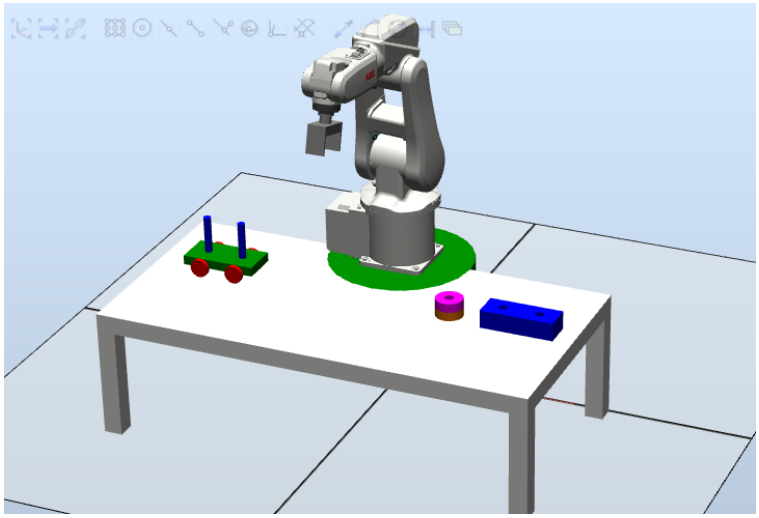

Figura 4.4.8 Tren desmontado

Capítulo 5

Trabajo de simulación Multi-Robot

En este capítulo se va a explicar cómo trabajar con un sistema de dos robots, moviéndose y ejecutando instrucciones de manera sincronizada mediante la función Multimove. Así como explicar cómo se ha resuelto el reto de ABB "Fanta Can Challenge", que se basa en dicho movimiento.

5.1 Movimiento coordinado – Multimove

Multimove es una aplicación interna de RobotStudio, que le permite crear y optimizar programas para sistemas de varios robots y/o unidades mecánicas, coordinados por un mismo controlador**.** En dichos sistemas, un robot objeto o posicionador sostiene la pieza de trabajo, mientras que los otros robots, que a partir de ahora se llamarán robots herramienta, trabajan en ella.

A continuación se va a explicar el flujo de trabajo principal para la programación de sistemas Multimove con RobotStudio, incluyendo los requisitos previos, la configuración de Multimove, la comprobación de Multimove y el ajuste del comportamiento de los movimientos.

Por otro lado, también se puede programar manualmente el sistema Multimove, mediante una combinación de las herramientas de programación normales y de un conjunto de herramientas específicas para la programación de Multimove. Estas herramientas pueden ser, listas de tareas e identidades de sincronización, adición y actualización de argumentos de ID a las instrucciones a sincronizar, adición y ajuste de instrucciones de sincronización a las trayectorias y programación de instrucciones de Multimove.

5.1.1 Requisitos previos

Para utilizar las funciones de Multimove, previamente se ha de disponer de lo siguiente:

- Un controlador virtual en el que se ejecuta un sistema Multimove iniciado en RobotStudio. Es decir, a la hora de crear el controlador virtual, se ha de activar la opción de "MultimoveCoordinated", así como la opción de Multitarea, es decir, permitir que el controlador sea capaz de realizar dos tareas simultáneamente.

- Todos los sistemas de coordenadas, objetos de trabajo y herramientas empleadas en dicho sistema se han de definir previamente.
- Las trayectorias sobre las cuales se va a mover la herramienta de trabajo, no el objeto. Dichas trayectorias han de crearse en un objeto de trabajo que pertenezca a un robot de herramienta, y esté fijado al robot objeto, que esté movido por esa unidad mecánica.

5.1.2 Configuración de Multimove

La aplicación Multimove dentro de RobotStudio se encuentra dentro de su pestaña "Posición Inicial", y al iniciarlo se abrirá una pestaña de asistente de configuración. Dicho asistente, tiene la función de ayudar a determinar qué Robots, o unidades mecánicas, trabajarán coordinadamente, cuál de ellos lleva el objeto de trabajo y cuál de ellos lleva la herramienta.

No obstante, estos parámetros podemos definirlos dentro del área de trabajo de Multimove, en la pestaña configuración. Una vez dentro, se puede indicar qué robots queremos programar, para dichos robots se permite especificar cuál de ellos sostiene la herramienta de trabajo, y cuál de ellos el objeto.

Por otro lado, dentro de esa misma pestaña, se puede especificar qué trayectorias del programa queremos activar para el "robot herramienta", así como el orden de dichas trayectorias.

5.1.3 Comprobación de Multimove

La comprobación de Multimove, ejecuta las instrucciones de movimiento a lo largo de las trayectorias y de acuerdo con los valores de la página de configuración de movimiento.

Dentro del área de trabajo de Multimove, se puede activar dicha opción de "probar", y una vez dentro de ellas simulará los movimientos a lo largo de las trayectorias, basándose en la posición de inicio actual o basándose a la primera posición de inicio que hayamos probado. Multimove permite, detenerse y verificar las trayectorias una vez éstas se hayan realizado, o por el contrario simular en bucle hasta que se haga clic en "pausa".

5.1.3 Ajuste del comportamiento de los movimientos

El ajuste del comportamiento de los movimientos en Multimove, implica la configuración de reglas para los movimientos del robot, como por ejemplo, limitaciones sobre la posición del

robot o la orientación de la herramienta. Multimove, por lo general, conseguirá movimientos más suaves, con tiempos de ciclo y proceso menores si se emplea el menor número posible de limitaciones.

Desde esta pestaña de Multimove, se puede modificar la influencia de los ejes, que a su vez controla el equilibrio del grado de utilización de dichos ejes por parte de los robots. Y por tanto, el aumento de la influencia de un eje fomentará su movimiento con respecto al resto de ejes alternativos, mientras que su disminución limitará el movimiento de dicho eje.

Otro parámetro que permite controlar es la restricción del movimiento del TCP, es decir, permite limitar para cada "pose", posición y orientación de un punto, el movimiento del TCP, así como darle un peso a esa limitación y cuánto más bajo sea ese peso, mayor será la limitación.

Por último, también se permite modificar el offset de la herramienta, es decir, modificar la distancia fija entre la herramienta y las trayectorias que se desee.

5.1.4 Acciones adicionales

Como se ha comentado previamente, se puede programar manualmente su Multimove mediante una combinación de herramientas de programación habituales para un único Robot y un conjunto de herramientas específicas para su programación.

Dichas acciones principales específicas, se describen a continuación:

- **Crear lista de tareas:** En el código RAPID, hemos de crear una variable tipo "tasks", para poder identificar las tareas que se sincronizarán. Por último, en cada instrucción "SyncMoveOn" o "WaitSyncTask" se especificará qué lista de tareas se desea utilizar. A continuación se muestra un ejemplo aclaratorio:

```
PERS tasks Task_list{2}:=[["T_ROB1"], ["T_ROB2"]];

....

SyncMoveOn SyncOn, Task_list;
```

- **Crear una identidad de sincronización:** De nuevo, en el código RAPID del programa, se ha de declarar una variable tipo "SyncIdent", para identificar las

instrucciones de sincronización que deben de estar sincronizadas. Por tanto, siguiendo el ejemplo anterior, el código correcto sería:

```
PERS tasks Task_list{2}:=[["T_ROB1"], ["T_ROB2"]];

VAR syncident SyncOn;

…

SyncMoveOn SyncOn, Task_list;
```

- **Adición y actualización de argumentos ID:** Una de las opciones que permite RAPID, es establecer nuevos argumentos de ID de sincronización en las instrucciones de movimiento de una trayectoria de Multimove, y por tanto, si ambas trayectorias tienen el mismo número de instrucciones de movimiento y sus respectivos argumentos de ID de identificación coinciden, podrán sincronizarse. A continuación se muestra un ejemplo de instrucción de movimiento con argumento ID incluido.

```
MoveJ Mesa1,\ID:=1,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
```

Éstas serían las formas que RobotStudio permite para el trabajo coordinado de varios robots y/o unidades mecánicas, un máximo de cuatro, controladas por un mismo controlador. Aclarar que para la estación FantaCan que se va a explicar a continuación, se ha seguido el segundo método, ya que es la mejor manera de conocer realmente que instrucciones y tareas estarán sincronizadas.

5.2 Estación FantaCan

Como se ha comentado previamente, esta estación se ha planteado gracias al “Fanta Can Challenge” lanzado por la propia ABB, ya que es una forma más visual de retar al alumnado a implementar acciones más difíciles como sincronizar el movimiento de dos robots.

5.2.1 Creación del escenario

Para este escenario, se han empleado dos robots modelo IRB 120, la “Mesa_Robot” comentada previamente para que haga de soporte de uno de los dos Robots, como muestra la Figura 5.2.1, la herramienta “MyTool” y herramienta que hace de trabajo “Mesa_Latas”.

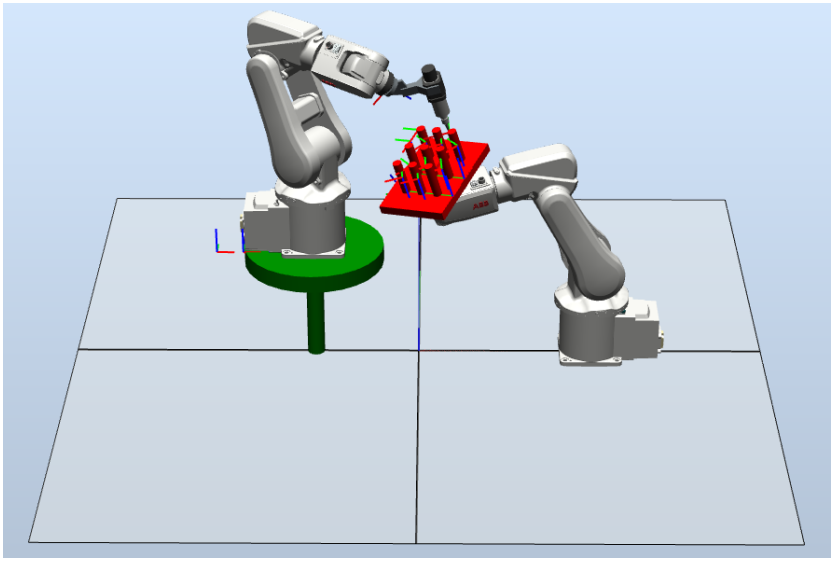

Figura 5.2.1 Estación FantaCan

La herramienta “MyTool, es una herramienta de trabajo similar a la herramienta empleada anteriormente, “MyPen”, y también se encuentra disponible en las bibliotecas oficiales de RobotStudio.

Sin embargo, la herramienta “Mesa_Latas” se muestra en la Figura 5.2.2, y se trata de una mesa cuadrada de 300*200, que posee nueve “latas”, cilindros de 25mm de diámetro y 75mm de altura, situadas a una distancia de 75mm en el eje X, y 50mm en el eje Y.

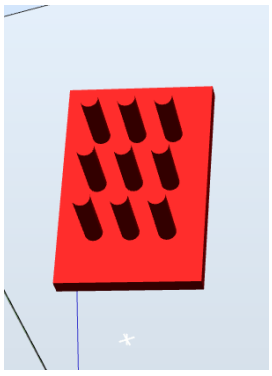

Figura 5.2.2 Mesa soporte de las latas

Por otro lado, como se ha comentado previamente, a la hora de definir dichas herramientas es muy importante determinar cuál es la pieza de trabajo sostenida por el robot, y cuál es la herramienta manipuladora de dicha pieza. En este caso, “MyTool” estará en el ROB_1, a partir de ahora denominado “Robot herramienta”, mientras que “Mesa_latas” estará sostenida por el ROB_2, que a partir de ahora se denominará “Robot objeto”.

5.2.2 Creación de objetos de trabajo y posiciones

Para la realización de esta estación, ha sido necesaria la creación de dos objetos de trabajo, “Mesa_Latas1” y “Mesa_Latas2”, ambos situados en el centro de la base de la herramienta “Mesa_Latas”. Están referenciados al Robot herramienta y al Robot objeto respectivamente, y se han definido de la siguiente forma:

- **MesaLatas1:** Este objeto de trabajo se ha definido de forma que el Robot herramienta no esté sosteniendo el objeto de trabajo, es decir, esté sosteniendo la herramienta. También cuenta con un sistema de coordenadas de usuario móvil, necesario para sistemas de movimiento coordinado como este. Por otro lado, se ha indicado cuál es unidad mecánica con la que se han de coordinar los movimientos del robot, en este caso “ROB_2”. Y por último, se ha especificado la posición y orientación, expresada en cuaternios, del sistema de coordenadas de objeto respecto del sistema de coordenadas de usuario. En este caso, la posición será [0,0,25] y la orientación [1,0,0,0]. A continuación se muestra en código RAPID, como se ha definido:

```
PERS wobjdata MesaLatas1:=[FALSE,FALSE,"ROB_2",[[0,0,0],[1,0,0,0]],[[0,0,25],[1,0,0,0]]];
```

- **MesaLatas2:** Este objeto de trabajo, de nuevo se ha definido indicando que sostiene la herramienta, pero en este caso su sistema de coordenadas de usuario con respecto a la base es fijo, situado en un punto arbitrario. Al tener un sistema de coordenadas de usuario móvil, no está sostenido por ninguna unidad mecánica, por lo que no se ha especificado. El sistema de coordenadas de objeto, se ha definido en un punto arbitrario de nuevo, pero accesible para ambos robots. A continuación se muestra en código RAPID, cómo se ha definido:

```
PERS wobjdata MesaLatas2:=[FALSE,TRUE,"",[[548.481,-115.705,297.178],[0.851014511,-
0.061555277,-0.282579821,0.438330805]],[[-34.964,140.479,-52.347],[1,0,0,0]]];
```

En cuanto a las posiciones definidas para esta estación, se han creado diecisiete posiciones de robot. De las cuales once están incluidas en la tarea del Robot herramienta, relativas al MesaLatas1, y las seis restantes están incluidas en la tarea del Robot objeto, relativas a MesaLatas2.

Las once posiciones del Robot herramienta, se sitúan entre varias latas con el objetivo de hacer moverse a “MyTool”, recorriendo toda la mesa. Sin embargo, las seis posiciones

asociadas al Robot objetivo, tienen el objetivo de mover la Mesa_Latas, dentro de un espacio accesible para ambos robots.

Por otro lado, para esta estación se han declarado en ambas tareas, las variables necesarias para el movimiento sincronizado de los dos robots. Éstas variables se muestran a continuación.

```
PERS tasks Task_list{2}:=[["T_ROB1"], ["T_ROB2"]];

VAR syncident SyncOn;
```

5.2.3 Creación de trayectorias y procedimientos

Para la realización de esta estación, se han creado dos procedimientos distintos, uno para la tarea del Robot herramienta, y uno para la tarea del Robot objeto.

- **Robot herramienta:** Se ha generado una trayectoria, llamada Path_10, con dieciséis instrucciones de movimiento, en la que se recorre los once puntos definidos previamente. En dichas instrucciones, se ha añadido la ID, correspondiente para garantizar la coordinación de ambas tareas.
- **Robot objeto:** Para este robot, se ha generado una trayectoria llamada Path_10, recorriendo los seis puntos mencionados previamente. Para ello, se emplean de nuevo, dieciséis instrucciones de movimiento, porque para que dos robots se muevan de manera síncrona, los procedimientos han de tener el mismo número de instrucciones de movimiento, y que los ID de dichas instrucciones, coincidan.

El procedimiento Main de ambas tareas es idéntico, y en él se incluyen las instrucciones comentadas previamente, necesarias para la sincronización. Por otro lado, los procedimientos anteriores, se han incluido en bucle para que se ejecuten constantemente, como se muestra a continuación.

```
IF NOT(IsSyncMoveOn()) THEN

SyncMoveOn SyncOn, Task_list;

ENDIF

FOR I FROM 1 TO 50 DO

  Path_10;
```

```
ENDFOR
```

Capítulo 6

Conclusiones

El presente proyecto ha servido, para la adquisición de conocimientos y nuevas habilidades por parte del proyectando, para obtener material docente, de cara a la docencia de las asignaturas que tengas contenidos relacionados con la robótica industrial, y en especial de cara a la asignatura Automatización Flexible y Robótica.

Terminado el desarrollo del proyecto, es un buen momento para dar una visión de conjunto de lo obtenido, así como dar una valoración de los elementos con los que se han trabajado durante la realización del mismo.

- En referencia al **robot industrial modelo IRB 120**, y una vez realizada un análisis de sus características principales, se concluye que se trata de un robot adecuado para la docencia, debido a su maniobrabilidad, pequeño tamaño y funcionalidad.
- Posteriormente, haciendo uso de los manuales oficiales proporcionados por ABB, y después de realizar un análisis de lo que ofrece el **Lenguaje de programación RAPID**, se ha realizado una descripción del mismo. Se ha dotado a esa descripción de un carácter más docente, para que pueda ser empleada como guía para el alumnado, ya que los manuales oficiales son de difícil acceso y comprensión, y se ha necesitado de un buen número de horas para ser capaz de navegar por ellos con la soltura necesaria.
- Una vez realizado lo anterior, se ha empleado el **Software de simulación Robot Studio** para realizar estaciones, en las que se muestran las principales funciones y características de dicho software, para que el alumnado consiga un dominio del mismo suficiente al finalizar la asignatura. Debido a la dificultad de enfrentarse a un software de simulación nuevo por parte del alumnado, y dada la amplitud y complejidad de los manuales oficiales de ABB, se ha generado una Guía, en la que se muestra paso a paso como generar una estación y cómo programarla desde cero. Una vez finalizado, se concluye que se trata de un software potente y versátil, capaz de realizar numerosas funciones distintas dentro de estaciones. Sin embargo, se recomienda por parte del proyectando guardar los progresos multitud

de ocasiones, ya que se han detectado varias “crashes” durante el tiempo trabajando con él.

Por otro lado, a la hora de realizar las estaciones, se ha detectado que resulta más sencillo reconstruir una estación “partiendo de cero”, que realizar modificaciones o cambios dentro de una ya existente.

- Tras finalizar lo comentado previamente, se quiere destacar las **estaciones de simulación** creadas en este proyecto. En dichas estaciones se ha pretendido que el alumnado compruebe las opciones del simulador, así como que consiga dominar una herramienta tan importante a nivel industrial. Por otro lado, dichas estaciones pueden ser realizadas con el Robot IRB 120 disponible en el laboratorio, en función del tiempo disponible para ello en la planificación de las asignatuas.

Anexo A: Lenguaje RAPID

A.1 Descripción del lenguaje RAPID

RAPID (Robotics Application Programming Interactive Dialogue) es un lenguaje de programación de robots, que permite escribir las instrucciones precisas para que el robot realice la rutina, o la acción que se desee.

Se trata de un lenguaje de programación de alto nivel, es decir un lenguaje de programación con palabras y letras fácilmente comprensibles.

RAPID fue desarrollado por la empresa suiza ABB y es lenguaje que emplean sus robots, por tanto, a continuación se va a explicar su principales características.

A.1.1 Características básicas

El lenguaje de programación RAPID es un lenguaje que no distingue entre mayúsculas y minúsculas.

Los identificadores, o palabras que se emplean para dar nombre a los módulos, rutinas, datos y etiquetas, han de empezar siempre por una letra y los siguientes pueden ser letras, dígitos o caracteres de subrayado (_).

En este lenguaje, existen un grupo de palabras que tienen un significado específico dentro del mismo y que por tanto, no pueden ser empleados como identificadores. En la *TABLA 1* se muestran dichas palabras reservadas:

ALIAS	AND	BACKWARD	CASE
DIV	DEFAULT	CONST	CONNECT
ENDFOR	ELSEIF	ELSE	DO
EDPROC	ENDMODULE	ENDIF	ENDFUNC

ENDWHILE	ENDTRAP	ENDTEST	ENDRECORD
FOR	FALSE	EXIT	ERROR
IF	GOTO	FUNC	FORM
MODULE	MOD	LOCAL	INOUT
OR	NOVIEW	NOT	NOSTEPIN
READONLY	RAISE	PROC	PERS
STEP	RETURN	RETRY	RECORD
TO	THEN	TEST	SYSMODULE
UNDO	TRYNEXT	TRUE	TRAP
WITH	WHILE	VIEWONLY	VAR
XOR			

Tabla A.1 Palabras reservadas en RAPID

Es posible emplear espacios en cualquier lugar del código, excepto dentro un identificador, una palabra reservada, un valor numérico o una marca de sustitución.

Se pueden emplear valores numéricos, los cuales pueden expresarse como un entero o como un número decimal (con “,”), valores lógicos, True o False, o valores de cadena de caracteres.

En RAPID se pueden escribir comentarios, que empezarán con un signo de exclamación (!) y terminan con un carácter de salto de línea.

A.1.2 Tipos de datos o variables

En RAPID, los datos o variables que creamos pueden tener un alcance u otro, es decir pueden ser accesibles únicamente desde el módulo en el que han sido declaradas o ser accesibles desde todos los módulos del programa:

Local: La variable o dato únicamente es accesible desde el modulo en el que ha sido declarada.

Global: La variable o dato es accesible desde todos los módulos del programa.

Persistente: La variable o dato es accesible desde todos los módulos del programa pero el valor de la misma no es reinicializable al comenzar de nuevo los programas.

A continuación se van a mostrar los principales tipos de datos que se emplean en este lenguaje, así como su principal utilización.

- **Num:** Se trata de un dato número, y se puede emplear en cualquier expresión, también existe el tipo **bool** para expresiones lógicas y el tipo **string** para expresiones de cadena de caracteres.

La declaración de estos tipos de datos se haría de la siguiente manera:

VAR num reg1;
VAR bool highvalue;
VAR string text;
Reg1 := 100;
Highvalue := reg1>100;
Text := "reg1 es mayor que 100";

- **Pos:** se trata de un registro, el cual permite expresar una posición mediante sus coordenadas cartesianas [X, Y, Z]. Cada una de esas coordenadas cartesianas se expresa como un tipo de dato *num*. La fa forma correcta de definir este tipo de dato es la siguiente:

VAR pos pos1

pos1 := [100,200,250];

A los registros también se puede acceder de la siguiente forma: *pos1.x:= 100;* con esto se ha asignado un valor a la componente "X" de *pos1*.

