Escuela Politécnica Superior de Linares

Grado en Ingeniería Industrial

**UNIVERSIDAD DE JAÉN**
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

# ESTUDIO DE SOFTWARE DE VIRTUALIZACIÓN DE SISTEMAS ROBÓTICOS

**Alumno:** **José Antonio Moreno Rascón**

**Tutor:** Prof. Dr. D. Luis Felipe Sesé
**Tutor:** Prof. D. Cesar Alvarez Arroyo
**Depto.:** Ingeniería Mecánica y Minera

**Septiembre, 2017**

# ***Índice.***

# ***Índice de Figuras.***

# ***Índice de Tablas.***

# 1. Resumen.

Este trabajo consiste en la virtualización de la célula robótica adquirida recientemente que se encuentra en los laboratorios de la Escuela Politécnica de Linares. Para ello se ha tenido que hacer uso del software de programación virtual RobotStudio, también de tener que crear modelos 3D con AutoCAD de los elementos que forman parte de la célula.

Por otra parte, además de recrear las prácticas ya realizadas otros años en este entorno virtual, se ha creado una nueva aplicación que use elementos de los que no se disponen en el laboratorio.

Este trabajo permitirá realizar pruebas con el brazo robot sin necesidad de estar presente en el laboratorio, lo cual sería muy útil desde un punto de vista docente.

# 1. Abstract.

This project consists in create a virtual robotic cell, based in the one that exist in the laboratories of the “School of engineering”. In order to create this virtual cell, the elements which form it had to be modelled using the program known as AutoCAD.

Not only the practise done in previous years are virtualised, there has been new application created, using elements that are not present in the laboratories.

This project will make possible to do tests without the need to be in the laboratories, something very interesting from a docent point of view.

## 2. Antecedentes.

Existen ciertas dificultades a la hora de establecer una definición formal de lo que es un robot industrial. La primera de ellas surge de la diferencia conceptual entre el mercado japonés y el euro-americano de lo que es un robot y lo que es un manipulador.

Así, mientras que para los japoneses un robot industrial es cualquier dispositivo mecánico dotado de articulaciones móviles destinado a la manipulación, el mercado occidental es más restrictivo, exigiendo una mayor complejidad, sobre todo en lo relativo al control.

En segundo lugar, y centrándose ya en el concepto occidental, aunque existe una idea común acerca de lo que es un robot industrial, no es fácil ponerse de acuerdo a la hora de establecer una definición formal. Además, la evolución de la robótica ha ido obligando a diferentes actualizaciones de su definición. Actualmente la Organización Internacional de Estándares (ISO) define al robot industrial como:

"*Manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales según trayectorias variables programadas para realizar tareas diversas* "[1].

El robot industrial más antiguo conocido [2], conforme a esta definición, se completó con Bill Griffith P. Taylor [3] en 1937 en Inglaterra y fue publicado en la revista Meccano, en marzo de 1938 (Figura 2.1 a). La grúa, como se le denominó al dispositivo, aunque también se encuentra referenciado como Gargantua fue construido casi en su totalidad con piezas Meccano y accionado por un único motor eléctrico.

Cinco ejes de movimiento son posibles, entre los cuales se incluyen los de la pinza y rotación de la misma.

La grúa está dirigida por un único motor, el cual acciona unos engranajes, los cuales generan los distintos movimientos de la grúa. Para seleccionar un grupo de engranajes se debe accionar una de las 5 palancas de las que dispone la grúa.

La automatización se logró mediante el uso de cinta de papel perforado, los agujeros de este papel regulan la activación de unos solenoides, estos a su vez activan las palancas que dirigen los movimientos.

Por tanto, la programación en el papel perforado se realiza mediante un seguimiento de las revoluciones del motor que alimenta el sistema, por ejemplo, abrir la pinza requiere

150 revoluciones, esta programación permite al robot apilar bloques de madera en los patrones pre-programados. Chris Shute construyó una réplica completa del robot en 1997, mostrada en la Figura 2.1 b, [2].

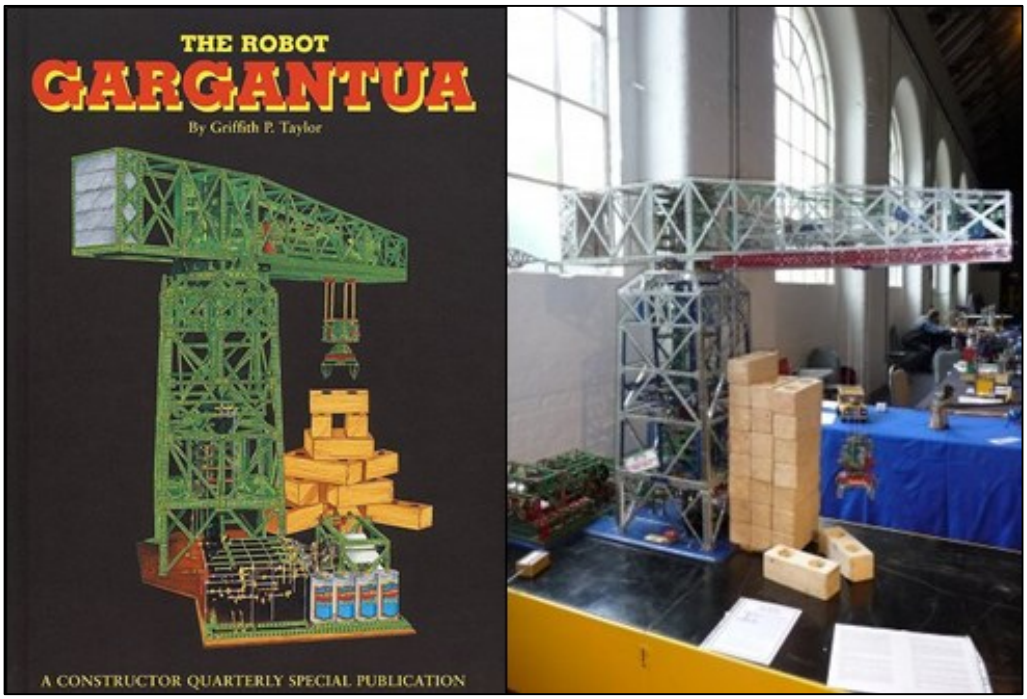


Figura 2.1 a) Portada Meccano 1937. b) Réplica de Robot Gargantua [3].

Si bien este dispositivo era muy innovador para la época y como se ha comentado es el primer robot conocido que entra dentro de la definición de robot industrial, no tenía las capacidades necesarias para llevar acabo ningún trabajo industrial.

La programación era lenta y minuciosa, y tenía tendencia a fallos de aproximación, razón por la que habría que esperar varias décadas para que la robótica y la industria se encontraran, este paso fue dado por George Devol y Joseph F. Engelberger (Figura 2.2) [4] que en 1956 fundaron Unimation, la primera empresa de USA en fabricar robots para uso industrial, su robot fue bautizado Unimate, visible en la Figura 2.2.

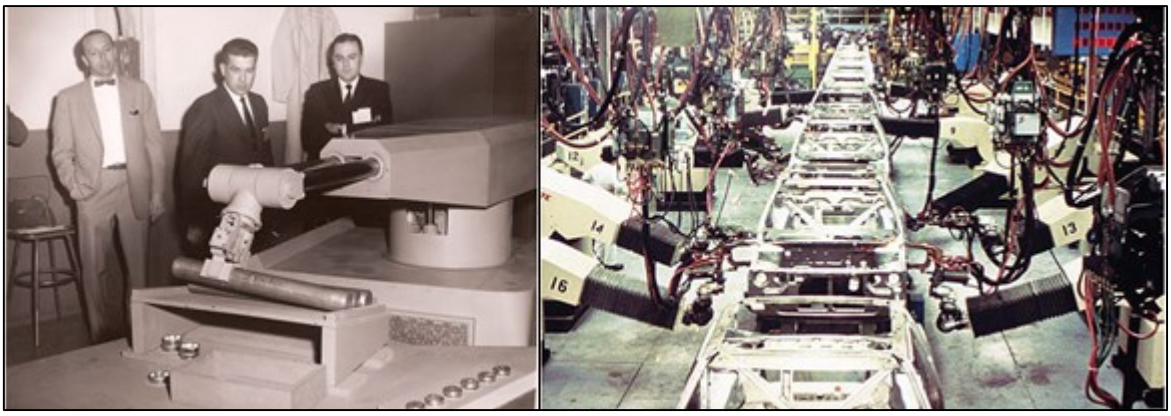

Figura 2.2 a) Unimate y sus creadores en una presentación. b) Unimate en una línea de montaje.

Estos primeros robots debían ser programados con cautela y puestos a prueba en una línea de montaje (Figura 2.2), para evitar fallos en la cadena de producción, lo cual implica asumir unas grandes pérdidas económicas, menores que las que supone una cadena de montaje defectuosa por supuesto, aun así, con el tiempo se decidió que se debían evitar estas pérdidas también, lo que hizo con el tiempo surgir los programas de simulación. [5].

Estos programas informáticos permitirían probar las líneas de montaje sin necesidad de recurrir a la misma, el uso de software específico para la simulación de brazos y productos se lleva a cabo por marcas como Siemens, ABB, etc. [6]

Si bien los principales programas usados son privados existen opciones gratuitas, como v-rep, gazebo (este programa se centra en mayor medida en las inteligencias artificiales), etc [7], [8].

El software del que se dispone es RobotStudio, una herramienta proporcionada por el propio grupo ABB [9], y con un enorme valor añadido, pues con en él se pueden crear ambientes virtuales que simulen los reales (Figura 2.3). El lenguaje informático que usan los robots ABB es el conocido como RAPID (Robotics Application Programming Interactive Dialogue).

En RobotStudio se puede programar offline aplicaciones en RAPID y comprobar si la aplicación tiene algún problema mediante las herramientas de simulación para luego volcar esta aplicación al robot que se desee, haciendo que sea la herramienta principal de programación usada a la hora de poner en marchas un proceso industrial.

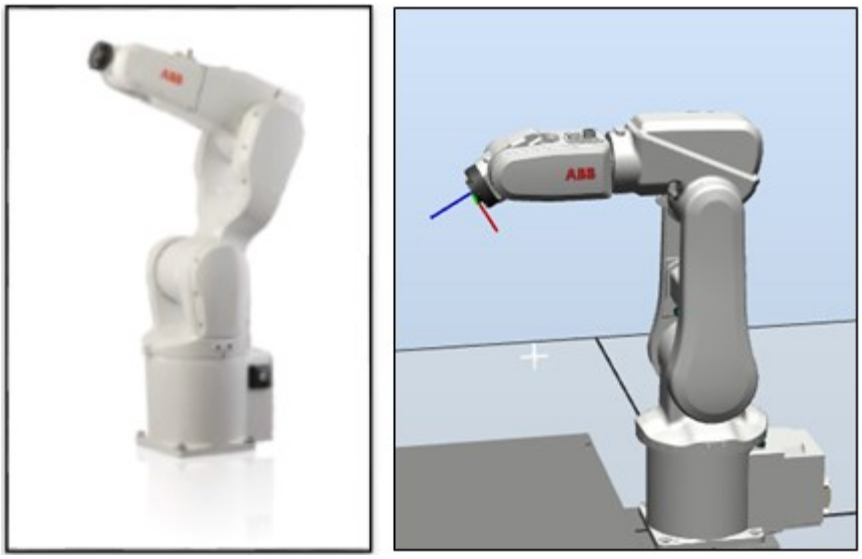

Figura 2.3 a) Robot real IRB 120 b) Modelo simulado del IRB 120.