- **Orient:** es un registro que permite especificar una orientación o una rotación, ambas expresadas como cuaternios**.** Dichos cuaternios tienen cuatro componentes, que se tratan de 4 tipos de datos *num*. El tipo de dato *orient* se define de forma muy similar al tipo de dato *pos:*

VAR orient orient1

orient1 := [1,0,0,0]; !! Lo que quiere decir que no se ha producido ninguna rotación.

- **Pose:** Se trata de un registro, y se emplea para cambiar de un sistema de coordenadas a otro. Es decir, describe como se desplaza y gira un sistema de coordenadas alrededor de otro. Por ejemplo, pueden definir como está situado el sistema de coordenadas de objeto con respecto al sistema de coordenadas de usuario. Tiene dos componentes, la parte de *trans,* que se trata de un tipo de dato pos, y la parte *rot,* que se trata de un tipo de dato orient. Por tanto, el tipo de dato *pose* se definiría de la siguiente manera:

VAR pose pose1

pose1 := [[100,200,250],[1,0,0,0]];

- **ConfData:** define las configuraciones del robot. Se utiliza para resolver las ambigüedades que se producen cuando una determinada posición y orientación es alcanzable con más de una combinación de valores para las articulaciones. Se especifican los cuadrantes en los que se encuentran los ejes del robot.

Particularizando para el caso del robot con el que se han realizado los ejemplos que se han mostrado previamente, el robot modelo IRB 120, el tipo de dato tiene el siguiente aspecto:

VAR confdata conf_reposo := [1,-1,0,1]:

Como se puede observar, el tipo de dato *confdata*, tiene cuatro componentes:

Cf1: se trata de un tipo de dato *num*, e indica, al ser un eje de rotación, el cuadrante en el que se encuentra el Eje1, y puede ser un numero positivo o negativo.

Cf4: se trata de un tipo de dato *num*, e indica, al ser un eje de rotación, el cuadrante en el que se encuentra el Eje4, y puede ser un numero positivo o negativo.

Cf6: se trata de un tipo de dato *num*, e indica, al ser un eje de rotación, el cuadrante en el que se encuentra el Eje6, y puede ser un numero positivo o negativo.

Cfx: Se utiliza para seleccionar una de las ocho configuraciones posibles del robot, numeradas de 0 a 7.

A continuación, se muestra la Figura 1.1 para que sirva de aclaración en cuanto a la utilización de valores positivos y negativos, así como al cuadrante en el que se encuentra el eje en cuestión.

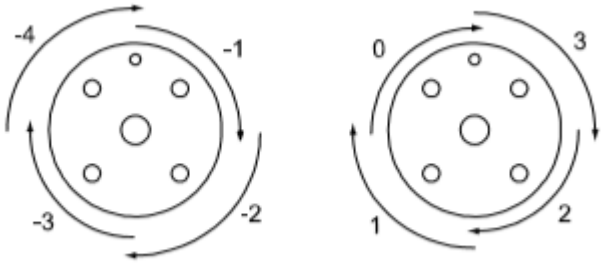

Figura A.1 Nomenclatura de los cuadrantes

Por tanto, para cada eje el cuadrante 0 es el primer cuadro de revolución de 0 a 90^{o} en sentido positivo a partir de la posición cero. El cuadrante 1 es el siguiente cuarto de revolución, de 90 a 180^{o}, y así en adelante. El cuadrante -1, es el cuarto de revolución de 0 a -90^{0}, etc.

- **LoadData:** define las características de la carga que deberá manipular el robot (del objeto a agarrar con la pinza). Se utiliza para ajustar los pares a ejercer por los motores del robot. Este tipo de dato, tiene varios componentes dentro del mismo:

Mass: Indica el peso en Kg de la carga, y se emplea el tipo de dato *num*.

Cog ("Center of gravity"): Indica el centro de gravedad de la carga (en mm), en el sistema de coordenadas de la herramienta si el robot es el que sujeta la herramienta. Si por el contrario se utiliza una estacionaria, e centro de gravedad de la carga sostenida por la herramienta se expresara en la base de coordenadas de objeto, del sistema de coordenadas del objeto de trabajo movido por el robot. Para ello se emplea un tipo de dato *pos*.

Aom ("axes of movement"): Indica la orientación de los ejes de momento, es decir, la orientación de los ejes principales del momento de inercia de la carga útil, con centro en *cog*. Si el robot es el que sujeta la herramienta los ejes de momento, se expresan en el sistema de coordenadas de la herramienta, si por el contrario, se utiliza una herramienta estacionaria se utiliza el sistema de coordenadas del objeto. El tipo de dato utilizado es el tipo *orient*.

Ix: Indica el momento de inercia de la carga alrededor del eje X del momento., expresado en kgm^2. Por tanto, el tipo de dato empleado es el tipo *num.*

Iy: Indica el momento de inercia de la carga alrededor del eje Y del momento., expresado en kgm^2. Por tanto, el tipo de dato empleado es el tipo *num.*

Iz: Indica el momento de inercia de la carga alrededor del eje z del momento., expresado en kgm^2. Por tanto, el tipo de dato empleado es el tipo *num*.

Si dichos momentos de inercia son todos iguales a 0, se trata de una carga puntual.

De este modo, el tipo de dato *Loaddata* se definiría de la siguiente manera:
PERS loaddata loaddata1 := [5,[100,0,100],[1,0,0,0],0,0,0];

- **Robjoint:** expresa la posición en grados de cada uno de los ejes o articulaciones del robot. Consta de 6 componentes, que indican la posición de cada uno de los ejes del robot, en grados, respecto a la posición de calibración.

VAR robjoint robjoint1 := [0,0,0,0,0,0];

- **RobTarget:** define la posición y orientación deseada para el extremo del robot, incluyendo los datos de configuración necesarios para que no existan ambigüedades. También especifica la posición deseada para los ejes externos, si éstos existieran. Este tipo de dato, está compuesto de 4 componentes distintas:

Trans: Es un tipo de dato *pos*, e indica la posición del punto central de la herramienta (X,Y,Z). La posición se especifica respecto al sistema de coordenadas del objeto de trabajo actual. Si este no se ha definido todavía, se especifica respecto al sistema de coordenadas mundo.

Rot: se trata de un tipo de dato *orient*, e cual indica la orientación dela herramienta, expresada en cuaternios (q1,q2,q3,q4) respecto al sistema de coordenadas del objeto de trabajo actual. De nuevo, si este no se ha especificado, se expresara respecto al sistema de coordenadas mundo.

Robconf: Es del tipo *confdata*, y expresa la configuración de los ejes del robot.

Extax ("externa axes"): Se trata del tipo de dato *extjoint,* el cual indica la posición de los ejes adicionales, si es un eje de rotación se indicara la rotación en grados con respecto a la posición de calibración, y si se trata de un eje lineal, se indicara la distancia con respecto a la posición de calibración. Si dichos ejes adicionales, o externos, no se están empleando, se les da el valor de "9E9".

Por tanto, se definiría un tipo de dato *robtarget* como:

CONST robtarget robtarget1 :=[
[0,100,25],[1,0,0,0],[1,1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]]

- **JointTarget:** especifica una posición deseada para los ejes (articulaciones) del robot. Se utiliza cuando se quiere gobernar el robot directamente a través de sus articulaciones y no especificando posiciones y orientaciones. Este tipo de dato esta compuesto por, un tipo *confdata* y otro *extjoint,* por tanto se definirá de manera similar a *robtarjet*.

- **MotSetData:** define las características básicas de todos los movimientos a efectuar por el robot, siempre que no se especifique lo contrario. Estas características son: velocidad, aceleración, precisión, comportamiento en puntos singulares, etc.

- **SpeedData:** define la velocidad a la que debe efectuarse un movimiento. Esta velocidad se define por separado para la traslación, la rotación y el movimiento de los ejes externos. Existen unos valores predefinidos por el sistema, en los que la velocidad de reorientación, y la velocidad a la que se mueven los ejes lineales y de rotación permanecen constantes, 500°/s, 5000mm/s y 1000°/s respectivamente. Por tanto, en dichos datos predefinidos por el sistema solo variara la velocidad del TCP, por ejemplo: v5, v10, v20, v30… En dichos valores, la velocidad del TCP será de 5, 10, 20, 30… mm/s en cada caso.

- **StopPointData:** especifica la precisión a alcanzar en el punto final de una trayectoria. Se establece también un tiempo límite para alcanzar el destino con la precisión pedida.

- **ToolData:** describe las características de una determinada herramienta: pinza de agarre, boquilla de soldadura, etc. Las dimensiones de la herramienta se utilizan para calcular las trayectorias respecto al punto central de la misma en lugar de calcularlas respecto al extremo del robot. Los datos relativos al peso y centro de gravedad de la herramienta se utilizan para adecuar los pares a ejercer por los motores del robot. Este tipo de datos, tiene varios componentes:

  Robhold: es un tipo de dato *bool*, e indica si es el robot el que sostiene la herramienta o si por el contrario se trata de una herramienta estacionaria.

  Tframe: se trata de un dato tipo *pose*, y por tanto define el sistema de coordenadas de la herramienta. Tanto la posicion del TCP como la orientación de dicho sistema.

  Tload: define las características de la carga con la que va a trabajar el robot, por tanto se trata de un tipo de dato *loaddata.*

  Por ello, el tipo de dato *tooldata* se definiría de la siguiente forma:

PERS tooldata tooldata1 := [TRUE, *[[100,0,150],[1,0,0,0]],[5,[100,0,100],[1,0,0,0],0,0,0]];*

- **ZoneData:** especifica la precisión a alcanzar en un punto que no necesariamente debe ser el final de una trayectoria (es válido en cualquier caso). Existen unas zonas predefinidas por el sistema que van en progresivo aumento de la distancia respecto a la posicion programa, a la que se van a encontrar los ejes antes de iniciar el movimiento hacia el destino siguiente. Estos valores serian: fine (para puntos de paro), z0, z1, z5, z10, z15… etc.

- **Wobjdata:** Se utiliza para describir el objeto de trabajo que el robot está soldando, procesando, moviendo, etc. Si los objetos de trabajo están definidos en una instrucción de posicionamiento, la posición se basara en las coordenadas del objeto de trabajo. Por el contrario, si se utiliza una herramienta estacionaria o ejes externos coordinados, es necesario definir el objeto de trabajo, ya que en este caso la trayectoria y la velocidad estarian relacionadas con el objeto de trabajo y no con el TCP.

El tipo de dato *Wobjdata* tiene varios componentes:

Robhold: Es un tipo de dato *bool*, y define si el robot de la tarea de programa actual es el que está sosteniendo el objeto de trabajo. TRUE, el robot sostiene el objeto de trabajo, es decir, la herramienta es estacionaria, FALSE, el robot no está sosteniendo el objeto de trabajo y por el contrario está sosteniendo la herramienta.

Ufprog: Es un tipo de dato *bool*, y define si está utilizando un sistema fijo de coordenadas de usuario. TRUE, Sistema de coordenadas fijo, FALSE, Sistema móvil de coordenadas de usuario es decir, se utilizan ejes externos.

Ufmec: Es un tipo de dato *string*, que define la unidad mecánica con la que se coordinan los movimientos del robot. Solo es necesario en el caso de los sistemas móviles de coordinadas de usuario (*Ufprog* es FALSE).

Uframe: Se trata de un tipo de dato *pose*, y se encarga de definir el Sistema de coordenadas de usuario, es decir la posición de la superficie o del util de trabajo actual. La posición del origen del de Sistema de coordenadas (x,y,z) en m, y la rotación del sistema de coordenadas, expresada como un cuaternio (q1,q2,q3,q4). Si el robot es el que sostiene la herramienta, el Sistema de coordenadas del usuario se define en el

Sistema de coordenadas Mundo. En el caso de una base de coordenadas de usuario móvil (*Ufprog* es FALSE), la base de coordenadas del usuario está definida constantemente por el Sistema.

*Oframe***:** de nuevo se trata de un tipo de dato *pose*, y en este caso define el Sistema de coordenadas del objeto, es decir la posición del objeto de trabajo actual. La posición del origen del de Sistema de coordenadas (x,y,z) en m, y la rotación del sistema de coordenadas, expresada como un cuaternio (q1,q2,q3,q4). El Sistema de coordenadas del objeto se define dentro del Sistema de coordenadas del usuario.

A modo de ejemplo, para una mayor aclaración:

PERS wobjdata cilindro2 := [FALSE,TRUE,"", [[200,550,300],[1,0,0,0]],[[150,500,400], [1,0,0,0]]]:

A.1.3 Funciones, operadores e instrucciones más empleados en RAPID

A.1.3.1 Operadores numéricos

Los operadores numéricos se emplean para evaluar valores numéricos y realizar operaciones con ellos:

- **Suma (+):** Se puede realizar tanto suma de escalaras (*num + num*) o suma de vectores (*pos + pos).*
- **Resta (-):** Se puede realizar tanto resta de escalares (*num – num*) o resta de vectores (*pos – pos).*
- **Multiplicación (*):** Se puede realizar multiplicación de escalares (*num * num*), de un vector por un escalar (*num*pos ó pos*num*), multiplicación de vectores (*pos*pos)* o vinculación de rotaciones (*orient*orient).*
- **División (/), División entera (DIV), Módulo del entero ó resto (MOD):** Solo se realiza con escalares.

A.1.3.2 Operadores relacionales

Son operadores que devuelven el tipo de dato *bool*, y establecen relaciones entre otros tipos de datos.

- **Menor que (<):** Por ejemplo, *Valor:= dato1<dato2* !Valor será TRUE si *dato1* es menor que *dato2*.
- **Menor o igual que (<=):** Por ejemplo, *Valor:= dato1<=dato2* !Valor será TRUE si *dato1* es menor o igual que *dato2*.
- **Mayor que (>):** Por ejemplo, *Valor:= dato1>dato2* !Valor será TRUE si *dato1* es mayor que *dato2*.
- **Mayor o igual que (>=):** Por ejemplo, *Valor:= dato1>=dato2* !Valor será TRUE si *dato1* es mayor o igual que *dato2*.
- **Igual a (=):** Por ejemplo, *Valor:= dato1=dato2* !Valor será TRUE si *dato1* es igual que *dato2*.
- **Distinto de (<>):** Por ejemplo, *Valor:= dato1<>dato2* !Valor será TRUE si *dato1* es distinto que *dato2*.

A.1.3.3 Operadores lógicos

Se tratan de operadores que, de nuevo, devuelven el tipo de dato *bool*, pero esta vez se emplean para evaluar valores lógicos (TRUE/FALSE).

Algunos de ellos son: *AND, OR, XOR y NOT*, y su uso más común es emplearlos para establecer condiciones en instrucciones de control del flujo del programa dentro de una rutina (Se verán más adelante en este trabajo).

A.1.3.4 Instrucciones de control de flujo del programa

El flujo del programa puede controlarse de acuerdo a cinco reglas o principios distintos:

1. Se llama a otro procedimiento y una vez ejecutado dicho procedimiento, se continúa con la siguiente instrucción, la que sigue al llamamiento del procedimiento, en el programa principal. Se emplean instrucciones como *ProccCall, CallByVar, Return.*
2. Se ejecutan instrucciones diferentes en función de si se cumple una condición determinada. Para ello se emplean instrucciones típicas cómo *IF…THEN, TEST/CASE.*
3. Se repite una secuencia un número determinado de veces o hasta que se cumple una condición determinada.
4. Ir a una etiqueta dentro de la misma rutina.
5. Detener la ejecución del programa

Las instrucciones que realizan el control del flujo del programa son muy similares a las que se emplean en los lenguajes de programación convencionales (C, Pascal...).

- **Instrucciones de llamada a otra rutina/procedimiento**:

Instrucción	Se usa para:
ProcCall	Llamar (Saltar) a otra rutina
CallByVar	Llamar a procedimientos que tienen nombres concretos
RETURN	Volver a la rutina original

Tabla A.2 Instrucciones de llamada a otra rutina en RAPID

- **Instrucciones de control del programa dentro de la rutina:**

Instrucción	Se usa para
IF	Ejecutar una instrucción sólo si se cumple una condición
IF...THEN	Ejecutar una secuencia de instrucciones en función de si se cumple una condición
FOR	Repetir una sección del programa un número determinado de veces
WHILE	Repetir una secuencia de instrucciones mientras siga cumpliéndose una condición
TEST/CASE	Ejecutar instrucciones diferentes en función del valor de una expresión
GOTO	Saltar a una etiqueta
label	Especificar una etiqueta (un nombre de línea)

Tabla A.3 Instrucciones de control del programa dentro de la rutina

- **Instrucciones de detención de la ejecución del programa:**

Instrucción	Se usa para:
Stop	Detiene la ejecución del programa
EXIT	Detiene la ejecución del programa de forma que no se permita reiniciarlo
Break	Detener temporalmente la ejecución del programa con fines de depuración
SystemStopAction	Detener la ejecución

Tabla A.4 Instrucciones de detención de la ejecución del programa

A.1.3.5 Instrucciones de movimiento del robot

A la hora de programar un movimiento del robot, se programa de posición a posición, es decir, moverse de la posición en la que se encuentra a una posición nueva. Luego el robot es el que realmente calcula automáticamente la trayectoria entre las dos posiciones.

Las características básicas del movimiento, como el tipo de trayectoria (lineal, articular, circular), se especifican mediante la adecuada selección de la instrucción de movimiento.

El resto de características se especifican mediante la definición de datos como argumentos de dicha instrucción de movimiento. Estos datos son:

- Datos de posición
- Datos de velocidad
- Datos de la zona
- Datos de la herramienta

A continuación de muestran las instrucciones más empleadas de movimiento de robot.

Instrucción	Se usa para:
MoveL	Movimiento del robot siguiendo una trayectoria lineal

MoveJ	Movimiento del robot de forma articular
MoveC	Movimiento del robot siguiendo una trayectoria circular

Tabla A.5 Instrucciones de movimiento

Para una mayor aclaración se va a mostrar un ejemplo sencillo de cómo serían dichas instrucciones de movimiento:

$MoveL \quad posdestino,\ v200,\ z20,\ tool1$

$MoveJ \quad posdestino,\ v200,\ z20,\ tool1$

$MoveC \quad posdestino,\ posint,\ v200,\ z20,\ tool1$ *! Es necesario indicar un punto de paso intermedio para indicar la curvatura que tendrá la trayectoria circular*

En los tres casos, la orientación del extremo del robot cambiará desde la orientación inicial hasta la orientación especificada en la posición de destino, si se diese el caso en el que las dos orientaciones fuesen muy similares, la orientación del extremo del robot no cambiaría.

Los tipos de datos empleados como argumentos de entrada en estas instrucciones de movimiento los que siguen:

Posdestino → *RobotTarget*

Posint → *RobotTarget*

V200 → *SpeedData*

Z20 → *ZoneData*

Tool1 → *ToolData*

A.1.3.6 Instrucciones de movimiento relativo del robot

En RAPID, los puntos de destino del robot pueden definirse relativos a otro punto previamente definido, es decir con un ligero desplazamiento en el eje X, por ejemplo, o una rotación de 40 grados respecto el eje Z. Para ello, se emplean estas dos instrucciones:

- ***Offs:*** Se utiliza para añadir un offset en el Sistema de coordenadas de objeto a una posición del robot. La función tendría esta forma: Offs (point XOffset YOffset ZOffset), es decir, tiene varios argumentos.

Point: Es un tipo de dato *RobotTarget,* indica el punto sobre el que se va a realizar el desplazamiento.

XOffset: Se trata de un tipo de dato *num*, e indica el desplazamiento en la dirección X del sistema de coordenadas del objeto.

YOffset: Se trata de un tipo de dato *num*, e indica el desplazamiento en la dirección Y del sistema de coordenadas del objeto.

ZOffset: Se trata de un tipo de dato *num*, e indica el desplazamiento en la dirección Z del sistema de coordenadas del objeto.

- ***RelTool:*** Se utiliza para añadir un desplazamiento y/o una rotación, expresada en el sistema de coordenadas de la herramienta activa, a una posición del robot. Dicha función tendría el siguiente aspecto: RelTool (Point Dx Dy Dz [\Rx] [\Ry] [\Rz]), por tanto tiene varios argumentos:

Point: Es un tipo de dato *RobotTarget* e indica la posición de entrada del robot. La parte de orientación de esta posición define la orientación actual del sistema de coordenadas de la herramienta.

Dx: Se trata de un tipo de dato *num*, e indica el desplazamiento en la dirección X del sistema de coordenadas de la herramienta.

Dy: Se trata de un tipo de dato *num*, e indica el desplazamiento en la dirección Y del sistema de coordenadas de la herramienta.

Dz: Se trata de un tipo de dato *num*, e indica el desplazamiento en la dirección Z del sistema de coordenadas de la herramienta.

\Rx: Se trata de un tipo de dato *num*, e indica la rotación en grados alrededor del eje X del sistema de coordenadas de la herramienta.

\Ry: Se trata de un tipo de dato *num*, e indica la rotación en grados alrededor del eje Y del sistema de coordenadas de la herramienta.

\Rz: Se trata de un tipo de dato *num*, e indica la rotación en grados alrededor del eje Z del sistema de coordenadas de la herramienta.

Si se especifican dos o tres rotaciones al mismo tiempo, estas se realizarán en el siguiente orden: Primero se realizara la rotación alrededor del eje X, posteriormente alrededor del nuevo eje Y, y para terminar alrededor del nuevo eje Z.

A.1.4 Señales de entrada y salida

El robot puede contar con varias señales de usurario digitales y analógicas que pueden leerse y modificarse desde el propio programa.

Una señal digital puede tener dos valores, 0 o 1, mientras que el valor de una señal analógica o de un grupo de señales digitales se especifica como un valor numérico.

Los nombres de estas señales se definen siempre en los parámetros del sistema. Una vez allí, siempre se encuentran disponibles para su lectura o el establecimiento de operaciones de entrada/salida.

Las operaciones principales que se pueden realizar con las señales son, comprobar el valor de una determinada entrada, por ejemplo, comprobar mediante un sensor una pieza está colocada en la posición correcta para ser manipulada, o buen fijar el valor de una determinada salida, por ejemplo abrir o cerrar una herramienta tipo pinza cuando se quiere agarrar o soltar un objeto.