El grupo ABB es además consciente de los usos de elementos de otras posibles marcas de componentes electrónicos (como pueden ser herramientas pinza, ventosa, etc.) y del uso de este software para un uso didáctico. Por estas razones el software permite la creación de herramientas virtuales y ha puesto en marcha una serie de video tutoriales es su página web oficial [9], haciendo a este software muy versátil, si bien el traslado total de la programación off-line a la real con el uso de otras marcas tiene una dificultad añadida.

Además, el programa RobotStudio permite al usuario simular el control manual conocido como FlexPendant Figura 2.4.

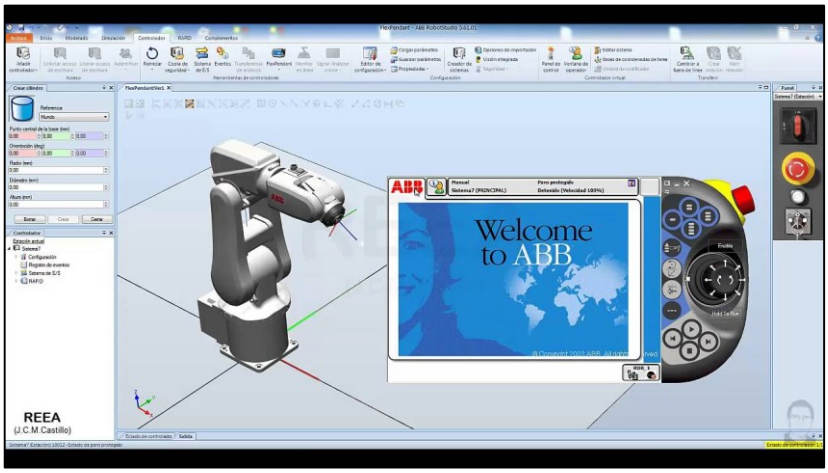


Figura 2.4 Control manual FlexPendant simulado.

# 3. Objetivo y motivación.

En este apartado se expondrá el objeto de realizar este trabajo su motivo y los objetivos a cumplir.

## 3.1 Objeto

La titulación del Grado en Ingeniería Mecánica, al igual que el resto de titulaciones impartidas en la Escuela Politécnica Superior de Linares, debe concluir según la normativa del plan vigente con la elaboración y la defensa por parte del estudiante de un Trabajo Fin de Grado. Este trabajo debe realizarse en la fase final del plan de estudios y debe estar orientado a la evaluación de competencias asociadas al título.

## 3.2 Motivación

En la industria de hoy en día la robótica están ocupando un puesto cada vez más indispensable, este auge no ha pasado desapercibido para las empresas, de manera que el futuro ingeniero deberá familiarizarse con este nuevo medio.

El primer paso que ya ha sido dado por la Escuela Politécnica Superior de Linares es la incorporación de un brazo robótico industrial en sus laboratorios, el siguiente es la planificación de sesiones didácticas que expliquen su uso a los alumnos.

En anteriores años esto se llevó a cabo con sesiones extraordinarias en las que el alumno observaba al robot hacer determinadas funciones e incluso programar ellos mismo movimientos para el brazo, teniendo ciertas limitaciones, no solo por tiempo sino por el número de alumnos.

Este proyecto solventará este límite al crear estas mismas prácticas y algunas nuevas en un espacio virtual, lo que permitirá al alumno manejar el robot de manera independiente, además de no depender tanto de los horarios del laboratorio, dando más libertad al administrar el tiempo de uso del laboratorio.

### 3.3 Objetivos

Por lo tanto, en este trabajo se pretenden alcanzar los siguientes objetivos:

- Crear una estación virtual con todos los elementos disponibles en el laboratorio.
- Crear una guía de usuario del software RobotStudio, que permita al alumno simular las prácticas hechas en años anteriores.
- Crear aplicaciones que no estén sometidas a la falta temporal de sensores en el laboratorio o que incluyan objetos de trabajo (Cajas, cilindros, etc.) sin necesidad de tener que fabricarlos físicamente.
- Realizar un análisis del software disponible
- Realizar un análisis de los resultados y propuesta de trabajos futuros.

# 4. Revisión bibliográfica.

En este apartado se pretende realizar una revisión bibliográfica sobre los métodos de programación y simulación, de esta forma se establecerán las bases sobre las que se desarrollará este trabajo final de grado [10], [11] y [12].

Los programas de simulación en robótica industrial tienen por lo general dos formas de programación con un uso indiscriminado de ambos, el lenguaje de programación usado por el software RobotStudio es una versión de RAPID, otros lenguajes usados son KAREL (Basado en Pascal y usado por los robots de FANUC) y KRL (Basado en el estándar IRL, usado por los robots de KUKA) [13], [14].