El valor de una señal de entrada puede leerse directamente desde el programa, como se indica a continuación:

```
IF di1=1 THEN…! Entrada digital

IF gi1=5 THEN…! Grupo de entradas digitales

IF ai1=5.3 THEN…! Entrada analógica
```

O bien, emplear una instrucción *wait*, que permite comprobar el valor de una señal de entrada y hace esperar el robot hasta la señal llega a un valor determinado.

```
WaitDI pieza, HIGH
```

WaitAI signal, 3.3

Para fijar el valor de una salida se emplean fundamentalmente las siguientes instrucciones:

- **Set:** Fija el valor de una salida digital a 1
- **Reset:** Fija el valor de una salida digital a 0
- **SetAO:** Cambia el valor de una salida analógica
- **SetDO:** Cambia el valor de una salida digital, a uno de sus valores simbólicos

A.2 Programación de movimiento

A continuación se va a hacer una pequeña aclaración en cuanto a la forma de programar el movimiento del robot en el lenguaje RAPID. Incluyendo como se relacionan los sistemas de coordenadas que definen dicho movimiento, así como el posicionamiento de la herramienta durante la ejecución del programa, la sincronización con las señales digitales, la configuración del robot y como controlarla y como controlar el movimiento mientras se pasa por una singularidad.

A.2.1 Sistema de coordenadas

A.2.1.1 Punto central de la herramienta (TCP)

La posición de un Robot y sus movimientos están relacionados en todo momento con el punto central de la herramienta (TCP). Este punto suele definirse como una parte determinada de una herramienta, por ejemplo, la boquilla de una pistola de adhesivo, el centro de una pinza o el extremo de una herramienta rectificadora.

Se puede definir más de un TCP (más de una herramienta), pero sólo uno estará activo en cada momento. El TCP es el punto que se desplaza a lo largo de una trayectoria determinada y a una velocidad especificada.

Cuando el robot está sosteniendo un objeto de trabajo, y trabajando con una herramienta estacionaria, se utiliza un TCP estacionario. En cambio, si dicha herramienta está activa, la trayectoria programa y la velocidad serán también las del objeto de trabajo.

A.2.1.2 Sistemas de coordenadas empleados para determinar la posición del TCP.

La posición de la herramienta (TCP) puede especificarse con distintos sistemas de coordenadas para facilitar la programación y el ajuste de los programas.

El sistema de coordenadas definido depende de lo que se espera que haga dicho robot. Si hay un sistema de coordenadas definido, las posiciones del robot se definen mediante el sistema de coordenadas de la base.

- Sistema de coordenadas de la base

En el sistema de coordenadas base el Eje Z, coincide con el eje 1 del robot. (Figura A.2)

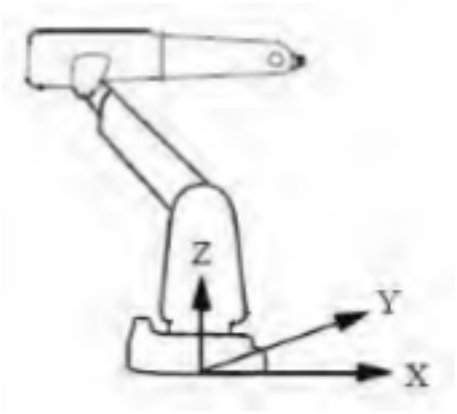

Figura A.2 Sistemas de coordenadas de la base

- El *Origen* está situado en la intersección del Eje 1 con la superficie de montaje de la base.
- El plano XY coincide con la superficie de montaje de la base
- El eje X apunta hacia delante
- El eje Y apunta hacia la izquierda (Desde la perspectiva del robot)
- El eje Z coincide con el Eje 1

- Sistema de coordenadas Mundo

Si el robot está montado en el suelo, la programación con el sistema de coordenadas base resulta sencilla. Sin embargo, cuando el robot está suspendido (montado de forma invertida), las direcciones de los ejes no coinciden con las direcciones principales del espacio de trabajo. Para estos casos, es conveniente definir un sistema de coordenadas mundo. Si éste no se especificase coincidiría con el sistema de coordenadas base.

Cuando dos robots trabajando dentro del mismo espacio de trabajo dentro de un centro de producción, se emplea un mismo sistema de coordenadas mundo para que los programas de los robots puedan comunicarse unos con otros.

En la Figura A.3 se muestra un ejemplo de dos robots, estando uno de ellos suspendido, con el mismo sistema de coordenadas mundo.

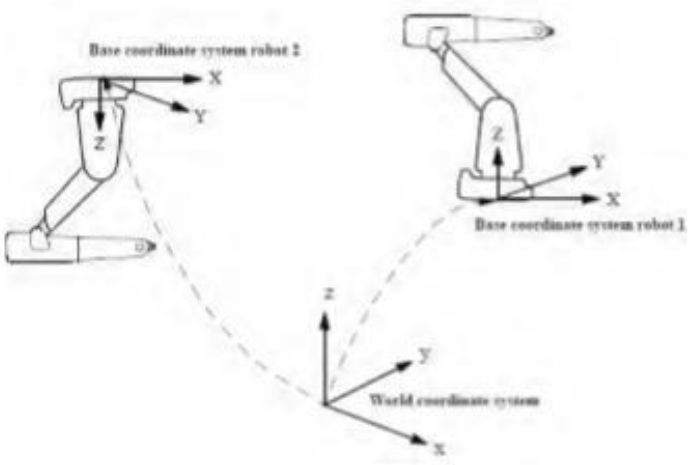

Figura A.3 Sistema de coordenadas de dos robots, con el mismo Sistema de coordenadas Mundo

- Sistemas de coordenadas del usuario

Un mismo robot puede trabajar con varias superficies de trabajo, que pueden tener orientaciones y posiciones diferentes. Se puede definir un sistema de coordenadas del usuario para cada superficie de trabajo.

Dicho sistema de coordenadas del usuario se define a partir del sistema de coordenadas mundo.

En la Figura A.4, se muestra como dos sistemas de coordenadas del usuario describen la posición de dos útiles diferentes.

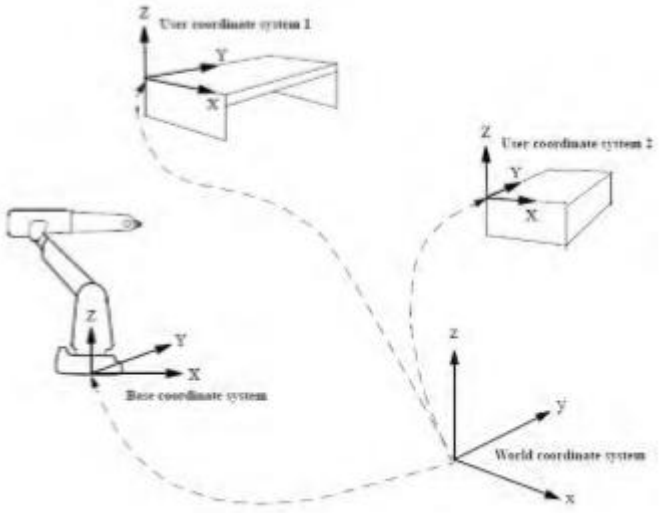

Figura A.3 Dos sistemas de coordenadas de Usuario

- Sistema de coordenadas de objeto

Como se ha mencionado el sistema de coordenadas del usuario se utiliza para definir distintas superficies de trabajo, sin embargo dentro de dichas superficies puede haber distintos objetos que van a ser manipulados por el robot. Por tanto, resulta útil definirse un sistema de coordenadas para cada objeto de trabajo (sistema de coordenadas de objeto) con el fin de facilitar el ajuste del programa si se mueve dicho objeto.

Por tanto, el sistema de coordenadas del objeto, se define a partir del sistema de coordenadas del usuario. Por tanto, como se muestra en el Figura A.5, puede haber dos sistemas de coordenadas de objeto describiendo la posición de dos objetos de trabajo diferentes en un mismo útil.

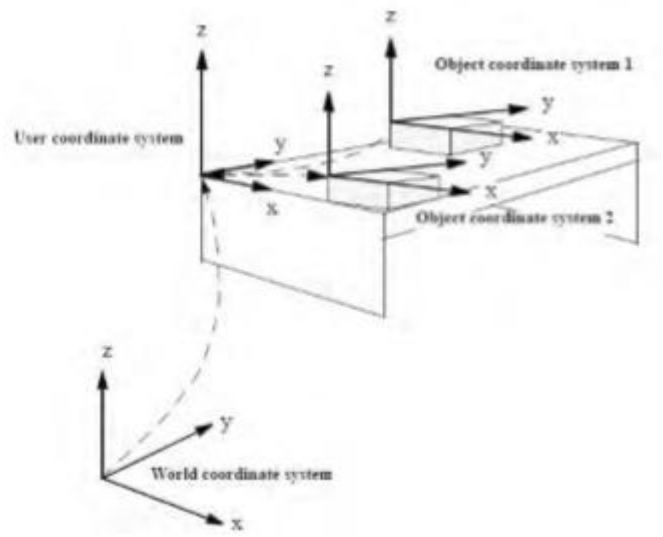

Figura A.5 Sistemas de coordenadas de Objeto

Las posiciones programadas se definen relativas a un sistema de coordenadas objeto. Si se mueve o se gira una superficie de trabajo, este cambio puede compensarse mediante el giro o movimiento del sistema de coordenadas del usuario, no es necesario modificar ni las posiciones programadas ni los sistemas de coordenadas del objeto que se hayan definido.

Por otro lado, si se mueve el objeto de trabajo, puede compensarse dicho cambio con un movimiento o giro del sistema de coordenadas del objeto.

- Sistema de coordenadas de desplazamiento

En ciertas ocasiones, es necesario realizar una misma trayectoria en varios lugares de un mismo objeto. Para evitar la reprogramación de todas las posiciones cada vez, es posible

definir un sistema de coordenadas (sistema de coordenadas de desplazamiento), para compensar las diferencias existentes entre las distintas piezas.

El sistema de coordenadas de desplazamiento se define a partir del sistema de coordenadas de objeto.

- Ejes adicionales coordinados

Coordinación del sistema de coordenadas del usuario

Cuando un objeto de trabajo se sitúa sobre una unidad mecánica externa que se mueve mientras el robot está siguiendo la trayectoria definida por el sistema de coordenadas de objeto, es posible definir un sistema de coordinadas de usuario. En tal caso, la posición y la orientación de dicho sistema dependerán de la rotación de los ejes de la unidad mecánica externa.

Así pues, la trayectoria y la velocidad programadas dependerán del objeto de trabajo y no será necesario tener en cuenta el hecho de que dicho objeto sea movido por una unidad mecánica externa. Esto se ilustra en la Figura A.6, en la que se muestra un sistema de coordenadas de usuario definido de forma que siga los movimientos de una unidad mecánica externa de 3 ejes.

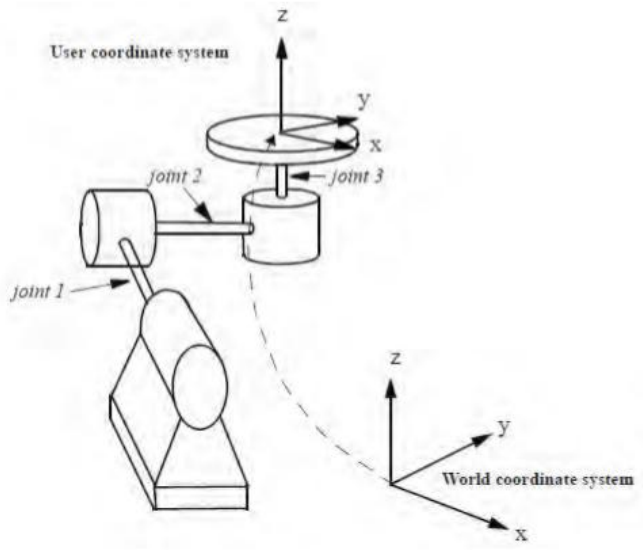

Figura A.6 Sistema de coordenadas de usuario Movil

Coordinación del sistema de coordenadas de la base

Además se puede definir un sistema de coordenadas móvil para la base del robot, lo cual resulta útil en el caso de que el robot esté montado sobre un track o un pórtico. Del mismo modo que en el caso anterior, la posición y orientación del sistema de coordenadas de la base dependerán de los movimientos de una unidad externa.

Por ello, la trayectoria y la velocidad programadas dependerán del sistema de coordenadas de objeto y no se tendrá en cuenta el movimiento de la base debido a la unidad externa. En la Figura A.7, se muestra definido el sistema de coordenadas de la base de forma que siga el movimiento del track.

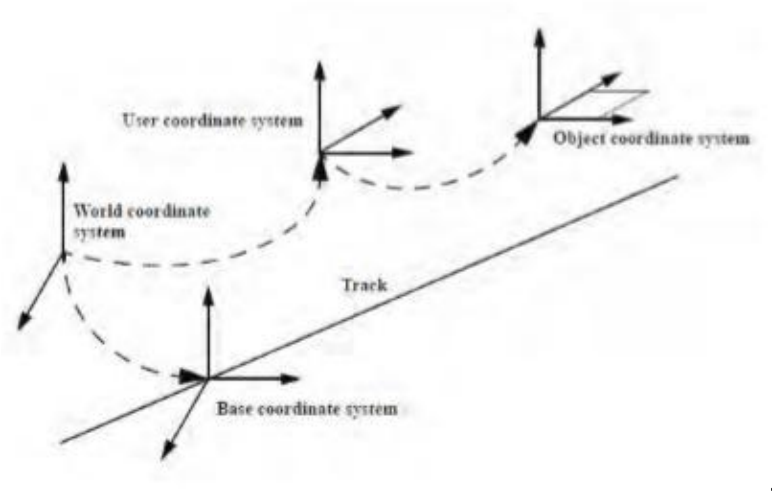

Figura A.7 Sistema de coordenadas de la Base siguiendo el movimiento del Track

Para poder calcular dichos sistemas de coordenadas cuando las unidades utilizadas están en movimiento, el robot ha de tener en cuenta:

- Las posiciones de calibración de dichos sistemas
- Las relaciones existentes entre los ángulos de los ejes adicionales y la traslación/rotación de los sistemas de coordenadas del usuario y de la base.
- Estas relaciones se definen a partir de los parámetros del sistema.

A.2.1.3 Sistemas de coordenadas utilizados para determinar la dirección de la herramienta

La orientación de una herramienta en una posición programada depende de la orientación del sistema de coordenadas de la herramienta, y dicho sistema depende del sistema de coordenadas de la muñeca definido en la brida de montaje de la muñeca del robot.

- Sistema de coordenadas de la muñeca

Cuando la aplicación es sencilla, el sistema de coordenadas de la muñeca puede usarse para definir el sistema de coordenadas de la herramienta. Siendo coincidentes en el caso de la Figura A.8 el eje Z, con el eje 6 del robot.

Figura A.8 Sistema de Coordenadas de la muñeca

El sistema de coordenadas de la muñeca no puede cambiarse y es siempre el mismo que el de la brida de montaje de la muñeca en los siguientes aspectos:

- El *origen* está situado en el centro de la brida de montaje
- El *eje X* apunta en el sentido opuesta, hacia el orificio de control de la brida de montaje
- El *eje Z* apunta hacia fuera, en un ángulo recto respecto de la brida de montaje

- Sistema de coordenadas de la herramienta

Sin embargo, en la mayoría de ocasiones la herramienta montada en la brida de montaje requiere de su propio sistema de coordenadas para la definición de su TCP (Tool Center Point). Dicho sistema de coordenadas también puede usarse para obtener los sentidos de movimiento adecuados durante el desplazamiento del robot.

El TCP, es el origen de coordenadas de dicho sistema, y se selecciona como el punto de la herramienta que debe estar correctamente posicionado.

En las siguientes figuras A.9, A.10 y A.11, se muestran los distintos sistemas de coordenadas que se definen para una pistola de soladura con arco, una pistola de soldadura por puntos y para una pinza respectivamente.

Figura A.9 TCP de la herramienta Pistola de soldadura con Arco

Figura A.10 TCP de la herramienta Pistola de soldadura por puntos

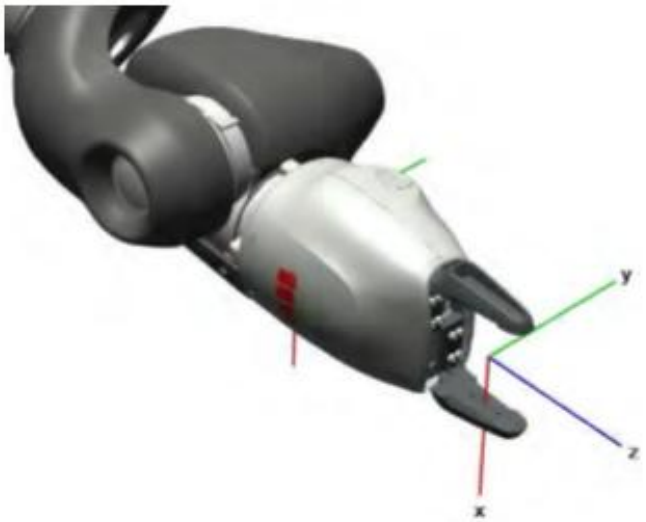

Figura A.11 TCP de la herramienta Pinza

- TCP estacionarios

Si el robot está sosteniendo un objeto de trabajo y la herramienta con la que está trabajando es estacionaria, se utiliza un TCP estacionario. Si dicha herramienta está activa, la trayectoria programada y la velocidad son las del objeto de trabajo. Por tanto, los sistemas de coordenadas se invierten.

En la Figura A.12 se muestra como quedarían algunos de los sistemas de coordenadas mencionados previamente, si el TCP fuese estacionario.

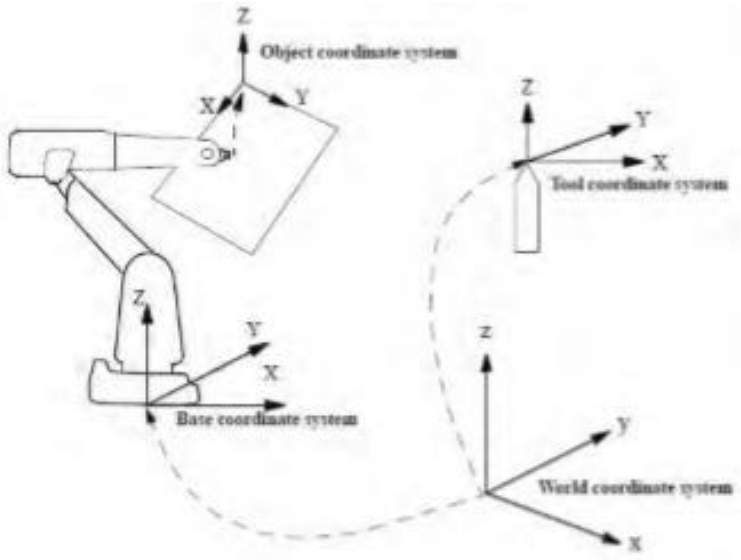

Figura A.12 Sistemas de Coordenadas con TCP estacionario

A.2.2 Posicionamiento durante la ejecución del programa

A.2.2.1 Introducción

Durante la ejecución del programa las instrucciones de posicionamiento controlan todo el movimiento del robot. Y la tarea principal de dichas instrucciones es aportar información acerca de:

- El punto de destino del movimiento (Definido por la posición del TCP, la orientación de la herramienta, la configuración del robot y la orientación de los ejes adicionales)
- El método de interpolación utilizado para llegar al punto de destino
- La velocidad de los ejes (adicionales y del robot)
- Los datos de la zona (definen como deben atravesar el punto de destino tanto los ejes del robot, como los adicionales)
- Los sistemas de coordenadas utilizados para el movimiento

A.2.2.2 Interpolación de la posición y la orientación de la herramienta

- Interpolación de los ejes

Si la exactitud de la trayectoria no es importante, se utiliza éste tipo de interpolación, ya que es la forma más rápida de mover la herramienta de una posición a otra.

Todos los ejes se mueven desde el punto de partida hasta el punto de destino con una velocidad constante de los ejes. Dicho movimiento se ilustra en la Figura A.12.

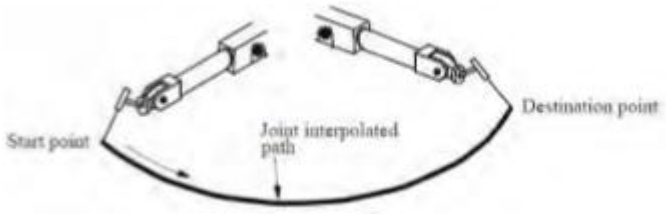

Figura A.13 Interpolación de los ejes

Durante la interpolación se determina la velocidad (con respecto a la máxima) del eje limitante y a partir de ahí se calculan las del resto, de forma que todos alcancen el punto de destino al mismo tiempo.

- Interpolación Lineal

Durante la interpolación lineal el TCP se desplaza en una línea recta, entre el origen y el destino (Figura A.14).

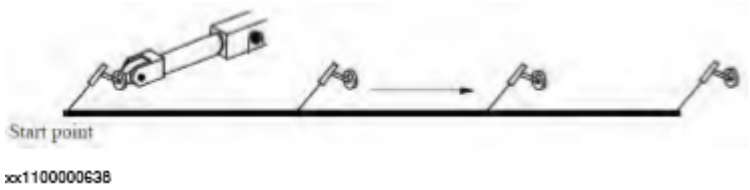

Figura A.14 Interpolación Lineal

Para obtener dicha trayectoria lineal del TCP, los ejes han de seguir una trayectoria no lineal dentro del espacio del eje. Por tanto, cuanto menor es la linealidad de la configuración del robot, mayores son las aceleraciones y deceleraciones para hacer que la herramienta se mueva en una línea recta y se consiga la orientación deseada.

La orientación de la herramienta permanece constante durante todo el movimiento a no ser que se haya programado una reorientación. En ese caso, la herramienta girará a velocidad constante.

- Interpolación circular

Las trayectorias circulares se definen mediante tres posiciones programadas que definen un segmento del círculo, siendo el primero el inicio del segmento, el siguiente es otro punto del circulo empleado para definir la curvatura del círculo y el último indica el final del segmento. Dichos puntos han de estar definidos en intervalos regulares, para conseguir que este sea lo más exacto posible.

Si la orientación programada es la misma en los puntos de inicio y de destino, y la orientación del punto de apoyo se acerca a la misma orientación respecto del círculo, la orientación de la herramienta permanece constante durante toda la trayectoria (Figura A.15).

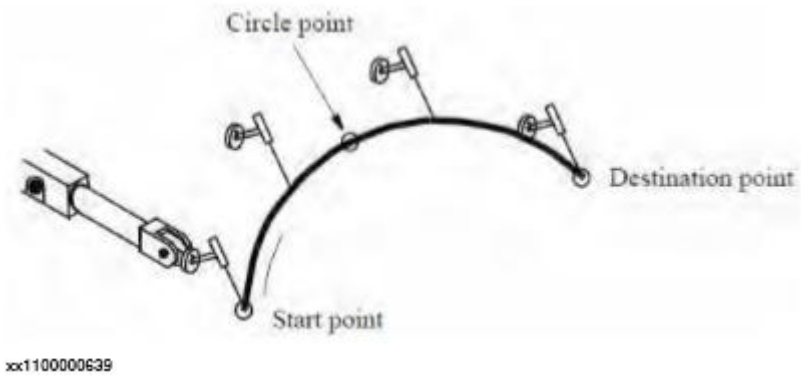

Figura A.15 Interpolación Circular con orientación constante a lo largo de la trayectoria

Sin embargo, si se programa para el punto de apoyo una orientación más próxima a la girada 180º se selecciona el giro alternativo (Figura A.16).

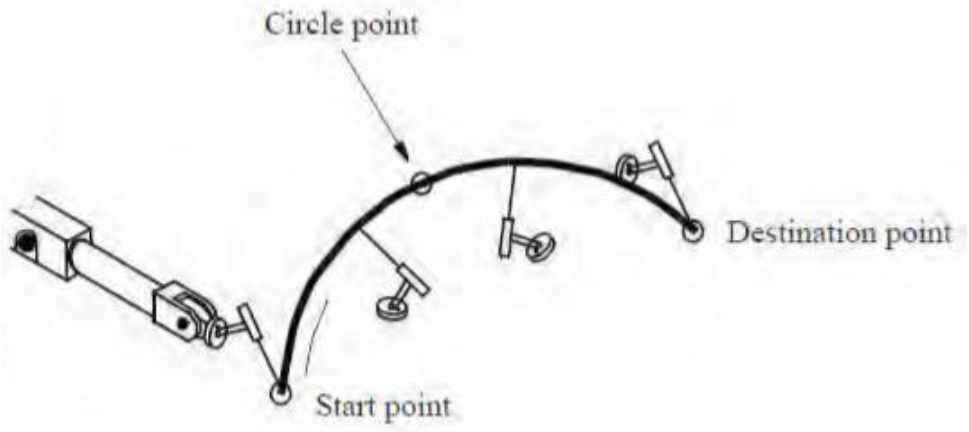

Figura A.16 Interpolación Circular con orientación variable a lo largo de la trayectoria

- SingArea\Wrist

Cerca de una singularidad, las interpolaciones línea o circular pueden ser problemáticas. En este caso, lo mejor es optar por una interpolación modificada, es decir, los ejes de la muñeca se interpolan eje por eje, con el TCP siguiendo una trayectoria lineal o circular. Sin embargo, la orientación de la herramienta será algo distinta a la de la orientación programada.

A.2.2.3 Interpolación de las trayectorias de esquina

El punto de destino se define como un punto de paro para conseguir el movimiento de un punto a otro. Por lo tanto, el robot y todos los ejes adicionales se detienen y no será posible

continuar con el posicionamiento hasta que las velocidades de los ejes sean cero y éstos se encuentren cerca de sus destinos.

Se utilizan puntos de paso para conseguir un movimiento continuo más allá de las posiciones programadas, de esta forma es posible atravesar zonas a alta velocidad sin necesidad de reducirla sin motivo. Dichos puntos de paso generan una trayectoria de esquina (trayectoria de parábola), lo que provoca que realmente el punto de destino nunca se alcance. El inicio y el final de la zona de esquina se definen mediante una zona alrededor de la posición programada, mostrado en la Figura A.16.

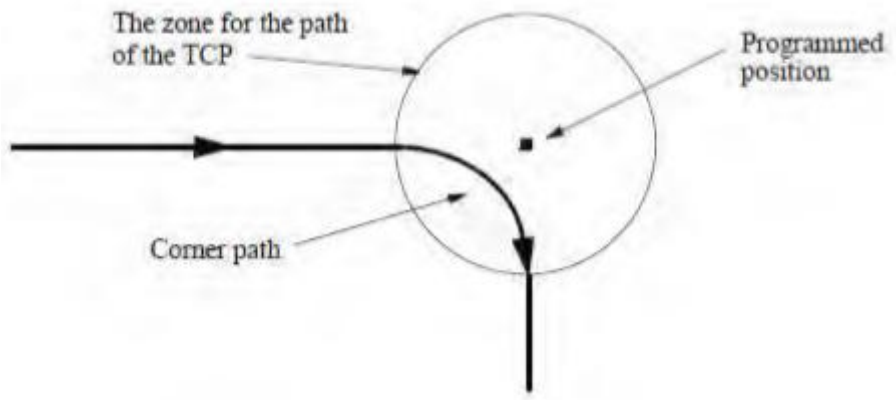

Figura A.17 Zonas de aproximación

- Interpolación de ejes en trayectorias de esquina

El tamaño de las trayectorias de esquina, o zonas, del movimiento del TCP se expresa en mm. Cuando se realiza la interpolación de ejes, el tamaño de dichas zonas hay que recalcularlo a radianes, es decir, hay que pasar de mm a rad, con lo que se comete un error del 10% como máximo, por lo que la zona real se desviará ligeramente de la programada (Figura A.18)

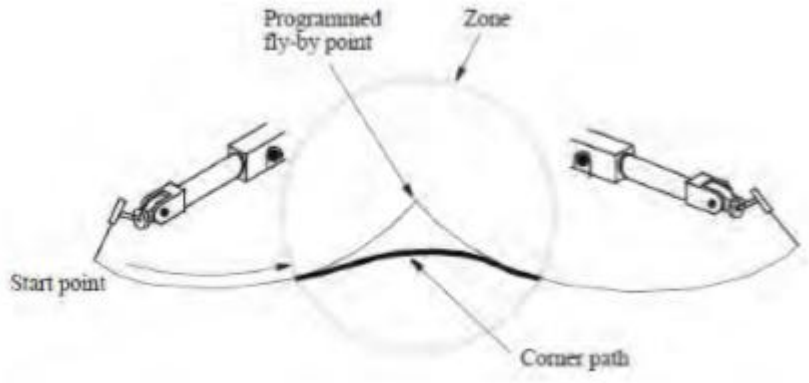

Figura A.18 Interpolación de ejes en trayectorias de esquina

- Interpolación lineal de una posición en trayectoria de esquina

Del mismo modo a como ocurre en la interpolación de ejes, durante la interpolación lineal, al atravesar un punto de paso se genera una trayectoria de esquina (Figura A.19).

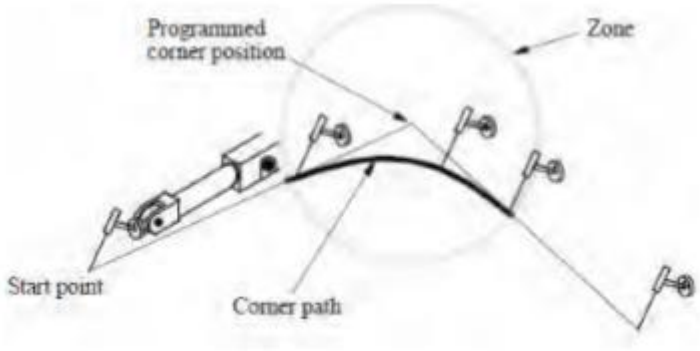

Figura A.19 Interpolación Lineal en trayectorias de esquina

En el caso de que realmente se requiera pasar por la trayectoria en esquina para un proceso determinado (soldadura por arco, pegar un adhesivo…), se puede ajustar los valores de la zona, programando las posicionas más cerca unas de otras, lo que permite aproximarse a la trayectoria deseada mediante dos o más trayectorias más pequeñas.

- Interpolación lineal de una orientación en trayectoria de esquina

Se pueden definir zonas para la orientación del mismo modo que para la posición, siendo las zonas definidas para la orientación, generalmente más grandes que las definidas para la posición.

Al final, la herramienta se reorientara de tal forma que se encuentre del mismo modo a como se encontraría si se hubiese programado como punto de paro, como se muestra en las figuras A.20.

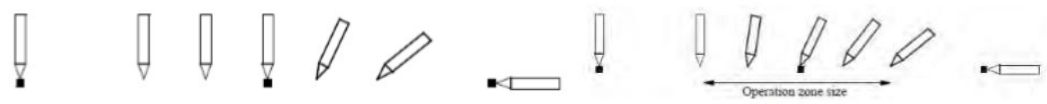

Figura A.20 Interpolación lineal de una orientación en las trayectorias de esquina

- Interpolación de ejes adicionales en zonas de esquina

Se pueden definir zonas para los ejes adicionales, de la misma forma que para la orientación. Si la zona del eje adicional definida es más grande que la zona del TCP, la interpolación de los ejes adicionales hacia el destino comenzará antes de que comience la trayectoria de la esquina del TCP.

Esto puede usarse para suavizar los movimientos de los ejes adicionales, del mismo modo que la zona de orientación se utiliza para suavizar los movimientos de la muñeca.

- Trayectorias de esquina al cambiar de método de interpolación

Las trayectorias de esquina también se generan cuando se cambia de un método de interpolación a otro.

Por ejemplo, en la Figura A.21, se muestra un movimiento del TCP con interpolación lineal entre los puntos P1 y P2, un movimiento con interpolación de ejes entre P2 y P3 y un movimiento con interpolación SingArea\Wrist entre P3 y P4. En dichos cambios de interpolación se produce una zona de esquina.

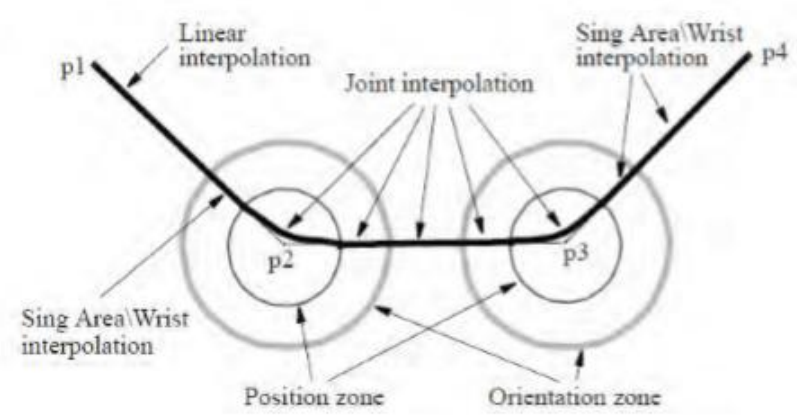

Figura A.21 Ejemplo de Interpolación Lineal entre P1 y P2, un movimiento de interpolación de ejes ente P2 y P3, e interpolación SingArea\Wrist entre P3 y P4.

- Interpolación al cambiar de sistema de coordenadas

Cuando se produce un cambio de sistema de coordenadas en una trayectoria de esquina, por ejemplo, con un nuevo TCP o un nuevo objeto de trabajo, se utiliza la interpolación de ejes de la trayectoria de esquina. Esto también se aplica al cambiar de una operación coordinada a una operación no coordinada y viceversa.

- Trayectoria de esquina con zonas superpuestas

En el caso de que dos posiciones se encuentren muy próximas una de la otra, es común que las zonas programas se solapen. Si se da el caso, el robot, con el fin de conseguir una trayectoria bien definida y a una velocidad óptima en todo el recorrido, recude el tamaño de ambas zonas a la mitad de la distancia que las separa, con el fin de obtener una trayectoria simétrica. En la Figura A.22, se muestra un ejemplo de esto que se acaba de explicar.

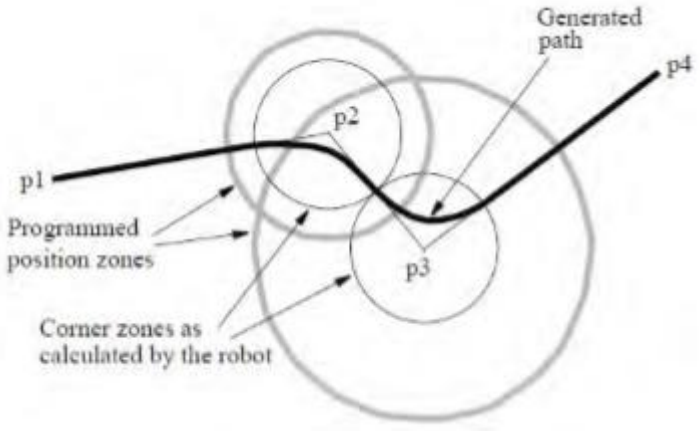

Figura A.22 Trayectorias de esquina, con zonas superpuestas de posición y de orientación

Del mismo modo que ocurre con la posición, ocurre con la orientación. Si se da el caso e el que las zonas de orientación se superpongas, pero las zonas de posición no lo hagan, éstas últimas no se reducen ni modifican su tamano, como muestra la Figura A.23.

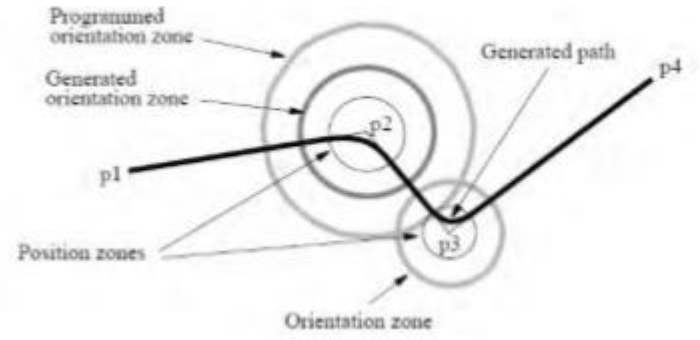

Figura A.23 Trayectorias de esquina con trayectorias de esquina, de orientación superpuestas pero de posición no

A.2.2.4 Ejes independientes

Los ejes independientes son los que se mueven independientemente de los demás ejes del robot. Es posible cambiar un eje a *modo independiente* y volverlo a poner en *modo normal* después.

Ind_move → cambiar modo independiente

IndReset → Volver modo normal

- Ejecución de programa

Si se detiene el programa mientras se está moviendo un eje en modo independiente, dicho eje se detiene, y cuando se reinicia el programa, el eje independiente se pone en movimiento inmediatamente.

Si se produce una caída de alimentación mientras hay un eje en modo independiente, no es posible reanudar el programa, es necesario reiniciarlo desde el principio.

- Ejecución paso a paso

Durante la ejecución paso a paso, un eje independiente se ejecuta sólo cuando se está ejecutando a la vez otra instrucción.

- Movimiento

Los ejes en modo independiente no pueden tener movimientos asignados. Si se intenta ejecutar el eje manualmente éste no se mueve y salta un mensaje de error.

A.2.2.5 Paro y reanudación

- Detención del movimiento
    - Con un paro normal: El robot se detiene sin abandonar la trayectoria, lo que hace que la reanudación del movimiento sea sencilla.
    - Con un paro rígido: El robot se detiene en un plazo más breve que en el paro normal, pero la trayectoria de deceleración no sigue la trayectoria programada. Utilizado para paros de búsqueda, por ejemplo.
    - Con un paro rápido: Se utilizan los frenos mecánicos para conseguir una distancia de deceleración que puede ser tan breve como se especifique para garantizar la seguridad.

- Inicio del movimiento

Después de un par (Cualquiera), la reanudación puede hacerse siempre sobre la trayectoria interrumpida. Sin embargo, si el robot se ha detenido fuera de la trayectoria programada, la reanudación comenzará con el regreso a la posición de la trayectoria en la que debería haberse detenido el robot.

La reanudación después de una caída de alimentación equivale a la de una con paro rápido.

Durante la reanudación, todos los tiempos se cuentan desde el principio, por ejemplo, al realizar el posicionamiento por tiempo o con una interrupción con la instrucción WaitTime.

A.2.3 Sincronización con instrucciones lógicas

- Instrucciones lógicas

Son aquellas instrucciones que no generan ningún movimiento en el robot, ni en ningún eje adicional, es decir, una instrucción de E/S.

- Ejecución secuencial del programa en puntos de paro

Cuando se programa un punto de paro, P1 en el caso de la Figura A.24, la instrucción siguiente no se ejecuta hasta que, tanto el robot como los ejes adicionales se hayan parado.

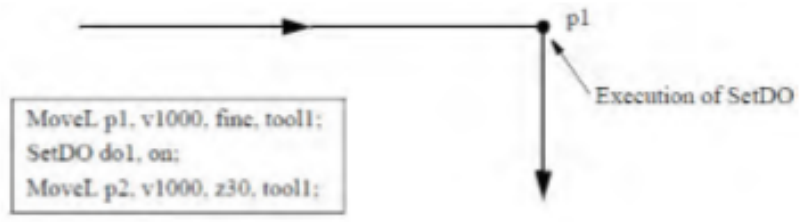

Figura A.24 Ejecución en puntos de paro

- Ejecución secuencial de programas en puntos de paso

Si por el contrario se programa un punto de paso, P1 en la Figura A.25, la instrucción lógica siguiente se ejecuta un tiempo antes de llegar a la zona más grande de esquina programada, en este caso la zona de orientación.

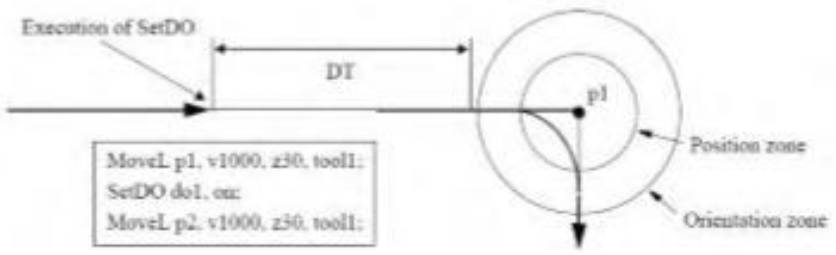

Figura A.25 Ejecución en puntos de paso

- Ejecución simultánea de programas

Se puede programar empleando el argumento *Conc* en la instrucción de posicionamiento. Con ello, lo que consigues es ejecutar una o varias instrucciones lógicas al mismo tiempo que el robot se desplaza, para reducir así el tiempo de ciclo.

En la Figura A.26, se muestra un ejemplo de lo que ocurría si la instrucción de posicionamiento terminase en un punto de paro. Como se puede observar, las instrucciones de posicionamiento y las instrucciones lógicas que van a continuación se ejecutan al mismo tiempo.

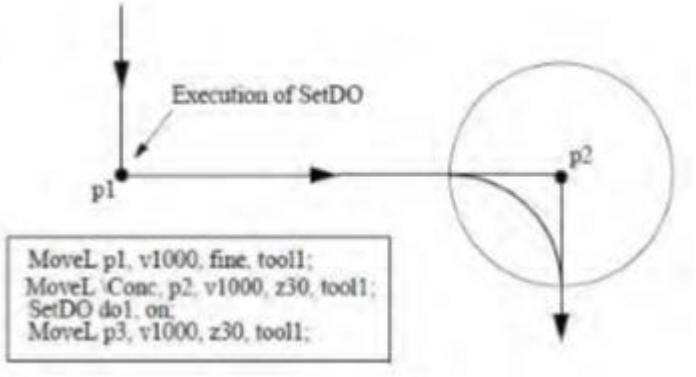

Figura A.26 Ejecución simultanea del programa después de punto de paro

Por el contrario, en el caso de la ejecución simultanea de programas después de un punto de paso, las instrucciones lógicas siguientes empiezan a ejecutarse antes de que comiencen a ejecutarse las instrucciones de posicionamiento que presentan el argumento */Conc*. Tal y como se ilustra en la Figura A.27.

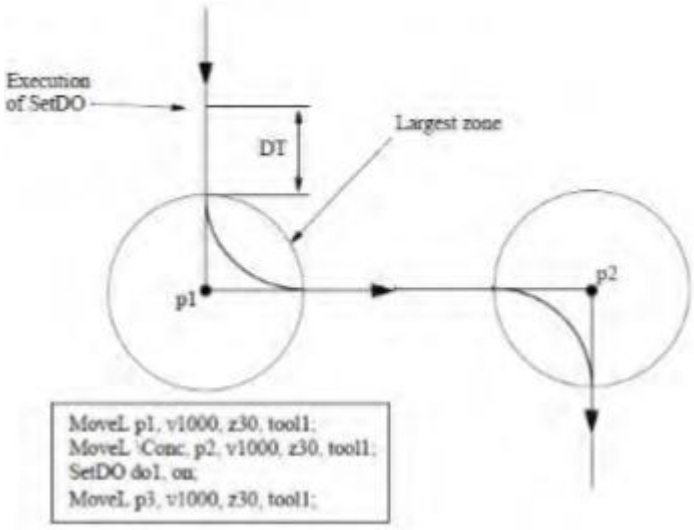

Figura A.27 Ejecución simultanea del programa después de puntos de paso

Del mismo modo, en la Figura A.28, se muestra como se ejecutaría un el programa en el caso de que hubiese, en secuencia, varias instrucciones de posicionamiento con el argumento */Conc* y varias instrucciones lógicas. Como se observa, dichas instrucciones lógicas conectadas se ejecutan en el mismo momento en el que se ejecuta la primera orden de posicionamiento.

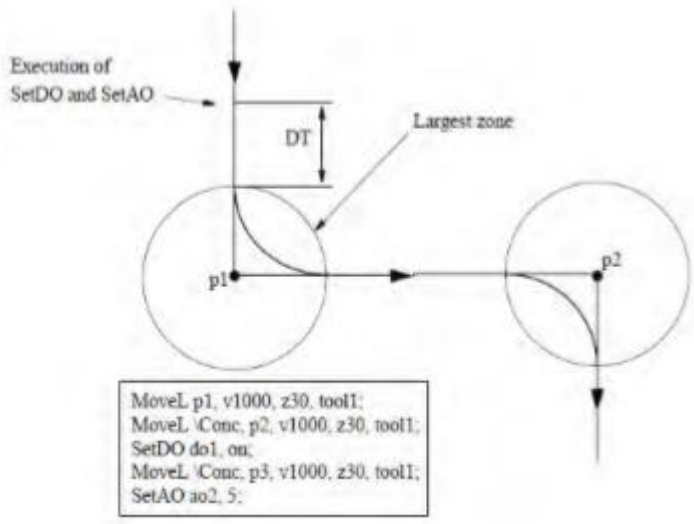

Figura A.28 Ejecución simultanea del programa, con varias instrucciones de posicionamiento y varias instrucciones lógicas

A.2.4 Configuración del Robot

- Distintos tipos de configuraciones de Robot

Normalmente es posible conseguir una misma posición y orientación de la herramienta de varias formas diferentes, utilizando distintos conjuntos de ángulos de ejes. A dichos conjuntos se les llama configuraciones.