En la Tabla 4-1 podemos observar varios de los lenguajes de programación de robots, algunos ya en desuso.

| **Lenguaje** | **Universidad Fabricante** | **Origen y Características** | **Aplicaciones** |
|---|---|---|---|
| Wave (1973) | Stanford | - Nivel actuador<br>- Compilador asociado a Robot.<br>- Sintaxis simple: MOVE SEARCH.<br>- Objetos tratados: constantes, variables, vectores, contadores. | - Manipulación de piezas mecánicas.<br>- Montaje de bomba de agua.<br>- Ensamblaje. |
| AL (Assembly Language) (1974) | Stanford | - Basado en PASCAL.<br>- Nivel actuador.<br>- Transformador de coordenadas. | -Manipulación de piezas mecánicas. |

| | | | |
|---|---|---|---|
| | | - Sintaxis compleja: MOVE TO, MOVE VIA, etc.<br>- Objetos tratados: los de Wave más planos traslaciones, rotaciones.<br>-Estructura Algol. | -Sistema de desarrollo de programas Pointy.<br>- Robots Unimation. |
| RAPT (1978) | Sistemas de Toulouse (Universidad de Edimburgo) | - Interprete escrito en APT.<br>- Nivel objeto. | - Montaje. |
| VAL I (1979)<br>VAL II (1983) | Unimation | - Se partió de AL, pero utilizando BASIC.<br>- Nivel actuador.<br>- Transformador de coordenadas.<br>- Instrucciones de movimiento y de control (MOVE-, DEPART-).<br>- Instrucciones aritméticas.<br>- Estructura Algol. | - Montaje. |
| MCL | Cincinnati Milacron | - Desarrollo por ICAM y realizado en Fortram | |
| AML (1979)<br>FUNKY<br>MAPLE<br>AUTOPASS (1977) | IBM | - Alto nivel interactivo estructurado.<br>- Nivel objeto.<br>- Compilador al nivel.<br>- Sintaxis evolucionada, como colocar X sobre Y alineando Z y T. | - Corresponde al nivel de descripción de las gamas de montaje actuales. |
| LRP | ACMA | - Desarrollado en colaboración con la Universidad de Montpellier. | |
| V+ (1989) | ADEPT | - Lenguaje textual de alto nivel.<br>- Ejecución de varios programas al mismo tiempo (Multitarea).<br>- Proceso asíncrono o ejecución de rutinas de reacción ante determinados eventos. | - Usado en Adept y Staübli. |
| RAPID (1994) | ABB | - Lenguaje textual de alto nivel altamente estructurado.<br>- Estructura modular de programa.<br>- Uso de estructuras predefinidas para especificar la configuración del robot y las características de la herramienta. | |

| | | | |
|---|---|---|---|
| KAREL | FANUC | - Basado en PASCAL.<br>- Soporta multitarea.<br>- Estructura de datos asociados modificable. | |
| KRL | KUKA | - Basado en el estándar IRL, descrito en la norma DIN 66312, lo que aporta transportabilidad. | |

Tabla 4-1 Lenguajes de programación usados en robótica, de mayor a menor antigüedad.

Al programar un robot industrial se usan principalmente dos métodos de programación [12], [15], la programación gestual o directa y la programación textual explicita, a continuación, se desarrollan en mayor profundidad.

1) Programación gestual o directa.

En este tipo de programación, el propio brazo interviene en el trazado del camino y en las acciones a desarrollar en la tarea de la aplicación. Esta característica determina, inexcusablemente, la programación "on-line".

La programación gestual se subdivide en dos clases:

a) Programación por aprendizaje directo:

En el aprendizaje directo, el punto final del brazo se traslada con ayuda de un dispositivo especial colocado en su muñeca, o utilizando un brazo maestro o maniquí, sobre el que se efectúan los desplazamientos que, tras ser memorizados, serán repetidos por el manipulador.

La programación por aprendizaje directo tiene pocas posibilidades de edición, ya que, para generar una trayectoria continua, es preciso almacenar o definir una gran cantidad de puntos, cuya reducción origina discontinuidades. El "software" se organiza, aquí, en forma de intérprete.

b) Programación mediante un dispositivo de enseñanza.

La programación, usando un dispositivo de enseñanza, consiste en determinar las acciones y movimientos del brazo manipulador, a través de un elemento especial para este cometido. En este caso, las operaciones ordenadas se sincronizan para conformar el programa de trabajo, el dispositivo de enseñanza del que se dispone con el robot IRB 120 se conoce como FlexPendant Figura 4.1.

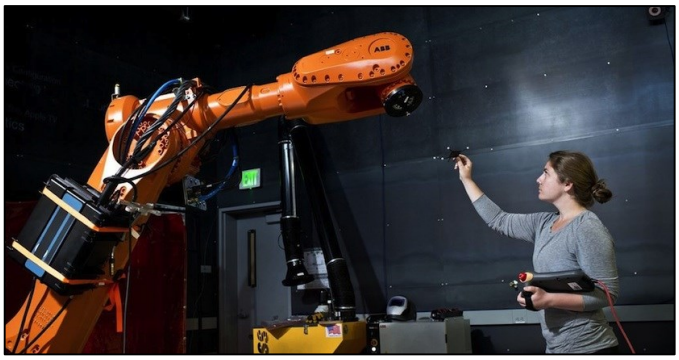

Figura 4.1 Robot de la marca ABB con su técnico usando un FlexPendant.

Los lenguajes de programación gestual, además de necesitar al propio robot en la confección del programa, carecen de adaptabilidad en tiempo real con el entorno y tratan las situaciones de emergencia de manera podo efectiva.

2) Programación textual explicita

El programa queda constituido por un texto de instrucciones o sentencias [12], cuya confección no requiere de la intervención del robot; es decir, se efectúan "off-line". Con este tipo de programación, las acciones del brazo manipulado, no se realizan, sino que se calculan y simulan en el programa.