En las Figuras A.29 y A.30, se muestran un ejemplo de dos posibles configuraciones del brazo para conseguir la misma posición y un ejemplo sobre dos configuraciones de los ejes de la muñeca para conseguir de nuevo la misma posición, respectivamente.

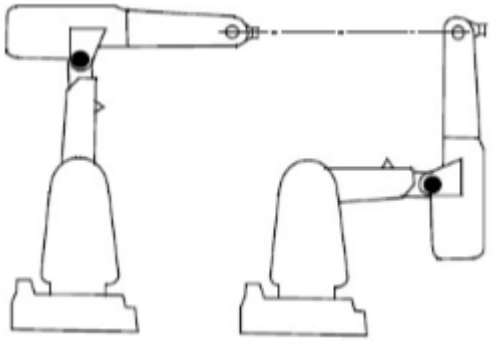

Figura A.29 Posibles configuraciones del brazo

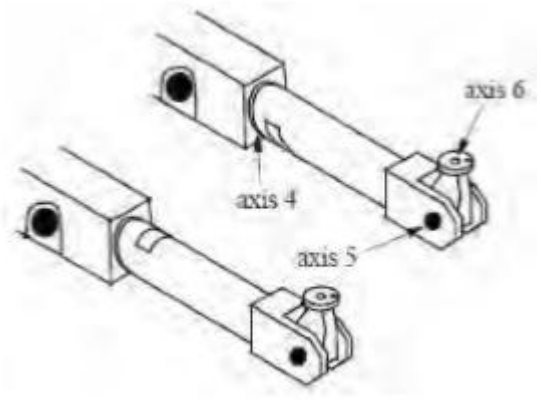

Figura A.30 Posibles configuraciones de la muñeca

- Especificación de la configuración del Robot

Al programar la posición del robot se especifica también una configuración de Robot mediante la instrucción *Conf (cf1, cf4, cf6, cfx).*

La forma de especificar la configuración del Robot es distinta con los distintos tipos de Robots. Para el caso particular del modelo de robot IRB 120 se ha explicado previamente en este trabajo. (*Tipo de dato confdata, apartado A.1.2).*

- Control y monitorización de la configuración

Para lograr la definición correcta del movimiento del robot, es recomendable que el robot mantenga la misma configuración durante la ejecución del programa, que la descrita durante el mismo. Para ello se emplean el control y monitorización de la configuración con *ConfL/on* y *ConfJ/on.*

Durante el movimiento lineal, el robot se mueve siempre hasta alcanzar la configuración más cercana posible. Si está activado el control y monitorización de la configuración del robot, ConfL/on, se realiza una verificación por adelantado para ver si es posible alcanzar la configuración programa. Si no es posible, el programa se detiene. Una vez finalizado el movimiento, también se verifica si el robot ha alcanzado la configuración programa. De nuevo, si no se hubiese alcanzado, el programa se detiene. Por el contrario, si el control y monitorización de la configuración no se encuentra activo, ConfL/off, el robot se mueve a lo largo de una línea recta hasta la posición programada con la posición de los ejes más cercana a la de inicio.

En el movimiento eje por eje, cuando este activado el control y monitorización de la configuración, ConfJ/on, el robot se mueve siempre hasta la configuración especificada. Durante el movimiento, no se realiza ningún control de los movimientos de los ejes. En función de la diferencia entre la configuración del punto de inicio y la configuración del punto final, puede producirse un gran movimiento especialmente de la muñeca. De nuevo, si no está activada la monitorización y el control de la configuración, el robot se moverá hasta el destino, con la posición de los ejes más cercana a la de inicio.

A.2.5 Modelos cinemáticos del Robot

La posición y la orientación de un robot, se determinan a partir de los modelos cinemáticos de su estructura mecánica. Es necesario definir los modelos específicos de la unidad mecánica

en cada instalación, aunque en el caso de los Robots principales y externos estándares de ABB, se encuentran predefinidos en el controlador.

El modelo cinemático de un robot principal, modela la posición y orientación de la herramienta del robot respecto a su base, en función de los ángulos de los ejes del robot.

Dichos parámetros cinemáticos que especifican las longitudes de los brazos, los offsets y las actitudes de los ejes se predefinen en el archivo de configuración de cada tipo de Robot.

En la Figura A.31 se muestra la estructura cinemática del robot IRB 120.

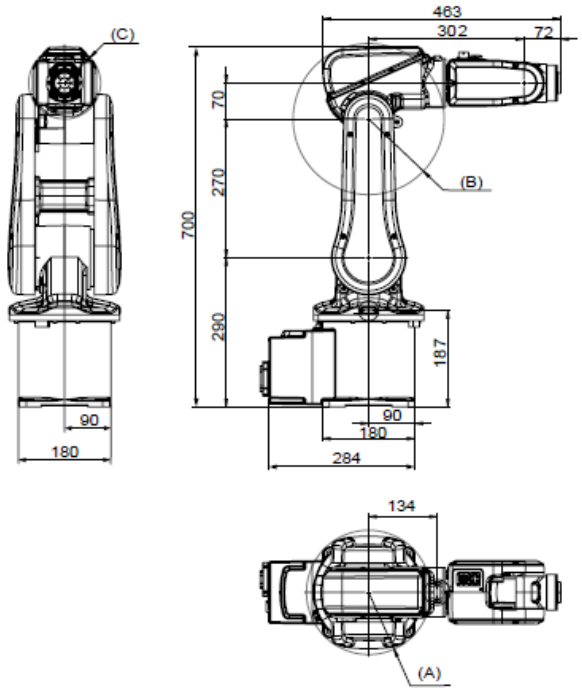

Figura A.31 Estructura cinemática del IRB 120

A.2.6 Singularidades

Se entiende por singularidades aquellas posiciones del espacio de trabajo del robot que pueden ser alcanzadas mediante un número infinito de configuraciones del robot en cuanto al posicionamiento y la orientación de la herramienta.

Por lo general los robots presentan dos tipos de singularidades: las singularidades de brazo, aquellas configuraciones en las que el centro de la muñeca (intersección de los ejes 4,5

y 6) coincide directamente con el eje1, y las singularidades de muñeca, aquellas en las que los ejes 4 y 6 quedan en la misma línea, es decir que el eje 5 tiene un ángulo igual a 0.

Ambas singularidades están indicadas en las Figuras A.32 y A.33 respetivamente.

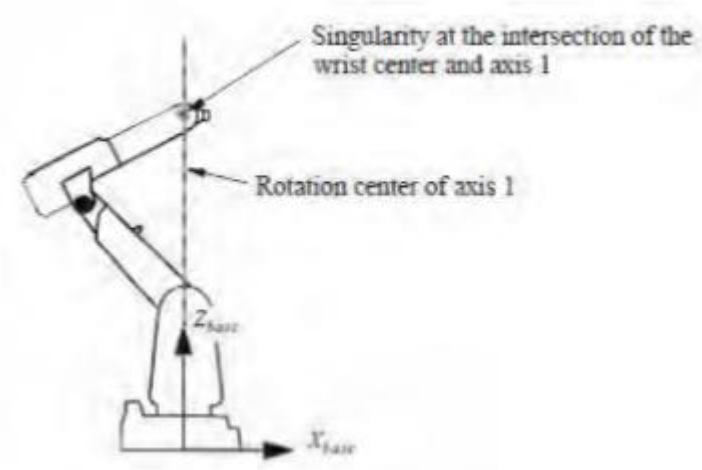

Figura A.31 Singularidad del brazo

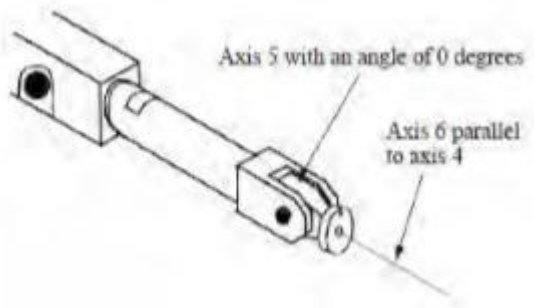

Figura A.32 Singularidad de la muñeca

- Ejecución del programa a través de singularidades

Cuando emplea el movimiento de ejes, moveJ, no se produce ningún problema cuando el robot atraviesa los puntos singulares.

Sin embargo, al ejecutar una trayectoria Lineal o Circular, moveL o moveC, las velocidades de algunos ejes (1 y 4/6 y 6) pueden ser muy elevadas. Por ello, para no superar las velocidades máximas de los ejes, se reduce la velocidad de la trayectoria lineal en general.

Se puede reducir la velocidad de los ejes, mediante el modo SingArea/Wrist (los ejes de la muñeca están interpolados como ángulos de eje), sin perder la trayectoria lineal de la

herramienta del robot. El problema reside en que se introduce un error de orientación en comparación con una interpolación lineal completa.

Anexo B Guía de RobotStudio

Robotstudio se trata de un software de programación de robots industriales, diseñado y patentado por la empresa ABB, el cual permite un abanico de posibilidades en el mundo de la automatización industrial y la robótica manipuladora gracias a la versatilidad de su entorno de programación y al lenguaje de programación RAPID.

Para los programas y estaciones creadas en este trabajo se ha empleado la versión …

A continuación se va a proceder a una explicación paso a paso sobre cómo crear una estación de trabajo para poder simular el tipo de proceso industrial que se desee.

B.1 Creación de una nueva estación

Una vez se tenga el programa instalado en el ordenador y se esté ejecutando el mismo, lo primero va a ser "Crear una nueva estación" tal y como se indica en la Figura B.1

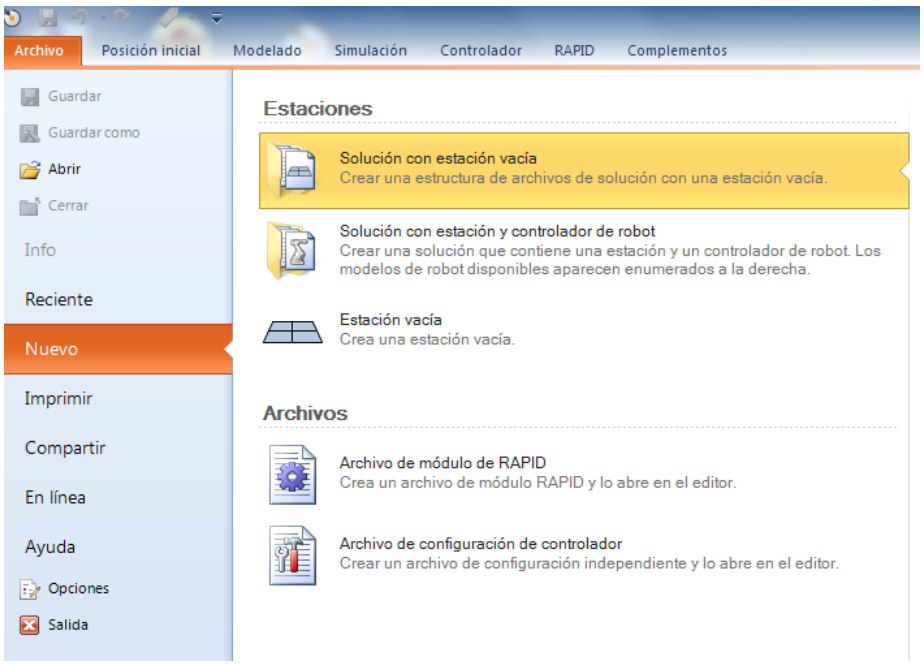

Figura B.1 Creación de una estación

En nuestro caso se va a seleccionar la opción "Estación vacía" a la cual le incluiremos el robot correspondiente.

Una vez se tenga la estación vacía creada, aparecen 7 pestañas distintas que se utilizaran para la creación de nuestra simulación. Estas son: Pestaña Archivo, pestaña Posición Inicial,

pestaña Modelado, pestaña Simulación, pestaña Controlador, pestaña RAPID y por ultimo pestaña Complementos (Figura B.2).

Figura B.2 Pestañas de RobotStudio

B.2 Importar Robot desde Biblioteca

Para poder continuar y trabajar con el robot, se ha de importar el mismo dentro de Robotstudio. Para ello dentro de la pestaña “Posición Inicial”, pinchar sobre Biblioteca ABB. Una vez hecho esto, se desplegara una lista de modelos de robots de ABB, con distintas aplicaciones: Robots manipuladores, de pintura, de posicionamiento y tracks (Figura B.3).

Figura B.3 Importar robot desde biblioteca

Para este ejemplo se va a emplear únicamente robots del primer tipo, robots manipuladores, más concretamente se emplea el Robot de ABB, modelo IRB 120, mostrado en la Figura B.4

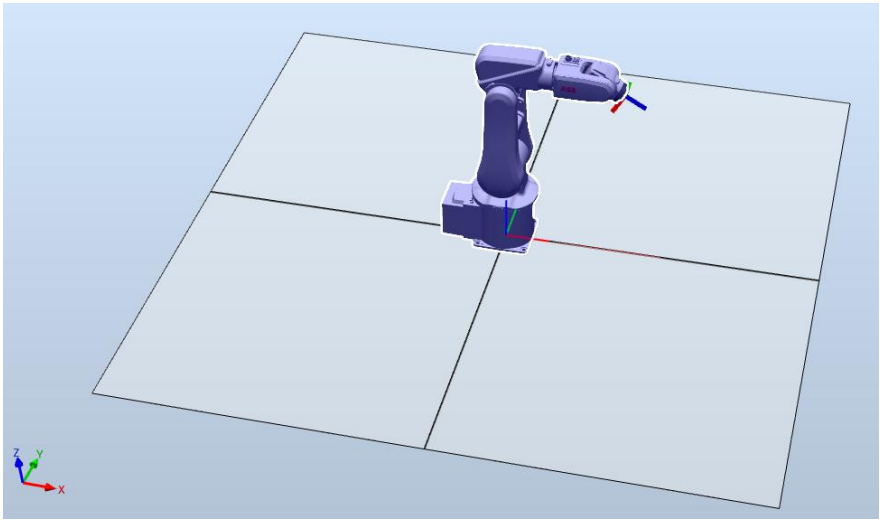

Figura B.4 Robot modelo IRB 120

B.3 Creación de la herramienta

Una vez importado el robot, para poder realizar operaciones y crear objetos de trabajo, se ha de realizar con respecto a una herramienta de trabajo. Esta herramienta se incorpora a la muñeca del mismo, pudiéndose tratar desde una pinza o un imán para manipular objetos, hasta una pistola de soldadura para unir objetos metálicos entre sí.

Si se pincha, dentro de la pestaña Posición Inicial, en “Importar Biblioteca > Equipamiento”, se pueden encontrar multitud de herramientas ya preparadas (Figura B.5).

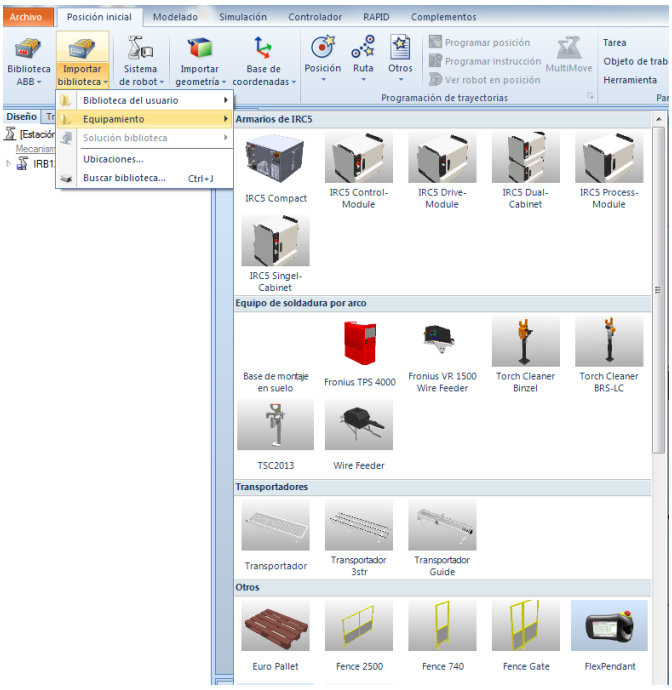

Figura B.5 Importar herramienta desde biblioteca

Otra opción que Robotstudio ofrece, es crear una herramienta a partir de una geometría. Es decir, se puede crear un cilindro, un cono, una pirámide o una geometría importada desde otro programa y emplearla como herramienta de trabajo.

En este ejemplo se va a crear un cono para que sea utilizado como herramienta. Para ello, desde la pestaña Modelado, se pincha sobre “Solido > Cono” (Figura B.6).

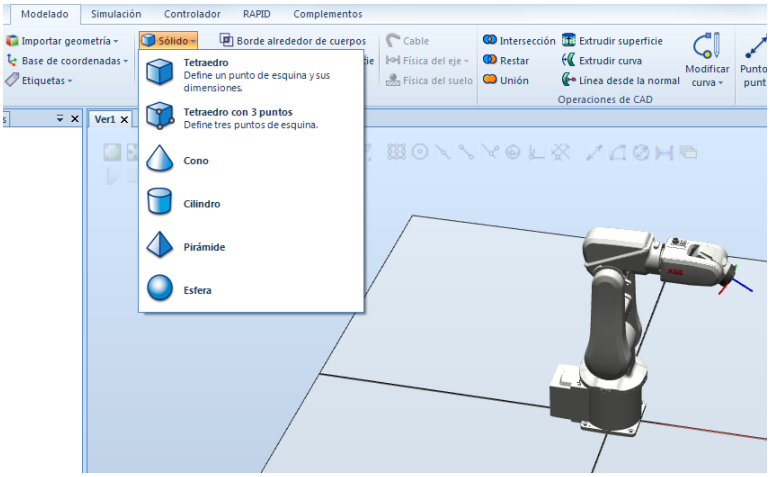

Figura B.6 Creación de sólido (1)

El siguiente paso, es configurar las características del sólido. Las cuales, en el caso del cono, se tratan del diámetro y la altura. Antes de hacerlo, se ha de medir el diámetro de la muñeca del robot, ya que el del cono tendrá que ser de una dimensión similar.

Para realizar dicha medición en Robotstudio, se puede hacer clic, dentro de la ventana de simulación (donde se encuentra situado el Robot), sobre “Punto a punto” y posteriormente sobre “Ajustar a objetos”, tal y como se indica en la figura B.7 y B.8

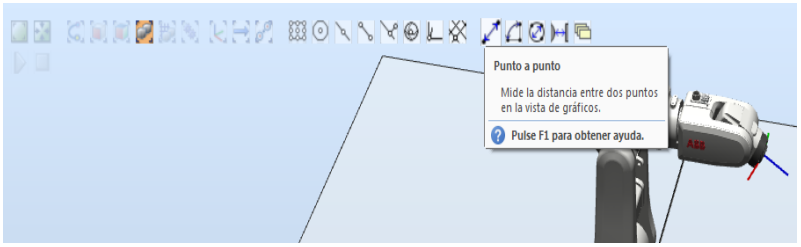

Figura B.7 Medir distancias en RobotStudio (1)

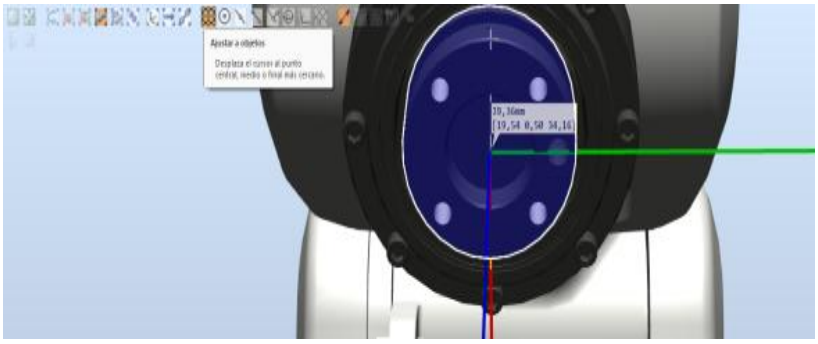

Figura B.8 Medir distancias en RobotStudio (2)

Una vez conocida esa dimensión, se configuran los parámetros del cono. Cabe mencionar, que si no se indica lo contrario, Robotstudio crea por defecto las piezas en la posición [0,0,0], Figura B.9.

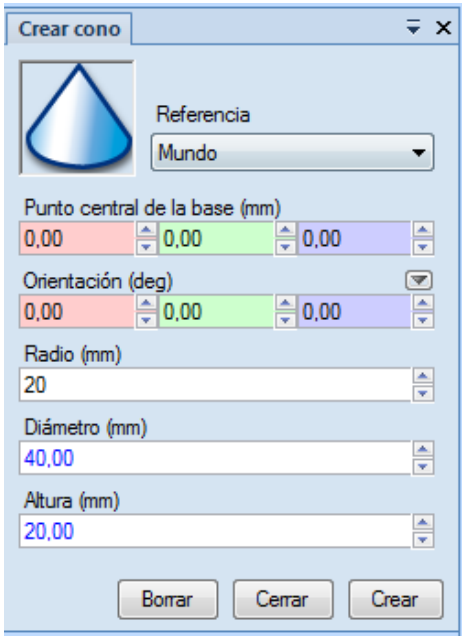

Figura B.9 Creación de sólido (2)

Una vez se ha creado la geometría deseada, el siguiente paso es transformarla en herramienta, sin esto el programa no la detectaría como tal. Para ello, dentro de la pestaña Modelado, se pincha en “Crear herramienta”.

Al hacerlo, aparece el cuadro de dialogo indicado en la Figura B.10. En dicho cuadro, tendremos que darle nombre a la herramienta, en este caso “Cono”, e indicar la opción “Seleccionar pieza existente, en este caso se trata de la geometría ya mencionada.