Según las características del lenguaje, pueden confeccionarse programas de trabajo complejos, con inclusión de saltos condicionales, empleo de bases de datos, posibilidad de creación de módulos operativos intercambiables, capacidad de adaptación a las condiciones del mundo exterior, etc.

Existen tres niveles de programación textual:

a) Nivel robot:

Las órdenes se refieren a los movimientos a realizar por el robot. Es necesario especificar cada uno de los movimientos y todas las variables de interés del robot, como velocidad que requiere el elemento terminal, precisión, activación de señales para herramientas como pinzas, soldadores, etc. Este nivel se encuentra completamente implementado en los robots industriales del mercado.

b) Nivel objeto:

Las órdenes se refieren al estado en que deben quedar los objetos a mover o con los que se interactúa, de manera que un planificador de tareas se encarga de consultar una base de datos y generar instrucciones a nivel robot.

Este nivel se encuentra en pleno desarrollo y algunos fabricantes ya lo implementan parcialmente, un ejemplo en RobotStudio de este nivel es la capacidad de generar una trayectoria automática que siga el contorno de un elemento, una orden útil en soldadura o controles de calidad con visión artificial.

c) Nivel tarea:

Las órdenes se refieren al objetivo a conseguir. El programa se reduce a sentencias globales en las que se indica qué debe conseguir el robot, en lugar de instrucciones de cómo llevar a cabo la tarea. Se encuentra en una fase primigenia en la que algunas empresas proveen de robots capaces de hacer una tarea de manera eficaz con una mayor cantidad de variables en un estado indefinido [12], [17].

RightHand Robotics es una empresa cuyo producto es un robot, visible en la Figura 4.2 que se adentra en el nivel tarea, siendo capaz de tomar objetos sin necesidad de conocer especificaciones previas aprendiendo con el tiempo como tomar los objetos de manera cada vez más eficaz.

Figura 4.2 Brazo robótico perteneciente a RightHand Robotics [17].

# 5. Materiales y medios informáticos disponibles.

En este apartado de darán a conocer los materiales usados, que son: RobotStudio, el brazo robótico IRB 120 y varios accesorios usados en las prácticas.

## 5.1 Brazo Robot IRB 120.

El robot industrial del que dispone el campus tecnológico de Linares es un brazo robot modelo IRB 120 (Figura 5.1 a), es el modelo más pequeño fabricado por el grupo ABB [9], lo que lo hace perfecto para pequeña industria (paletización, soldadura de pequeñas zonas, selección de elementos en una cinta, etc.) o usos didácticos y de investigación como es el caso del campus.

El robot disponible en el campus está unido a una mesa de trabajo que proporciona un área de trabajo amplia y regular que además ha sido modificada para que incluya una cuadrícula que ayuda a la hora de seleccionar puntos en programación gestual (Figura 5.1 b).


Figura 5.1 a) Comparativa de IRB 120 con adulto medio. b) IRB 120 disponible con su mesa reglada.

- Especificaciones técnicas

Las especificaciones técnicas de este modelo son suministradas por el grupo ABB [18], la información suministrada es: su capacidad de carga (3Kg), el área de trabajo viable (Figura 5.2) y la movilidad de los ejes de manera independiente entre otras.

El software de RobotStudio tiene implementadas estas especificaciones técnicas en sus modelos disponibles, lo que permite al software detectar si alguno de los parámetros límites del robot han sido sobrepasados, ejemplo de ello es la herramienta de alcanzabilidad [11]. Esta herramienta determina si el extremo del robot puede trabajar en ese punto.

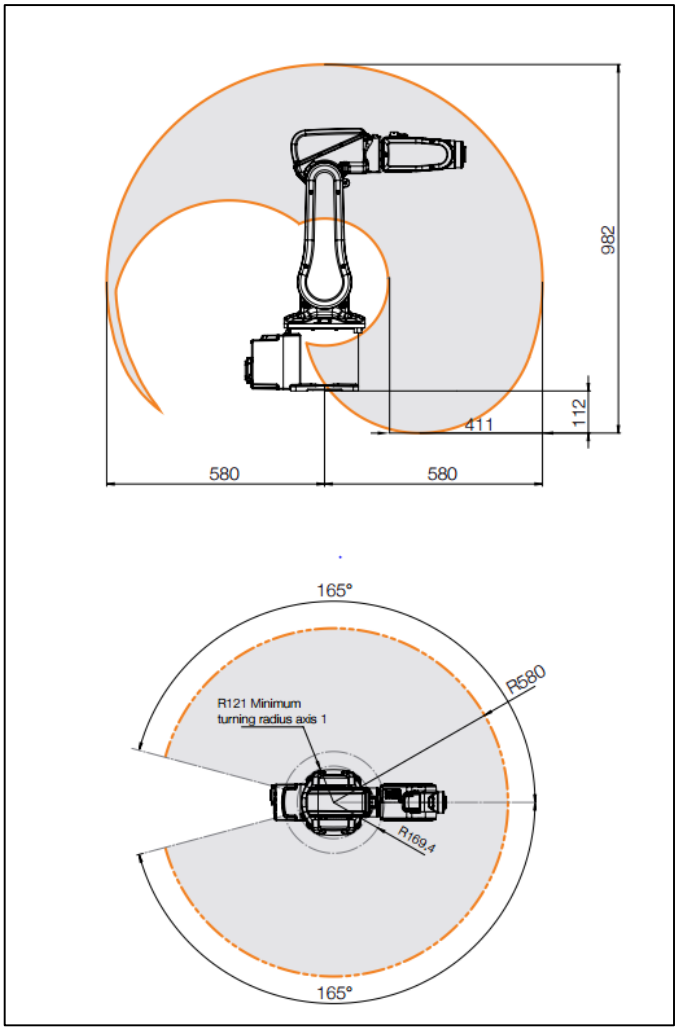


Figura 5.2 Rango de movimiento del IRB 120.