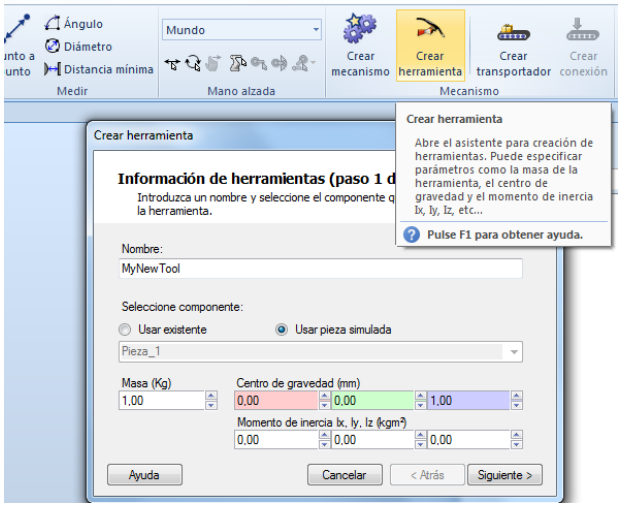

Figura B.10 Creación de la herramienta (1)

Una vez realizado esto, clicar en “siguiente” para continuar con el proceso de creación de la herramienta. El siguiente cuadro de dialogo que aparece, indicado en la Figura 3.10, nos solicita información sobre el TCP (Tool Center Point). Particularmente, se necesita asociarle una posición y una orientación relativas, para que las posiciones del robot que se creen posteriormente sean realizadas con respecto a este punto. En este ejemplo se ha asignado el extremo de cono como TCP. Como se ha asignado una altura de 20mm al cono, se tiene que asociar esa posición en el eje Z. Una vez se haya indicado correcta la posición del TCP y el nombre, realizar clic en la flecha que aparece en la ya mencionada Figura B.11.

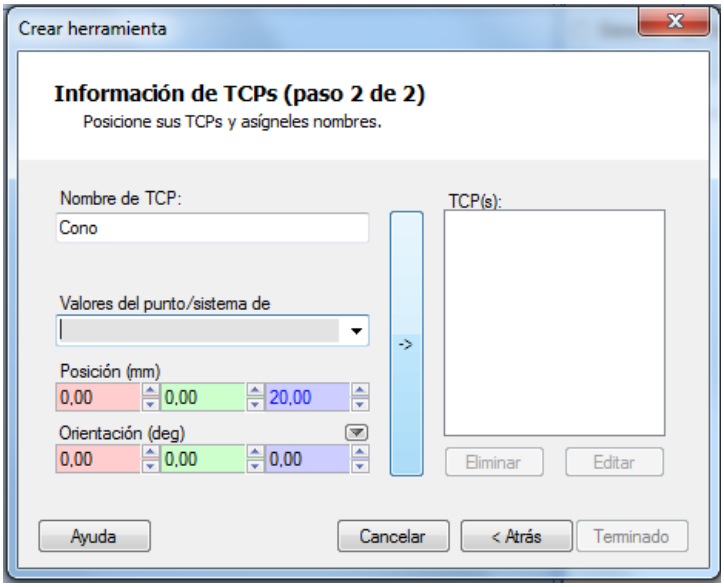

Figura B.11 Creación de la herramienta (2)

Por último, solo queda adjuntar dicha herramienta creada al robot y actualizar su posición para que se sitúe en la muñeca de este. Para adjuntarla, únicamente seleccionar la herramienta que se acaba de crear y arrastrarla hasta el Robot (Figura B.12).

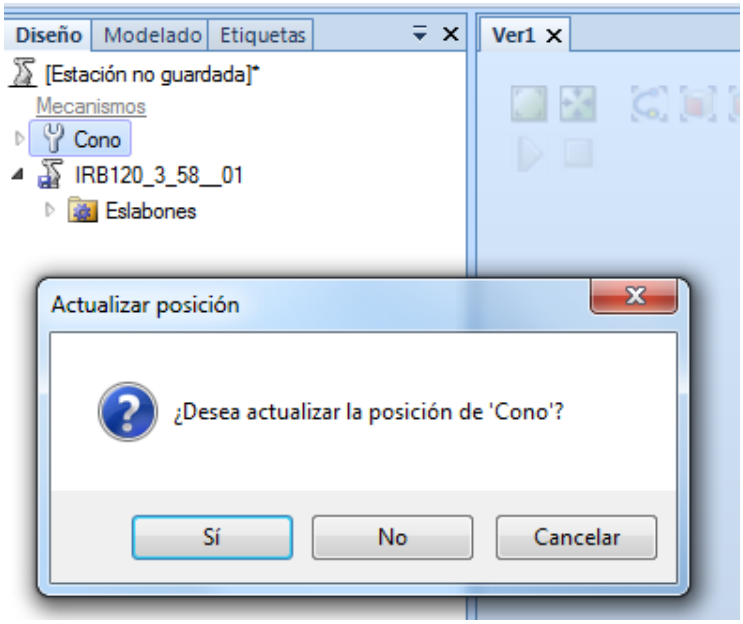

Figura B.12 Conectar la herramienta al Link6 del Robot

Este paso final también es necesario hacerlo si se está importando una herramienta desde la biblioteca.

B.4 Creación del controlador del Robot

En los pasos que se han realizado previamente, se tiene un brazo con una herramienta adjunta y por tanto, no se pueden realizar movimientos ni programar trayectorias. Para poder lograrlo, es necesario crear un controlador asociado a dicho robot.

Para ello, como se indica en la Figura B.13, desde la pestaña Posición Inicial se pincha en “Sistema de Robot > Desde diseño…”

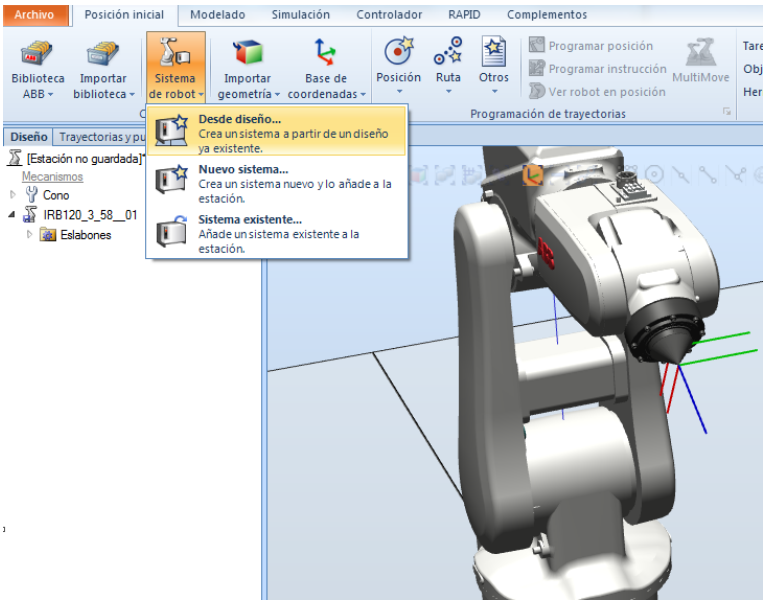

Figura B.13 Creación del controlador (1)

A continuación, nos aparece el cuadro de dialogo mostrado en la figura 3.13, en el que tendremos que indicar el nombre de nuestro controlador, en nuestro caso “Ejemplo_1”. El resto de elementos que allí aparecen los dejaremos como se encontraban por defecto.

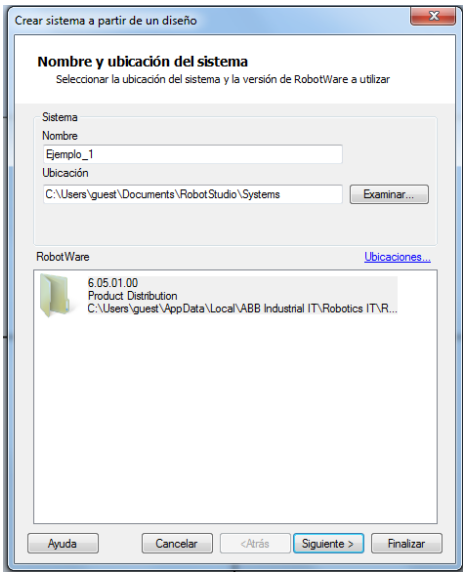

Figura B.14 Creación del controlador (2)

Una vez puesto el nombre, se hace clic en “Siguiente” y en el nuevo cuadro seleccionaremos el mecanismo que queremos controlar, en este caso el IRB 120.

Por último, se pide ajustar la lista de parámetros del sistema a crear. Se trata de una serie de parámetros más complejos, que dadas las aplicaciones sencillas y los objetivos buscados en

este trabajo, no se van a tocar y se van a dejar los que se encontraban por defecto en la Figura B.15.

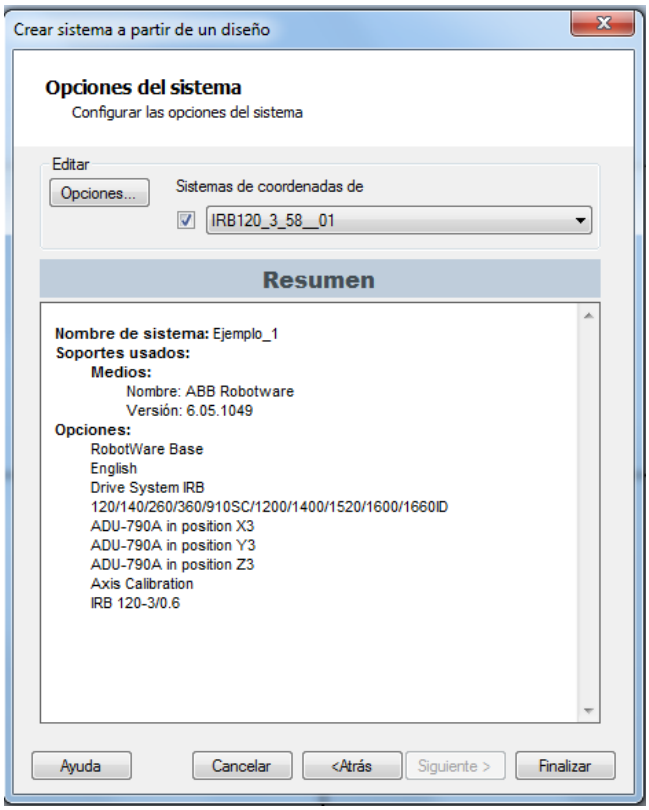

Figura B.15 Creación del controlador (3)

B.5 Creación de objetos de trabajo

Una vez realizados los pasos anteriores, se va a proceder a crear e objeto de trabajo que se vaya a manipular durante la simulación. Las opciones que Robotstudio ofrece para crear sólidos son varias; colorearlos, unirlos unos con otros, hacer intersecciones… etc. Se invita a probar diferentes opciones posibilidades que el programa ofrece, aunque cabe mencionar que una creado el objeto como tal, no se podrá modificar su geometría o sus propiedades.

En este caso, se tratara de un hexaedro de dimensiones de lado 150 mm, cuyo punto de esquina estará situado en [200, 200, 100]. El hexaedro se creará de la misma forma en la que se ha creado el cono previamente, es decir, acudir a la pestaña Modelado, “Sólido > Tetraedro” , incluir las dimensiones ya mencionadas, situar dicho hexaedro en las coordenadas previstas, así como darle un nombre, en este caso “cubo”.

Para crear este objeto de trabajo, se debe acudir a la pestaña Posición Inicial, hacer clic en “Otros > Crear objeto de trabajo” (Figura B.16).

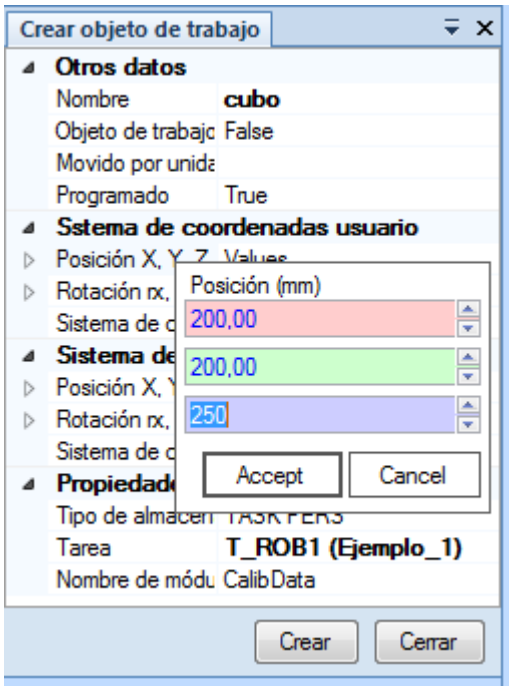

Figura B.16 Creación de un objeto de trabajo

En ese cuadro de diálogo, se deberá introducir su sistema de coordenadas, que para la realización de este ejemplo se ha colocado en [500, 100, 300], es decir en la esquina superior izquierda, como se muestra en la Figura B.17. Para finalizar se hace clic en “Crear” y con ello ya se tendría el objeto de trabajo deseado.

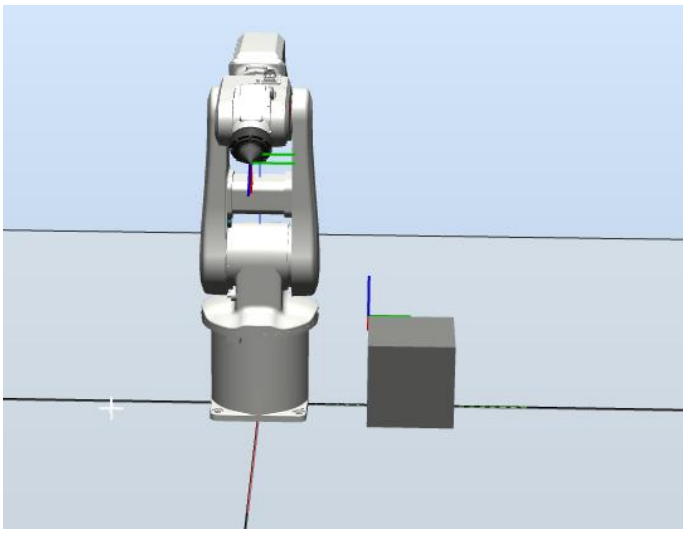

Figura B.17 Posición del objeto de trabajo respecto a la base

Una vez creado dicho objeto de trabajo, es conveniente asegurarse de que, entrando en la pestaña Inicio, en “Parámetros” aparecen marcados “Objeto de trabajo: Cubo” y “Herramienta: Cono”, de la forma indicada en la Figura B.18, para referir las posiciones al Objeto de trabajo y a la Herramienta creada. En caso de no aparecer, se debe retroceder y modificarlo.

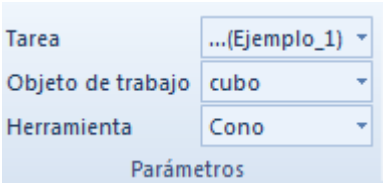

Figura B.18 Seleccionar tareas, objetos de trabajo y herramientas

B.6 Creación de posiciones

El siguiente paso, es crear puntos o posiciones de robot, relativas al Cubo y al cilindro, así como crear una posición inicial del robot relativa al Work Object 0 (wobj0) que se crea de forma predeterminada por el Robot. Para ello, desde la pestaña Posición Inicial, se pincha en “Posición > Crear objetivo” (Figura B.19).

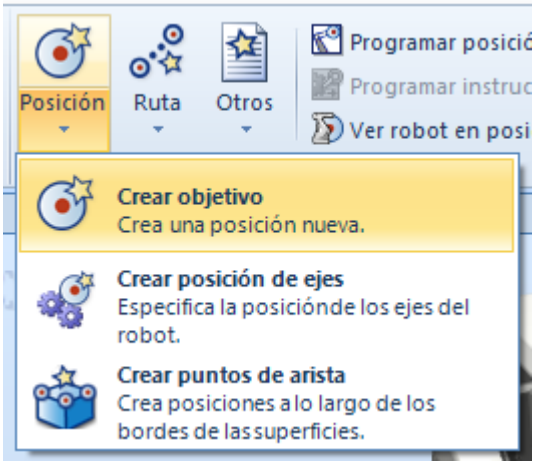

Figura B.19 Crear posiciones objetivo (1)

En el cuadro de dialogo emergente (Figura B.20), se ha de crear, por ejemplo, los cuatro puntos correspondientes a las cuatro esquinas superiores del hexaedro. Para ello, se indica la posición de dichos puntos, uno a uno, y se van añadiendo.

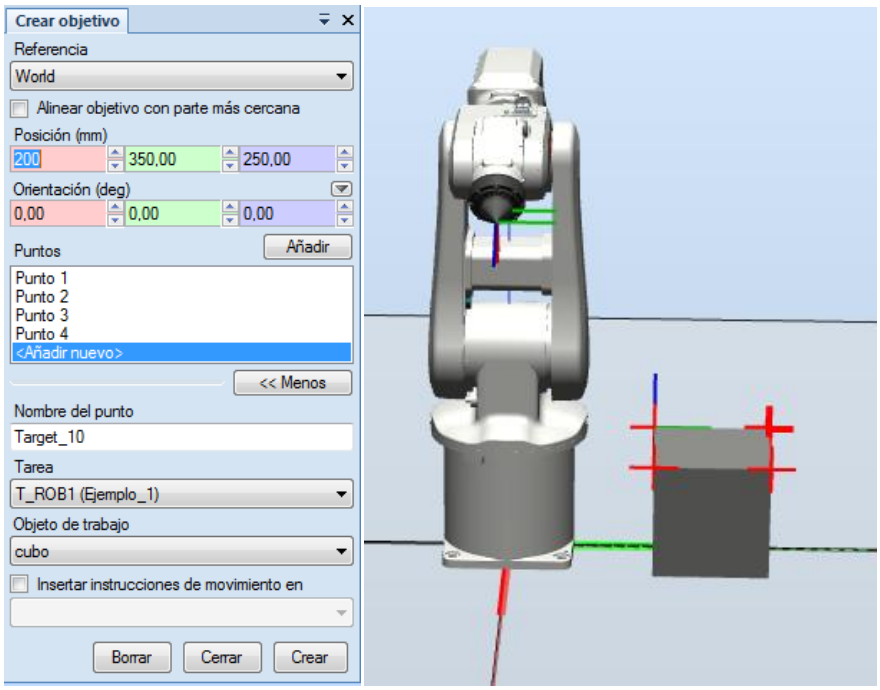

Figura B.20 Crear posiciones objetivo (2)

Antes de hacer clic en “Crear” es conveniente asegurarse de que el objeto de trabajo y la tarea son correctos, que en nuestro caso serían “Cubo” y “Ejemplo” respectivamente. Una vez comprobado, hacer clic en “Crear”.

La posición inicial mencionada previamente, se crea, en primer lugar, situando el robot de forma manual en una posición natural. Para lograrlo, es necesario acudir al apartado “Mano alzada” dentro de la pestaña Posición Inicial, y hacer clic en “movimiento de ejes”. Tras mover dichos ejes a la posición que se desea, ahora pinchar en “Ajustar a objetos”, y en este caso pinchar en el extremo final del cono.

A la hora de crear el punto, hay que comprobar lo mismo que en el caso anterior, pero esta vez el objeto de trabajo será el Work Object 0 (wobj0) mencionado previamente (Figura B.21).

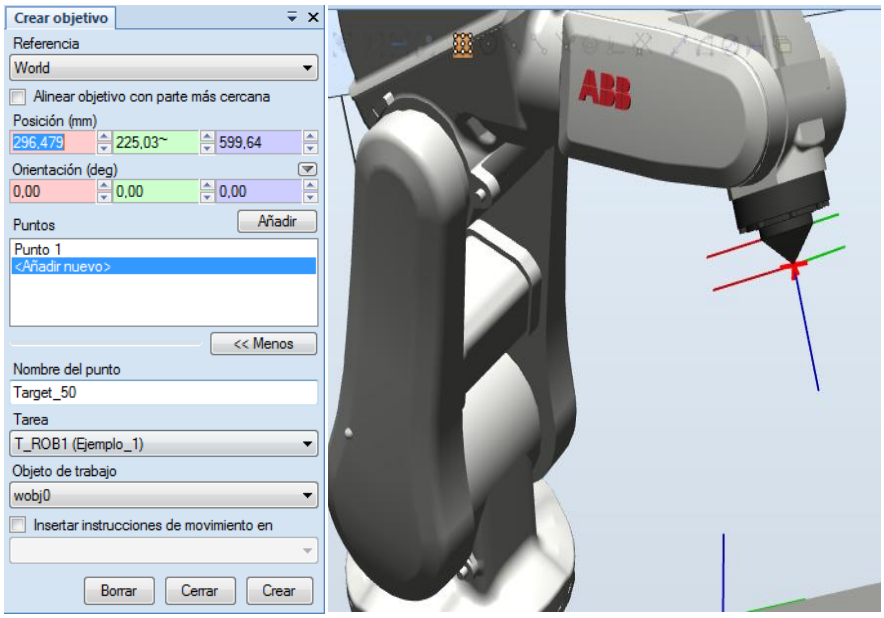

Figura B.21 Crear posición de inicio

B.7 Ver la herramienta en una posición

Previamente a crear una trayectoria, es necesario ver como accede la herramienta a dicha posición, ya que lo más probable es que tanto la orientación como la configuración asignada por defecto, impida al robot acceder a dicha posición. Por tanto, se va a comprobar esto con los 5 puntos creados previamente.

B.7.1 Orientación de la herramienta

Para ello, se acude como indica la Figura B.22 al punto “Pos_ini” (Se le ha dado ese nombre para una mayor aclaración) se hace clic derecho sobre él y se busca la opción “Ver herramienta en la posición > Cono”.

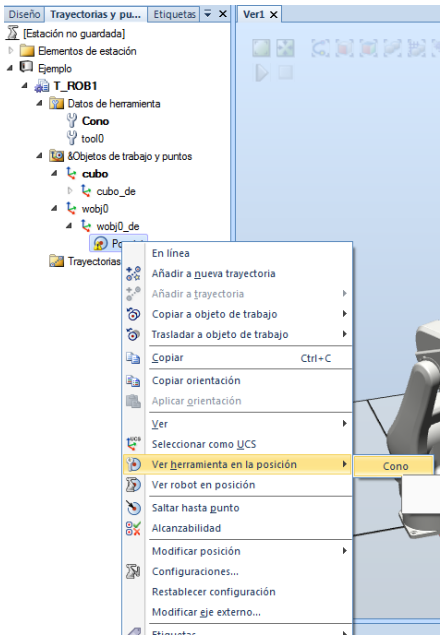

Figura B.22 Comprobar herramienta en posición en la posición inicial (1)

Si al realizar este paso la herramienta se encuentra con una orientación inalcanzable para el robot en dicha posición, algo muy sencillo de comprobar ya que si hace clic en “Ver robot en posición” (Justo debajo de “Ver herramienta en la posición”, en la figura B.22) aparecerá un mensaje de error indicando que el punto está fuera de alcance, y en tal caso, es necesario modificar la orientación.

En este caso de ejemplo caso, en el punto “Pos_ini”, la posición del Robot es alcanzable (Figura B.23), pero no es la que se desea, por tanto se va a modificar

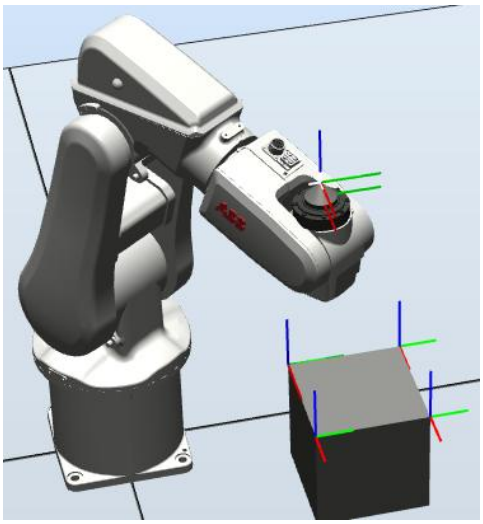

Figura B.23 Comprobar herramienta en posición en la posición inicial (2)

Para cambiarla, se ha de pinchar de nuevo en el punto de correspondiente, igual que se ha indicado previamente, pero en este caso en vez de seleccionar la opción “Ver herramienta en posición”, se selecciona la opción “Modificar posición > Girar” (Figura B.22).

En este ejemplo, lo único que habría que realizar sería rotar 180 grados, alrededor del eje Y, como se indica en la Figura B.24.

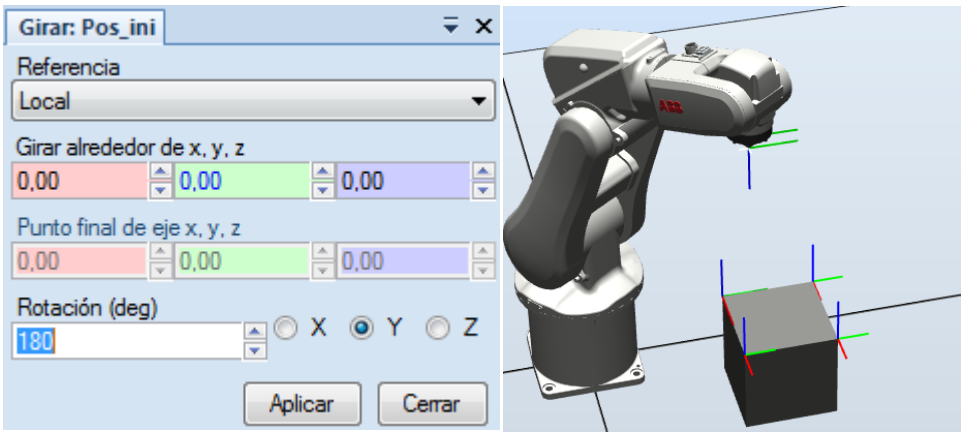

Figura B.24 Reorientar la posición inicial

Ahora para las posiciones que hemos creado para el cubo, se realizarían de forma similar. En este caso, empezando por “Target_10”, se indica que dicha posición es inalcanzable por el robot. Por lo que habría que modificarla, y en este caso se ha rota 180 grados con respecto al eje Y.

Si se quisiese dar a los otros puntos una orientación particular y específica, se tendría que seguir el mismo procedimiento que para el puno “Target_10”. Sin embargo, si se quiere que el robot acceda a esos puntos con la misma orientación que para el anterior, basta con acudir al punto “Target_10”, realizar clic derecho sobre él y seleccionar la opción “Copiar orientación”. Posteriormente, se seleccionan los demás puntos y de nuevo con el botón derecho, hacer clic, e indicar la opción “Aplicar orientación”.

Finalmente, después de este proceso, la herramienta en posición en todos los puntos en los que se ha modificado, se vería de la siguiente forma (Figura B.25).

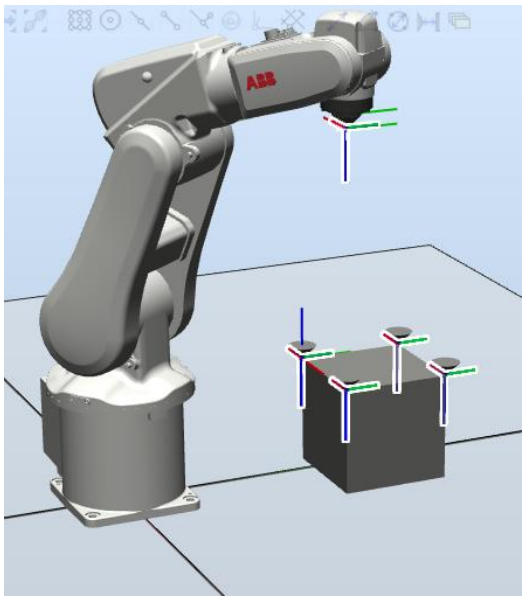

Figura B.25 Reorientación del resto de puntos

B.7.2 Configuración de los ejes del Robot

La configuración con la que el robot accede a dichos puntos también puede no ser la adecuada ya que, por ejemplo, alguna de ellas podría hacer que el robot atravesase el objeto de trabajo. Hasta que no se le asigne una configuración específica a dicha posición, aparecerá un símbolo de advertencia al lado de cada posición.

Como ya se ha mencionado con anterioridad en este trabajo, el robot puede acceder a una misma posición a partir de varias configuraciones de sus ejes. Dichas configuraciones, se designan con una serie de cuatro números enteros que especifican en que cuadrante se encuentran los ejes.

Para acceder a las distintas configuraciones posibles para cada punto, se realiza clic derecho sobre el punto, y se selecciona la opción “configuraciones”. Una vez hecho esto, se abre una nueva ventana (Figura B.26), en la que tendremos que seleccionar una de las configuraciones posibles que aparecen en el primer cuadro.

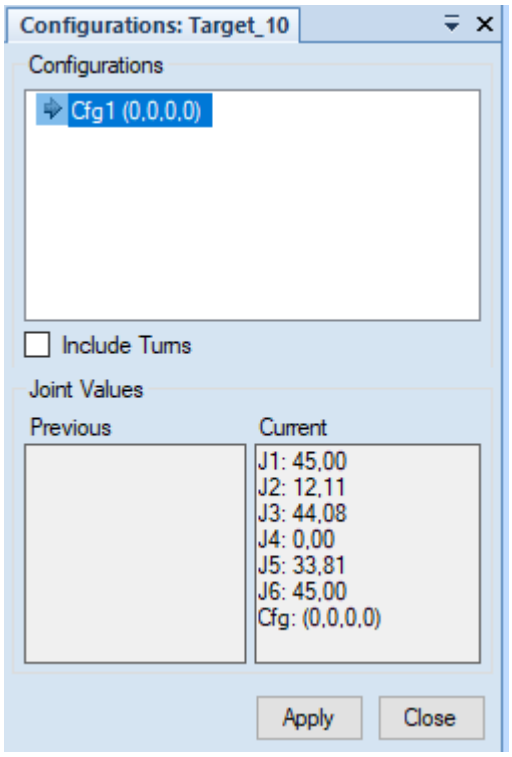

Figura B.26 Seleccionar la configuración del robot en una posición

En nuestro caso, en el "Target_10", al sólo aparecer una configuración, (0,0,0,0), se selecciona esa y se aplica. Si hubiese más de una, se buscaría una configuración con la mayor cantidad posible de ejes en el primer cuadrante.

Del mismo modo, se realizaría con los demás puntos del cubo, así como con "Pos_ini".

B.8 Creación de una trayectoria

Una vez verificado lo anterior, ya se puede crear la trayectoria que el TCP va a recorrer. Para ello, es necesario dirigirse a la pestaña "Posición Inicial" y dentro de ella "Trayectorias y puntos > Trayectorias", hacer clic con el botón derecho y seleccionar "Crear trayectoria" (Figura B.27).

Figura B.27 Creación de una trayectoria

Con esto, se ha creado una trayectoria, cuyo nombre por defecto es “Path_10”. Si se le quiere asociar diversos puntos a dicha trayectoria, simplemente se arrastran los puntos en cuestión hacia “Path_10” en el orden en el que se quiera realizar. En el caso de este ejemplo, el resultado final de esta operación sería el indicado en la Figura B.28.

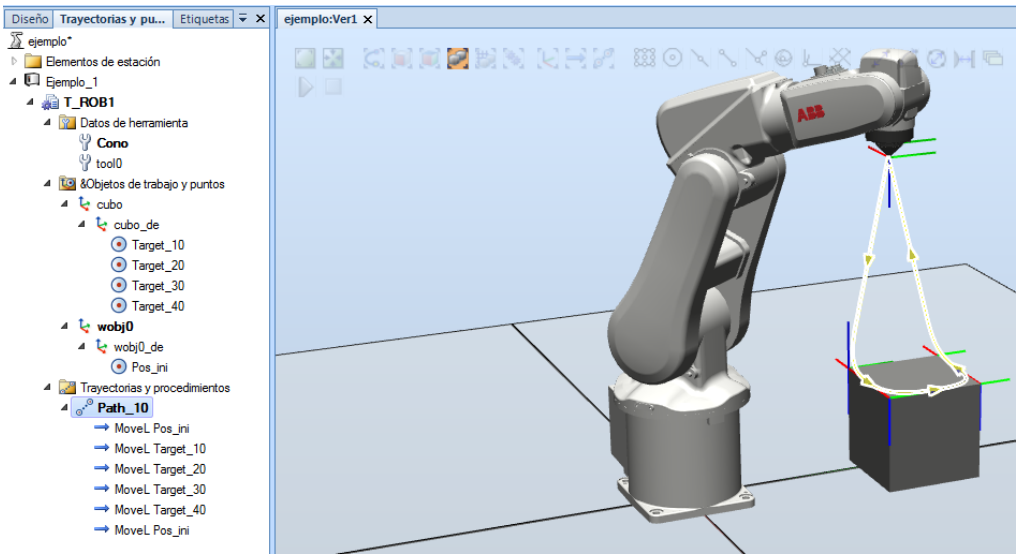

Figura B.28 Trayectoria creada

B.9 Sincronizar con el controlador

Si se quiere emplear el lenguaje de programación RAPID en Robotstudio, primero se tiene que sincronizar la estación creada con el controlador virtual. Para ello, se acude a la pestaña Inicio de nuevo, y dentro de ella “Sincronizar > aceptar” (Figura B.29).

Figura B.29 Sincronización con RAPID

De esta forma, se consigue que la trayectoria “Path_10” se sincronice automáticamente, lo cual es necesario para que aparezcan, en la pestaña RAPID, los módulos de programa “CalibData” y “Module1”, necesarios para programar el robot.

Para poder programar como tal y poder utilizar los comandos mencionados previamente en este trabajo, se ha de acudir a la pestaña RAPID, y en la ventana que aparece en la izquierda, se hace clic en “Controlador > Estación actual (Ejemplo) > RAPID > T_ROB1 > Módulos de programa (Module1). Con ello nos aparece el código de programación en RAPID, que se puede modificar según se quiera. Mencionar que en él ya está incluidos los comandos de definición de las posiciones y la trayectoria, según el tipo de movimiento que hemos determinado, así como sus propiedades (Figura B.30).

```
MODULE Module1
    CONST robtarget Pos_ini:=[[296.479,225.031,599.64],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_10:=[[0,0,0],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_20:=[[150,0,0],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_30:=[[150,150,0],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_40:=[[0,150,0],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    !***********************************************************
    !
    ! Módulo:  Module1
    !
    ! Descripción:
    !   <Introduzca la descripción aquí>
    !
    ! Autor: guest
    !
    ! Versión: 1.0
    !
    !***********************************************************


    !***********************************************************
    !
    ! Procedimiento Main
    !
    !   Este es el punto de entrada de su programa
    !
    !***********************************************************
    PROC main()
        !Añada aquí su código
        Path_10;
    ENDPROC
    PROC Path_10()
        MoveL Pos_ini,v1000,z100,Cono\WObj:=wobj0;
        MoveL Target_10,v1000,z100,Cono\WObj:=cubo;
        MoveL Target_20,v1000,z100,Cono\WObj:=cubo;
        MoveL Target_30,v1000,z100,Cono\WObj:=cubo;
        MoveL Target_40,v1000,z100,Cono\WObj:=cubo;
        MoveL Pos_ini,v1000,z100,Cono\WObj:=wobj0;
    ENDPROC
ENDMODULE
```

Figura B.30 Código en RAPID

Una vez hecho esto, se puede iniciar la simulación, realizando clic en el botón de “inicio”, mostrado en la Figura B.31.

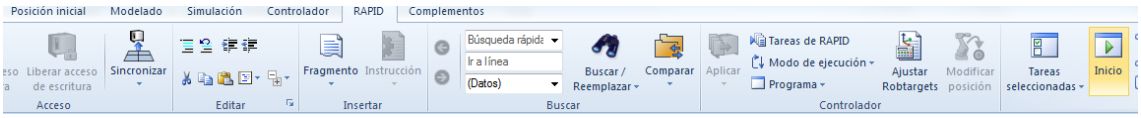

Figura B.31 Inicio de la simulación

Anexo C: DemoFlexPendant

```
MODULE FlexPendant

    !***********************************************************

    !

    ! Módulo: FlexPendant

    !
```

```
        ! Descripción:
        !   <Introduzca la descripción aquí>
        !
        ! Autor: guest
        !
        ! Versión: 1.0
        !
        !*********************************************************


    PERS tooldata
MyTool:=[TRUE,[[31.792631019,0,229.638935148],[0.945518576,0,0.325568154,0]],[1,[0,0,1],
[1,0,0,0],0,0,0]];
    PERS wobjdata Sistema_1:=[FALSE,TRUE,"",[[0,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
        CONST robtarget Pos_reposo:=[[0,-
547.331,451.647],[0.134922335,0.694115238,0.694115238,-0.134922335],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
        CONST robtarget Pos_Letra:=[[200,-400,250],[0.5,-0.5,-0.5,0.5],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
        CONST robtarget Pos_Contorno:=[[0,-400,325],[0,0.707106781,0.707106781,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
        CONST robtarget Pos_Taladro:=[[-200,-400,250],[0.5,0.5,0.5,0.5],[-2,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


        !*********************************************************
```

```
!
! Procedimiento Main
!
!   Este es el punto de entrada de su programa
!
!*********************************************************
PROC main()
   ConfL\on;
   SingArea\Wrist;
   FOR i FROM 1 TO 20 DO
      IF SignalTaladro = 1 THEN
         taladro;
         Reposo;
      ELSEIF SignalContorno = 1 THEN
         Contorno;
         Reposo;
      ELSEIF SignalLetra = 1 THEN
         Letra;
         Reposo;
      ELSE Reposo;
      ENDIF
   ENDFOR
```

```
ENDPROC

PROC Reposo()

   MoveJ Pos_reposo,v1000,z100,MyTool\WObj:=wobj0;

ENDPROC

PROC Letra()

   MoveL Offs(Pos_Letra,0,0,200),v1000,z100,MyTool\WObj:=Sistema_1;

   MoveL Pos_Letra,v1000,z100,MyTool\WObj:=Sistema_1;

   MoveL offs(Pos_Letra,-50.01,-50,-50),v200,fine,MyTool\WObj:=Sistema_1;

   SetDO Pinta,1;

   MoveL offs(Pos_Letra,-50.01,-50,50),v200,fine,MyTool\WObj:=Sistema_1;

   MoveL offs(Pos_Letra,-50.01,0,25),v200,fine,MyTool\WObj:=Sistema_1;

   MoveL offs(Pos_Letra,-50.01,50,50),v200,fine,MyTool\WObj:=Sistema_1;

   MoveL offs(Pos_Letra,-50.01,50,-50),v200,fine,MyTool\WObj:=Sistema_1;

   SetDO Pinta,0;

   MoveJ Pos_Letra,v1000,z100,MyTool\WObj:=Sistema_1;

   MoveL Offs(Pos_Letra,0,0,200),v1000,z100,MyTool\WObj:=Sistema_1;

ENDPROC

PROC Contorno()

   MoveJ Offs(Pos_Contorno,0,0,100),v1000,z100,MyTool\WObj:=Sistema_1;

   MoveJ Offs(Pos_Contorno,150,-150,0.01),v1000,z0,MyTool\WObj:=Sistema_1;

   MoveL Offs(Pos_Contorno,150,150,0.01),v1000,z0,MyTool\WObj:=Sistema_1;

   MoveL Offs(Pos_Contorno,-150,150,0.01),v1000,z0,MyTool\WObj:=Sistema_1;
```

```
    MoveL Offs(Pos_Contorno,-150,-150,0.01),v1000,z0,MyTool\WObj:=Sistema_1;
    MoveL Offs(Pos_Contorno,150,-150,0.01),v1000,z0,MyTool\WObj:=Sistema_1;
    MoveJ Offs(Pos_Contorno,0,0,100),v1000,z100,MyTool\WObj:=Sistema_1;
  ENDPROC
  PROC Taladro()
    MoveL Offs(Pos_Taladro,0,0,200),v1000,z100,MyTool\WObj:=Sistema_1;
    MoveL Pos_Taladro,v1000,z100,MyTool\WObj:=Sistema_1;
    MoveL Offs(Pos_Taladro,100,0,0),v200,fine,MyTool\WObj:=Sistema_1;
    MoveL Pos_Taladro,v200,fine,MyTool\WObj:=Sistema_1;
    MoveL Offs(Pos_Taladro,0,0,200),v1000,z100,MyTool\WObj:=Sistema_1;
  ENDPROC
ENDMODULE
```

Anexo D: Solución PintaNombre

```
MODULE PintaNombre
!*********************************************************
  !
  ! Módulo:  Pintanombre
  !
  !
  ! Autor: Pablo Molina Mata
  !
  ! Versión: 1.0
  !
```

```
!*********************************************************

        PERS tooldata
Pen_TCP:=[TRUE,[[110,0,140],[0.707106781,0,0.707106781,0]],[1,[10.7435139,0,140],[1,0,0,0]
,0,0,0]];

        TASK PERS wobjdata
MesaPiezas:=[FALSE,TRUE,"",[[600,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

        CONST robtarget P1:=[[-100,-200,0],[0.258819045,0,-0.965925826,0],[-1,1,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget P2:=[[-100,-400,0],[0.258819045,0,-0.965925826,0],[-1,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget P4:=[[-150,-350,0],[0.258819045,0,-0.965925826,0],[-1,1,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget P3:=[[-100,-300,0],[0.258819045,0,-0.965925826,0],[-1,1,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget A1:=[[-250,-250,0],[0.25,-0.25,-0.933012702,-0.066987298],[-1,-
1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget A2:=[[-350,-450,0],[0.25,-0.25,-0.933012702,-0.066987298],[-1,-
1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget A3:=[[-450,-250,0],[0.25,-0.25,-0.933012702,-0.066987298],[-1,-1,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget A4:=[[-400,-350,0],[0.25,-0.25,-0.933012702,-0.066987298],[-1,-
1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget A5:=[[-300,-350,0],[0.25,-0.25,-0.933012702,-0.066987298],[-1,-
1,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget Pos_reposo:=[[-600,-500,300],[0.183012702,-0.683012702,-
0.683012702,-0.183012702],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget B1:=[[-550,-300,0],[0.183012702,-0.683012702,-0.683012702,-
0.183012702],[-1,0,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget B2:=[[-550,-400,0],[0.183012702,-0.683012702,-0.683012702,-
0.183012702],[-1,0,-1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget B3:=[[-550,-500,0],[0.183012702,-0.683012702,-0.683012702,-
0.183012702],[-1,0,-1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget B4:=[[-600,-450,0],[0.183012702,-0.683012702,-0.683012702,-
0.183012702],[-1,0,-1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
        CONST robtarget B5:=[[-600,-350,0],[0.183012702,-0.683012702,-0.683012702,-
0.183012702],[-1,0,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget L1:=[[-750,-250,0],[0,0.965925826,0,0.258819045],[-2,0,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget L2:=[[-750,-450,0],[0.183012702,-0.683012702,-0.683012702,-
0.183012702],[-2,1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget L3:=[[-850,-250,0],[0,0.965925826,0,0.258819045],[-2,0,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget O1:=[[-1000,-200,0],[0,0.965925826,0,0.258819045],[-2,0,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget O2:=[[-900,-300,0],[0,0.965925826,0,0.258819045],[-2,0,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget O3:=[[-1000,-400,0],[0,0.965925826,0,0.258819045],[-2,0,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

        CONST robtarget O4:=[[-1100,-300,0],[0,0.965925826,0,0.258819045],[-2,0,-
1,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];



    !***********************************************************

    !

    ! Procedimiento Main

    !

    !   Este es el punto de entrada de su programa

    !

    !***********************************************************


    PROC main()

       confL\on;

       SingArea\Wrist;

       IrReposo;

       P;

       A;
```

```
  B;
  L;
  O;
  IrReposo;
ENDPROC
  PROC P()
     Movej Offs(P1,0,0,100),v500,fine,Pen_TCP\WObj:=MesaPiezas;
     setDO pinta,1;
     MoveL P1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL P2,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveC P4,P3,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL P1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     setDo pinta,0;
     MoveL Offs(P1,0,0,100),v200,fine,Pen_TCP\WObj:=MesaPiezas;
  ENDPROC
  PROC A()
    MoveJ Offs(A1,0,0,100),v500,fine,Pen_TCP\WObj:=MesaPiezas;
     SetDO pinta,1;
     MoveL A1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL A2,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL A3,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL A4,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL A5,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL A4,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     MoveL A3,v200,fine,Pen_TCP\WObj:=MesaPiezas;
     SetDO pinta, 0;
    MoveL offs(A3,0,0,100),v200,fine,Pen_TCP\WObj:=MesaPiezas;
  ENDPROC
```

```
PROC IrReposo()
   MoveJ Pos_reposo,v600,z100,Pen_TCP\WObj:=MesaPiezas;
ENDPROC
PROC B()
   MoveJ Offs(B1,0,0,100),v500,fine,Pen_TCP\WObj:=MesaPiezas;
   SetDO pinta,1;
   MoveL B1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveL B2,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveL B3,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveC B4,B2,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveC B5,B1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   SetDO pinta, 0;
   MoveL offs(B1,0,0,100),v200,fine,Pen_TCP\WObj:=MesaPiezas;
ENDPROC
PROC L()
   MoveJ Offs(L1,0,0,100),v500,fine,Pen_TCP\WObj:=MesaPiezas;
   SetDO pinta,1;
   MoveL L1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveL L2,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveL L1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   MoveL L3,v200,fine,Pen_TCP\WObj:=MesaPiezas;
   SetDO pinta, 0;
   MoveL offs(L3,0,0,100),v200,fine,Pen_TCP\WObj:=MesaPiezas;
ENDPROC
PROC O()
   MoveJ Offs(O1,0,0,100),v500,fine,Pen_TCP\WObj:=MesaPiezas;
   SetDO pinta,1;
   MoveL O1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
```

```
        MoveC O2,O3,v200,fine,Pen_TCP\WObj:=MesaPiezas;
        MoveC O4,O1,v200,fine,Pen_TCP\WObj:=MesaPiezas;
        SetDO pinta, 0;
        MoveL offs(O1,0,0,100),v200,fine,Pen_TCP\WObj:=MesaPiezas;
    ENDPROC
ENDMODULE
```