## 5.2 Programa de simulación: RobotStudio.

El uso de RobotStudio en la industria es esencial en los pasos iniciales de puesta en marcha de una célula robótica virtual, permitiendo comprobar la eficacia y viabilidad de las configuraciones y diseños propuestos antes de realizar cualquier montaje, este uso se extiende a la mejora de células ya existentes.

No solo permite crear y mejorar células robóticas, sino que permite determinar si es rentable el cambio de un equipo a uno de capacidades superiores y permite en el ámbito educativo enseñar cómo se debe plantear la solución de un problema industrial, permitiendo una mejor preparación de profesionales en su futuro en la industria.

El programa disponible de RobotStudio con el que se realizó este trabajo es el 6.04 [10], al ser un trabajo que hace una representación 3D a tiempo real de la estación tiene unos requisitos mínimos para su uso, mostrados en la Figura 5.3:

| Item | Requirement |
|---|---|
| CPU | 2.0 GHz or faster processor, multiple cores recommended |
| Memory | 3 GB if running Windows 32-bit<br>8 GB or more if running Windows 64-bit (recommended) |
| Disk | 10+ GB free space, solid state drive (SSD) |
| Graphics card[1] | High-performance, DirectX 11 compatible, gaming graphics card from any of the leading vendors. For the Advanced lightning mode Direct3D feature level 10_1 or higher is required. |
| Screen resolution | 1920 x 1080 pixels or higher is recommended |
| DPI | Normal size (100% / 96 dpi) up to Large size (150% / 144 dpi)<br>Only Normal size supported for Integrated Vision. |
| Mouse | Three-button mouse |
| 3D Mouse [optional] | Any 3D mouse from 3DConnexion, see http://www.3dconnexion.com. |

Figura 5.3 Requisitos de un sistema informático para el uso de RobotStudio.

Esta versión del software hizo mejoras sustanciales como la inclusión de distintas configuraciones de robots y nuevos brazos, además de una actualización en las capacidades de crear trazos automáticamente, se añadieron también más herramientas que permitan la recuperación de datos en caso de que se cerrara el programa bruscamente.

Una nueva capacidad añadida al programa en esta versión es la capacidad de usar cascos de realidad virtual en programación off-line, como el visible en la Figura 5.4, aunque es todavía una característica en desarrollo.


Figura 5.4 Casco de realidad virtual compatible con RobotStudio.

## 5.3 Herramienta Manipuladora, pinza.

La herramienta disponible en el laboratorio para manipular objetos con el brazo IRB 120 es la pinza electrónica de la marca SMC, modelo LEHZ25K2-14-R16P1. Un esquema de la herramienta es visible en la Figura 5.5 y una visualización 3D en la Figura 5.6.

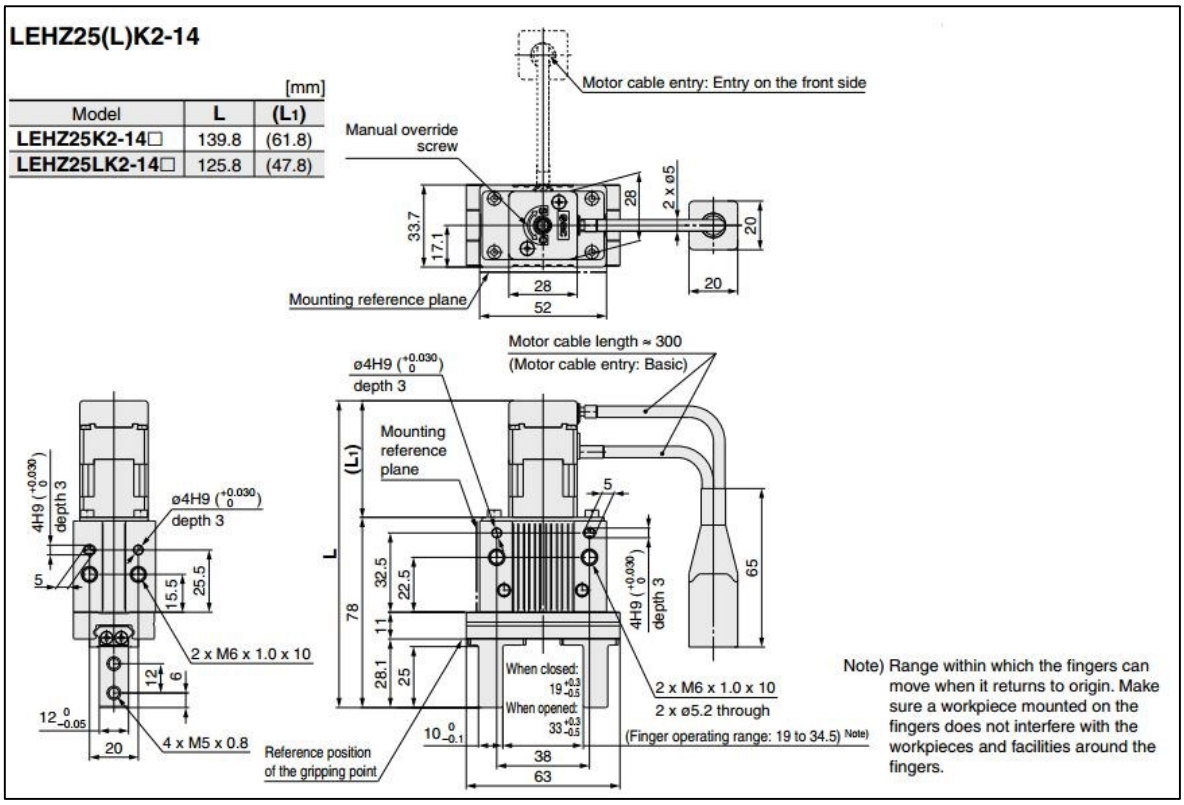


Figura 5.5 Esquema herramienta pinza LEHZ25K2-14-R16P1.

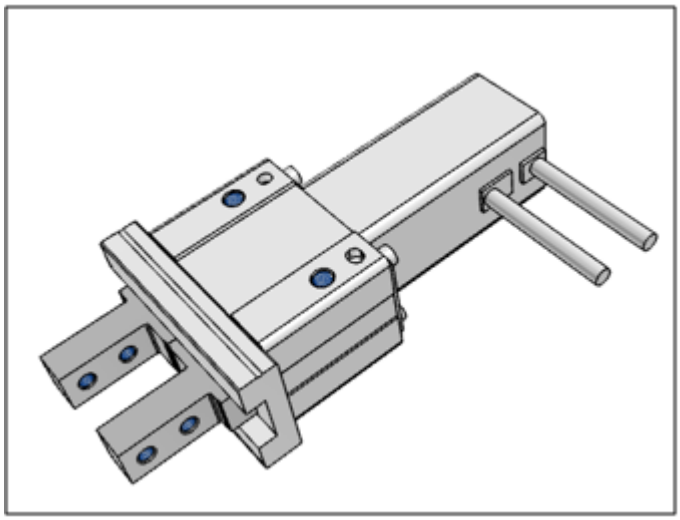

Figura 5.6 Modelo 3D de la herramienta pinza, SMC.

Las especificaciones de este modelo son conocidas, siendo el más importante a la hora de crear la herramienta en el software es su carrera de 14 mm, las principales especificaciones se muestran en la siguiente tabla (Tabla 5-1).

| | |
|---|---|
| **Tamaño del cuerpo** | 25 |
| **Modelo de motor** | Estándar |
| **Carrera** | 14 mm |
| **Opciones de dedo** | Modelo básico. |
| **Entrada del cable del motor** | Básico, entrada en el lado izquierdo |
| **Tipo de cable del actuador** | R [Cable robótica (cable flexible)] |
| **Tipo de controlador** | 6P (Con controlador PNP) |
| **Longitud del cable E/S** | 1 (1.5 m) |
| **Montaje del controlador** | Montaggio con viti |

Tabla 5-1 Datos de herramienta pinza, LEHZ25K2-14-R16P1.

Estos datos se han obtenido a partir del catálogo del producto [19].

# 6. RobotStudio, célula virtual.

A continuación, se ha realizado una serie de metodologías para poder crear la célula virtual, empezando con los componentes individuales y después como aplicarlos correctamente para crear con ellos una estación funcional.

Toda la información necesaria para conocer el uso del software RobotStudio se encuentra en el Manual de usuario [11], sin embargo, también se ha encontrado muy útil varios videos tutoriales [16].

En este trabajo las prácticas que se van a simular son cinco, en todas ellas hay objetos que el brazo debe manipular con el objetivo de reubicarlos de posición y/o apilarlos. Estos objetos son:

- Mesa: Se trata de la base del brazo y el plano de trabajo en el que se realizan las simulaciones.
- Sección de viga: Como su nombre indica se trata de una sección de viga de aluminio.
- Dispensador: Se trata de un elemento impreso en 3D, en el que se apilarán unos cilindros que ruedan sobre uno de sus planos inclinados.
- Cilindro: Se trata de un tubo, este se apila en el objeto conocido como Dispensador.
- Pinza: Es el elemento manipulador imprescindible para la realización de cualquier práctica, es un modelo exacto al disponible en el laboratorio.
- Lápiz: El Lápiz es un objeto que simula el conjunto de un rotulador y de los accesorios necesarios para que la pinza sea capaz de manipularlo.
- Cajas: Las cajas son unos paralepípedos de distintas dimensiones.
- Accesorio: Es un paralepípedo en el que se han realizado unos orificios y una muesca, simula el elemento que une la pinza con el brazo en el laboratorio.

## 6.1 Guía de modelado/creación.

Como se ha comentado anteriormente este apartado se ocupa de la preparación de los elementos necesarios para crear las estaciones, sean herramientas, mesas o componentes inteligentes.

### 6.1.1 Elementos sólidos:

En este apartado se dará una breve guía de los modelos tridimensionales que se han debido realizar, dando detalles de cada uno de ellos. Los planos usados para la realización de estos modelos se encuentran en un trabajo anterior [20].

#### 6.1.1.1 Mesa

La mesa representa el banco de trabajo en el que el brazo robótico se encuentra anclado en el laboratorio, es por tanto no solo la base en la que se encuentra el robot, sino que a su vez es la zona de trabajo del mismo.

Se han realizado diversas medidas en el banco de trabajo, gracias a ellas se ha podido modelar esta mesa, en la que se ha determinado que la diferencia de alturas entre la base en la que se sitúa el brazo y el plano de trabajo es de 2.3 cm (Figura 6.1, a).

La distancia del plano de trabajo con el suelo no es relevante a la hora de realizar las simulaciones, el origen de coordenadas seleccionado es la base en la que está anclado el robot.