Anexo E: Solución Demoensamblado

```
MODULE Demoensamblado

    !

    !*********************************************************

    !

    ! Módulo:  Demoensamblado

    !

    !

    ! Autor: Pablo Molina Mata

    !

    ! Versión: 1.0

    !

    !*********************************************************


    TASK PERS wobjdata Mesa_2:=[FALSE,TRUE,"",[[-
350,250,50],[0.707106781,0,0,0.707106781]],[[0,0,0],[1,0,0,0]]];

    TASK PERS wobjdata Mesa_1:=[FALSE,TRUE,"",[[250,-
150,50],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];

    CONST robtarget junta:=[[125,-125,225],[0.707106781,0,0,0.707106781],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget piezagrande:=[[150,225,225],[0.707106781,0,0,0.707106781],[0,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    CONST robtarget piezas:=[[125,-285.617,225],[0.707106781,0,0,0.707106781],[1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
CONST robtarget pos_reposo:=[[221.221,342.409,569],[0.889696159,0.115336203,-
0.189007304,0.399267509],[0,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget perno1:=[[125,-400,225],[0.707106781,0,0,0.707106781],[0,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget perno2:=[[125,-450,225],[0.707106781,0,0,0.707106781],[0,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

PERS tooldata T_PinzaAnalogica2:=[TRUE,[[0,0,50],[0,0,1,0]],[1,[0,0,30],[1,0,0,0],0,0,0]];

  PROC main()

    SingArea\Off;

    ConfL\On;

    SetDO cerrar_pinza,1;

    SetDO cerrar_pinza,0;

    reposo;

    monta_junta;

    monta_pieza1;

    monta_pieza2;

    monta_perno1;

    monta_perno2;

    reposo;

    WaitTime 1;

    Desmonta_perno2;

    Desmonta_perno1;

    Desmonta_pieza2;
```

```
    Desmonta_pieza1;
    Desmonta_junta;
    reposo;
ENDPROC
PROC monta_junta()
    MoveJ junta,v500,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveL offs(junta,0,0,-162.5),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    SetDO cerrar_pinza,1;
    WaitDI pillada,1;
    MoveL junta,v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveJ piezagrande,v500,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
    MoveL offs(piezagrande,0,0,-137.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    SetDO cerrar_pinza,0;
     WaitAI AnchoPieza, \GT,89;
    MoveL piezagrande,v200,z5,T_PinzaAnalogica2\WObj:=Mesa_1;
ENDPROC
PROC monta_pieza1()
    MoveJ piezas,v500,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveL offs(piezas,0,0,-137.5),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    SetDO cerrar_pinza,1;
    WaitDI pillada,1;
    MoveL piezas,v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
```

```
        MoveJ piezagrande,v500,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveJ offs(piezagrande,0, -35.62, 0),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveL offs(piezagrande,0,-35.62,-
112.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        SetDO cerrar_pinza,0;
        WaitAI AnchoPieza, \GT,89;
        MoveL piezagrande,v200,z5,T_PinzaAnalogica2\WObj:=Mesa_1;
    ENDPROC
    PROC monta_pieza2()
        MoveJ piezas,v500,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
        MoveL offs(piezas,0,0,-162.5),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
        SetDO cerrar_pinza,1;
        WaitDI pillada,1;
        MoveL piezas,v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
        MoveJ piezagrande,v500,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveJ offs(piezagrande,0, 39.38, 0),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveL offs(piezagrande,0,39.38,-
112.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        SetDO cerrar_pinza,0;
        WaitAI AnchoPieza, \GT,89;
        MoveL piezagrande,v200,z5,T_PinzaAnalogica2\WObj:=Mesa_1;
    ENDPROC
    PROC monta_perno1()
```

```
    MoveJ offs(perno1,0,0,25),v500,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveL offs(perno1,0,0,-112.5),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    SetDO cerrar_pinza,1;
    WaitDI pillada,1;
    MoveL offs(perno1,0,0,25),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveJ offs(piezagrande,0,0,100),v500,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    MoveJ offs(piezagrande,0, 50, 100),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    MoveL offs(piezagrande,0,50,-87.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    SetDO cerrar_pinza,0;
    WaitAI AnchoPieza, \GT,89;
    MoveL offs(piezagrande,0,0,50),v200,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
ENDPROC
PROC monta_perno2()
    MoveJ offs(perno2,0,0,25),v500,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveL offs(perno2,0,0,-112.5),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    SetDO cerrar_pinza,1;
    WaitDI pillada,1;
    MoveL offs(perno2,0,0,25),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
    MoveJ offs(piezagrande,0,0,100),v500,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    MoveJ offs(piezagrande,0, -50, 100),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    MoveL offs(piezagrande,0,-50,-87.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
    SetDO cerrar_pinza,0;
```

```
   WaitAI AnchoPieza, \GT,89;
   MoveL offs(piezagrande,0,0,50),v200,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
ENDPROC
PROC reposo()
   MoveJ pos_reposo,v500,z20,T_PinzaAnalogica2\WObj:=wobj0;
ENDPROC
PROC Desmonta_perno2()
   MoveJ Offs(piezagrande,0,0,50),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
   MoveJ Offs(piezagrande,0,-50,50),v200,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
   MoveL Offs(piezagrande,0,-50,-87.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
   SetDO cerrar_pinza,1;
   WaitDI pillada,1;
   MoveL Offs(piezagrande,0,-50,125),v100,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
   MoveJ Offs(perno2,0,0,125),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
   MoveL Offs(perno2,0,0,-112.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
   SetDO cerrar_pinza,0;
   WaitAI AnchoPieza, \GT,89;
   MoveL Offs(perno2,0,0,25),v200,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
ENDPROC
PROC Desmonta_perno1()
   MoveJ Offs(piezagrande,0,0,50),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
   MoveJ Offs(piezagrande,0,50,50),v200,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
```

```
        MoveL Offs(piezagrande,0,50,-87.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        SetDO cerrar_pinza,1;
        WaitDI pillada,1;
        MoveL Offs(piezagrande,0,50,125),v100,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveJ Offs(perno1,0,0,125),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
        MoveL Offs(perno1,0,0,-112.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
        SetDO cerrar_pinza,0;
        WaitAI AnchoPieza, \GT,89;
        MoveL Offs(perno1,0,0,25),v200,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
    ENDPROC
    PROC Desmonta_pieza2()
        MoveJ Offs(piezagrande,0, 39.38, 0),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveL Offs(piezagrande,0, 39.38, -
112.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;
        SetDO cerrar_pinza,1;
        WaitDI pillada,1;
        MoveL Offs(piezagrande,0, 39.38, 50),v100,z0,T_PinzaAnalogica2\WObj:=Mesa_1;
        MoveJ Offs(piezas,0,0,50),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
        MoveL Offs(piezas,0,0,-162.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
        SetDO cerrar_pinza,0;
        WaitAI AnchoPieza, \GT,89;
        MoveL piezas,v200,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
```

```
ENDPROC

PROC Desmonta_pieza1()

  MoveJ Offs(piezagrande,0, -35.62, 0),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_1;

  MoveL Offs(piezagrande,0, -35.62, -
112.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_1;

  SetDO cerrar_pinza,1;

  WaitDI pillada,1;

  MoveL Offs(piezagrande,0, -35.62, 50),v100,z0,T_PinzaAnalogica2\WObj:=Mesa_1;

  MoveJ Offs(piezas,0,0,50),v500,z0,T_PinzaAnalogica2\WObj:=Mesa_2;

  MoveL Offs(piezas,0,0,-137.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;

  SetDO cerrar_pinza,0;

   WaitAI AnchoPieza, \GT,89;

  MoveL piezas,v200,z0,T_PinzaAnalogica2\WObj:=Mesa_2;

ENDPROC

PROC Desmonta_junta()

  MoveJ piezagrande,v500,z0,T_PinzaAnalogica2\WObj:=Mesa_1;

  MoveL offs(piezagrande,0,0,-137.5),v200,fine,T_PinzaAnalogica2\WObj:=Mesa_1;

  SetDO cerrar_pinza,1;

  WaitDI pillada,1;

  MoveL offs(piezagrande,0,0,75),v100,z0,T_PinzaAnalogica2\WObj:=Mesa_1;

  MoveJ Offs(junta,0,0,75),v500,fine,T_PinzaAnalogica2\WObj:=Mesa_2;

  MoveL offs(junta,0,0,-162.5),v100,fine,T_PinzaAnalogica2\WObj:=Mesa_2;
```

```
        SetDO cerrar_pinza,0;
        WaitAI AnchoPieza, \GT,89;
        MoveL junta,v200,z0,T_PinzaAnalogica2\WObj:=Mesa_2;
    ENDPROC
ENDMODULE
```

Anexo F: Solucion Demotren

```
MODULE Demotren
    !*********************************************************
    !
    ! Módulo:  Demotren
    !
    !
    ! Autor: Pablo Molina Mata
    !
    ! Versión: 1.0
    !
    !*********************************************************
    TASK PERS wobjdata
Sistema_piezas:=[FALSE,TRUE,"",[[600,0,0],[1,0,0,0]],[[0,0,0],[1,0,0,0]]];
    TASK PERS wobjdata Sistema_tren:=[FALSE,TRUE,"",[[-
600,0,0],[0.707106781,0,0,0.707106781]],[[0,0,0],[1,0,0,0]]];
    CONST robtarget Pos_reposo:=[[-39.377,-379.371,478.24],[0.991444861,-
0.130526192,0,0],[-2,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Tren:=[[-250,-200,75],[0.707106781,0,0,-0.707106781],[-2,0,-
2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Cilindros:=[[-350,-262.5,137.5],[1,0,0,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Pieza:=[[-150,-262.5,125],[1,0,0,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
PERS tooldata T_PinzaAnalogica2:=[TRUE,[[0,0,50],[0,0,1,0]],[1,[0,0,30],[1,0,0,0],0,0,0]];
!*********************************************************
!
! Procedimiento Main
!
!   Este es el punto de entrada de su programa
!
!*********************************************************


PROC main()
   ConfL\on;
   SingArea\Wrist;
   SetDO Cierra_Pinza,1;
   SetDO Cierra_Pinza,0;
   reposo;
   MontaPieza;
   MontaCilindro1;
   MontaCilindro2;
   reposo;
   WaitTime 1;
   DesmontaCilindro2;
   DesmontaCilindro1;
```

```
    DesmontaPieza;
    reposo;
ENDPROC
PROC reposo()
    MoveJ Pos_Reposo,v1000,z100,T_PinzaAnalogica2\WObj:=wobj0;
ENDPROC
PROC MontaPieza()
    MoveJ Offs(Pieza,0,0,100),v500,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveL Offs(Pieza,0,0,-50),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    SetDO Cierra_Pinza,1;
    WaitDI Pillada,1;
    MoveL Offs(Pieza,0,0,100),v100,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveJ Offs(Tren,0,0,250),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,0,50),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    SetDO Cierra_pinza,0;
    WaitAI AnchoPieza,\GT,89;
    MoveL Offs(Tren,0,0,250),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
ENDPROC
PROC MontaCilindro1()
    MoveJ Offs(Cilindros,0,0,100),v500,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveL offs(Cilindros,0,0,-50),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    SetDO Cierra_Pinza,1;
```

```
    WaitDI Pillada,1;
    MoveL Offs(Cilindros,0,0,100),V100,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveJ Offs(Tren,0,0,250),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveJ Offs(Tren,0,50,250),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,50,87.5),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    SetDO Cierra_Pinza,0;
    WaitAI AnchoPieza,\GT,89;
    MoveL Offs(Tren,0,50,250),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,0,250),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
ENDPROC
PROC MontaCilindro2()
    MoveJ Offs(Cilindros,0,0,100),v500,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveL Offs(Cilindros,0,0,-75),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    SetDO Cierra_Pinza,1;
    WaitDI Pillada,1;
    MoveL Offs(Cilindros,0,0,100),V100,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveJ Offs(Tren,0,0,250),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveJ Offs(Tren,0,-50,250),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,-50,87.5),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    SetDO Cierra_Pinza,0;
    WaitAI AnchoPieza,\GT,89;
    MoveL Offs(Tren,0,-50,250),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
```

```
    MoveL Offs(Tren,0,0,250),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
ENDPROC
PROC DesmontaCilindro2()
    MoveJ Offs(Tren,0,0,250),v500,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveJ Offs(Tren,0,-50,250),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,-50,87.5),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    SetDO Cierra_Pinza,1;
    WaitDI Pillada, 1;
    MoveL Offs(Tren,0,-50,250),v100,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveJ Offs(Cilindros,0,0,100),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveL Offs(Cilindros,0,0,-75),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    SetDO Cierra_Pinza,0;
    WaitAI AnchoPieza,\GT,89;
    MoveL Offs(Cilindros,0,0,100),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
ENDPROC
PROC DesmontaCilindro1()
    MoveJ Offs(Tren,0,0,250),v500,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveJ Offs(Tren,0,50,250),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,50,87.5),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    SetDO Cierra_Pinza,1;
    WaitDI Pillada, 1;
    MoveL Offs(Tren,0,50,250),v100,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
```

```
    MoveJ Offs(Cilindros,0,0,100),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveL offs(Cilindros,0,0,-50),v100,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    SetDO Cierra_Pinza,0;
    WaitAI AnchoPieza,\GT,89;
    MoveL Offs(Cilindros,0,0,100),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
  ENDPROC
  PROC DesmontaPieza()
    MoveJ Offs(Tren,0,0,250),v500,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveL Offs(Tren,0,0,50),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_tren;
    SetDO Cierra_Pinza,1;
    WaitDI Pillada, 1;
    MoveL Offs(Tren,0,0,250),v100,z0,T_PinzaAnalogica2\WObj:=Sistema_tren;
    MoveJ Offs(Pieza,0,0,100),v200,z0,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    MoveJ Offs(Pieza,0,0,-50),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
    SetDO Cierra_Pinza,0;
    WaitAI AnchoPieza,\GT,89;
    MoveL Offs(Pieza,0,0,100),v200,fine,T_PinzaAnalogica2\WObj:=Sistema_piezas;
  ENDPROC
ENDMODULE
```

Anexo G: Solucion FantaCan

ROBOT1 (Herramienta)

```
MODULE Module1
!***********************************************************
    !
    ! Módulo:  FantaCan1
    !
    ! Descripción:
    !   Movimiento Coordinado de dos Robots, controlados por un único controlador
    !
    ! Autor: Pablo Molina Mata
    !
    ! Versión: 1.0
    !
    !***********************************************************
     PERS tooldata MyTool:=[ TRUE,
[31.792631019,0,229.638935148],[0.945518576,0,0.325568154,0 ]],[1,[0,0,1],[1,0,0,0],0,0,0]];
     PERS wobjdata
MesaLatas1:=[FALSE,FALSE,"ROB_2",[[0,0,0],[1,0,0,0]],[[0,0,25],[1,0,0,0]]];
     CONST robtarget Target_10:=[[100,-25,75],[0,0.5,0.866025404,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
     CONST robtarget Target_20:=[[50,-25,75],[0,0.5,0.866025404,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
CONST robtarget Target_30:=[[50,25,75],[0,0.258819045,0.965925826,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_40:=[[-50,25,75],[0,0.258819045,0.965925826,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_50:=[[-50,-25,75],[0,0.5,0.866025404,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_60:=[[-100,-25,75],[0,0.5,0.866025404,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_70:=[[-100,25,75],[0,0.5,0.866025404,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_80:=[[-50,-75,75],[0,0.5,0.866025404,0],[-1,-
1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_90:=[[50,-75,75],[0,0.258819045,0.965925826,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_100:=[[50,75,75],[0,0,1,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST robtarget Target_110:=[[100,75,75],[0,0,1,0],[-1,0,-
1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

PERS tasks Task_list{2}:=[["T_ROB1"], ["T_ROB2"]];

VAR syncident SyncOn;



!***********************************************************

!

! Procedimiento Main

!
```

```
! Este es el punto de entrada de su programa
!
!*********************************************************
PROC main()
  ConfL\on;
  SingArea\Wrist;
  IF NOT(IsSyncMoveOn()) THEN
  SyncMoveOn SyncOn, Task_list;
  ENDIF
  FOR I FROM 1 TO 50 DO
    Path_10;
  ENDFOR
ENDPROC
  PROC Path_10()
    MoveL Target_10,\ID:=1,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_20,\ID:=2,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_30,\ID:=3,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_40,\ID:=4,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_50,\ID:=5,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_60,\ID:=6,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_70,\ID:=7,v500,z0,MyTool\WObj:=MesaLatas1;
    MoveL Target_40,\ID:=8,v500,z0,MyTool\WObj:=MesaLatas1;
```

```
            MoveL Target_50,\ID:=9,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_80,\ID:=10,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_90,\ID:=11,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_20,\ID:=12,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_30,\ID:=13,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_100,\ID:=14,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_110,\ID:=15,v500,z0,MyTool\WObj:=MesaLatas1;
            MoveL Target_10,\ID:=16,v500,z0,MyTool\WObj:=MesaLatas1;
        ENDPROC
    ENDMODULE
```

ROBOT2 (MesaLatas)

```
    MODULE Module1
        !***********************************************************
        !
        ! Módulo:  Module1
        !
        ! Descripción:
        !   Movimiento coordinado del Robot 2, el robot que sostiene el objeto de trabajo en
esta
        !   Estación
        !
        ! Autor: Pablo Molina Mata
```

```
!
! Versión: 1.0
!
!*********************************************************
PERS tooldata Mesa_Latas:=[TRUE,[[0,0,25],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
PERS wobjdata MesaLatas2:=[FALSE,TRUE,"",[[548.481,-
115.705,297.178],[0.851014511,-0.061555277,-0.282579821,0.438330805]],[[-
34.964,140.479,-52.347],[1,0,0,0]]];
CONST robtarget Mesa1:=[[-46.642,-
180.621,30.694],[0.987725131,0.138052069,0.056875534,0.045889709],[0,-1,-
2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Mesa2:=[[-42.894536631,-
228.504576085,16.797250035],[0.987725131,0.138052069,0.056875534,0.045889709],[0,-1,-
2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Mesa3:=[[-50.389463369,-
132.737423915,44.590749965],[0.987725131,0.138052069,0.056875534,0.045889709],[0,-1,-
2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Mesa4:=[[23.809432867,-
124.760693318,37.114416215],[0.987725131,0.138052069,0.056875534,0.045889709],[0,-1,-
2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Mesa5:=[[27.556896236,-
172.644269403,23.21766625],[0.987725131,0.138052069,0.056875534,0.045889709],[0,-1,-
2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget Mesa6:=[[31.304359605,-
220.527845488,9.320916285],[0.987725131,0.138052069,0.056875534,0.045889709],[0,-1,-
2,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
PERS tasks Task_list{2}:=[["T_ROB1"], ["T_ROB2"]];
```

```
VAR syncident SyncOn;


!***********************************************************

!

! Procedimiento Main

!

!   Este es el punto de entrada de su programa

!

!***********************************************************

PROC main()

   ConfL\Off;

   SingArea\Wrist;

   IF NOT(IsSyncMoveOn()) THEN

   SyncMoveOn SyncOn, Task_list;

   ENDIF

   FOR I FROM 1 TO 50 DO

      Path_10;

   ENDFOR

ENDPROC

  PROC Path_10()

     MoveJ Mesa1,\ID:=1,v500,z0,Mesa_Latas\WObj:=MesaLatas2;

     MoveJ Mesa2,\ID:=2,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
```

```
        MoveJ Mesa3,\ID:=3,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa4,\ID:=4,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa5,\ID:=5,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa6,\ID:=6,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa1,\ID:=7,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa2,\ID:=8,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa3,\ID:=9,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa4,\ID:=10,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa5,\ID:=11,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa6,\ID:=12,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa1,\ID:=13,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa2,\ID:=14,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa3,\ID:=15,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
        MoveJ Mesa4,\ID:=16,v500,z0,Mesa_Latas\WObj:=MesaLatas2;
    ENDPROC
ENDMODULE
```

Bibliografía

[1] ABB-ROBOTICS. Abb robot-studio (http://new.abb.com/products/robotics/robotstudio).

[2] ABB-ROBOTICS. *Manual de referencia técnica. Descripción general de RAPID.*

[3] ABB-ROBOTICS. *Manual de referencia técnica. Instrucciones, funciones y tipos de datos de RAPID.*

[4] ABB-ROBOTICS. *Manual del operador. IRC5 con FlexPendant.*

[5] ABB-ROBOTICS. *Manual del operador. RobotStudio.*

[6] ABB-ROBOTICS. *Especificaciones del producto. IRB 120.*

[6] ABB-ROBOTICS. *Especificaciones del producto. IRB 120.*

[7] ISA Ingeniería de Sistemas y Automática –UMH. *Control de Robots y Sistemas Sensoriales. Ejemplo de Programación en RAPID.* *(http://isa.umh.es/asignaturas/rvc/ejemplo%20rapid.pdf)*

[8] Agustín Ramos Hurtado. *Trabajo de Fin de Grado. Diseño, programación y simulación de estaciones robotizadas industriales con Robotstudio. Universidad de Sevilla, 2016.*

[9] J.J. Guerrero. *Apuntes Sistemas de Automatización Flexible y Robótica, Tema 7, Programación.*

[10] Jose Ignacio Abad Martínez. *Trabajo de Fin de Grado. Integración de los componentes y sistemas de percepción de una célula de fabricación robotizada. Universidad de Zaragoza, 2014.*