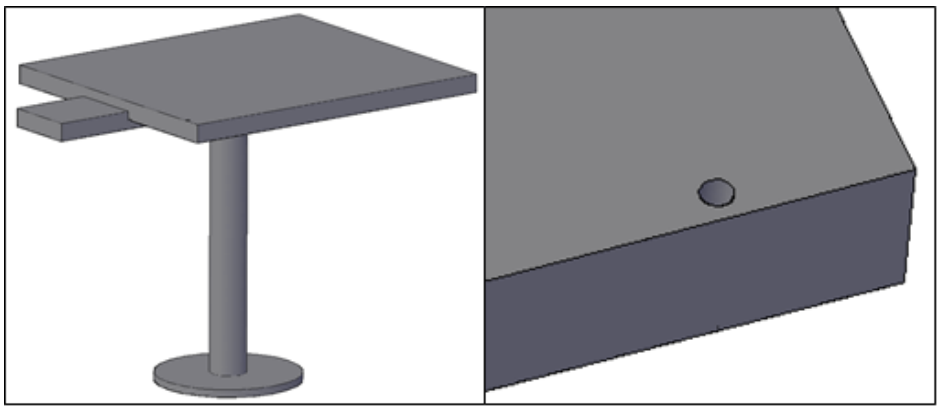

Figura 6.1 a) Modelo CAD de la mesa de trabajo. b) Marcador de centro de coordenadas (Fuente: [20]).

Teniendo en cuenta estos datos se ha modelado con el Software AutoCAD 3D un elemento tridimensional que cumpla estas características. Se ha tenido que crear también un pequeño orificio en la mesa (Figura 6.1, b), cuyo centro es el punto, este punto se usará como origen de coordenadas del plano de trabajo.

RobotStudio tiene herramientas de selección de puntos y "Seleccionar Centro" es una de ellas, permitiendo así localizar fácilmente el punto deseado.

En versiones anteriores RobotStudio permitía volcar archivos con los formatos estándares de AutoCAD como elementos 3D, pero eso ha cambiado a partir de la versión 6.04. En caso de que se deba usar una versión 6.04 [10] lo que se debe hacer es guardar el archivo desde AutoCAD con formato SAT, usando el comando ACISOUT, que permite seleccionar un modelo 3D y guardarlo con este formato.

6.1.1.2 Sección de Viga.

Esta sección de viga será necesaria para la simulación de varias prácticas, será un objeto manipulado por el brazo robot.

De la viga se disponían de planos realizados en años anteriores, sin embargo, el elemento no estaba completamente definido Figura 6.2 a, por lo que se tuvo que realizar medidas de la profundidad de la viga, siendo esta de 80 milímetros.

Asimismo, el modelo realizado Figura 6.2 b no es exacto en su totalidad, solo siéndolo en las medidas necesarias para una simulación satisfactoria (Una que pueda trasladarse a la realidad).

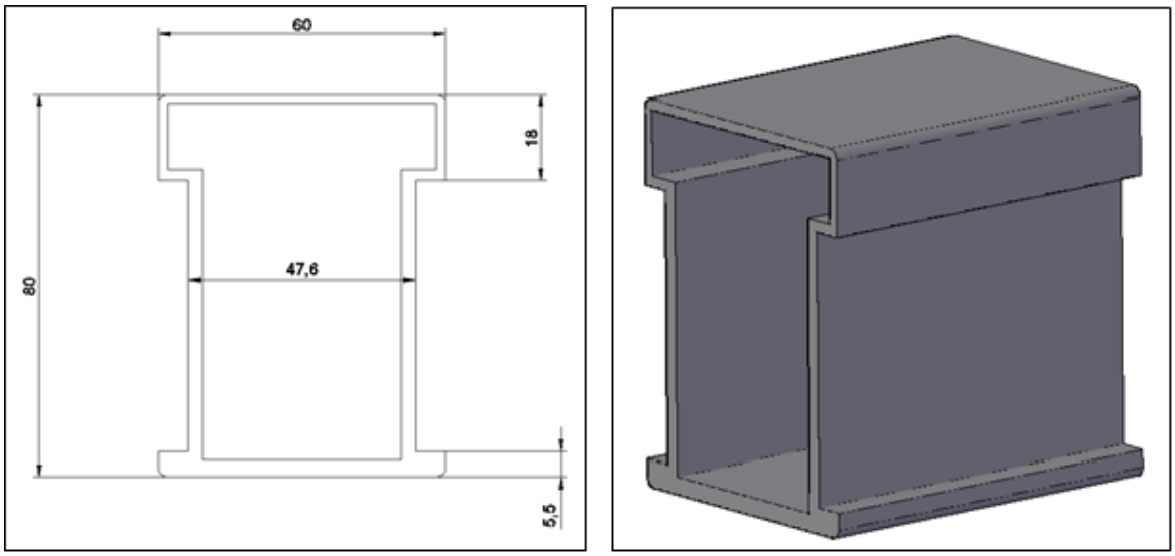


Figura 6.2 a) Plano de las medidas disponibles. b) Modelo CAD de la viga (Fuente: [20]).

La unidad de las medidas de la Figura 6.2 a es milímetros.

6.1.1.3 Dispensador

El dispensador es un objeto necesario para la simulación de una de las prácticas, sin embargo, no es un elemento que interactúe con el brazo robot, este objeto sirve de como base para unos cilindros, que al ser depositados en la parte superior del dispensador rodarán hasta la parte baja del dispensador.

El dispensador (Figura 6.3) es como el caso de la viga un elemento del que se disponía de planos anteriores, pero no estaba en su totalidad definido. Esta falta de completa definición no fue un problema ya que las medidas de las que se disponían eran las necesarias para crear un modelo funcional en la simulación.

Este elemento junto a los Cilindros conforma los elementos necesarios para simular la Práctica nº3.

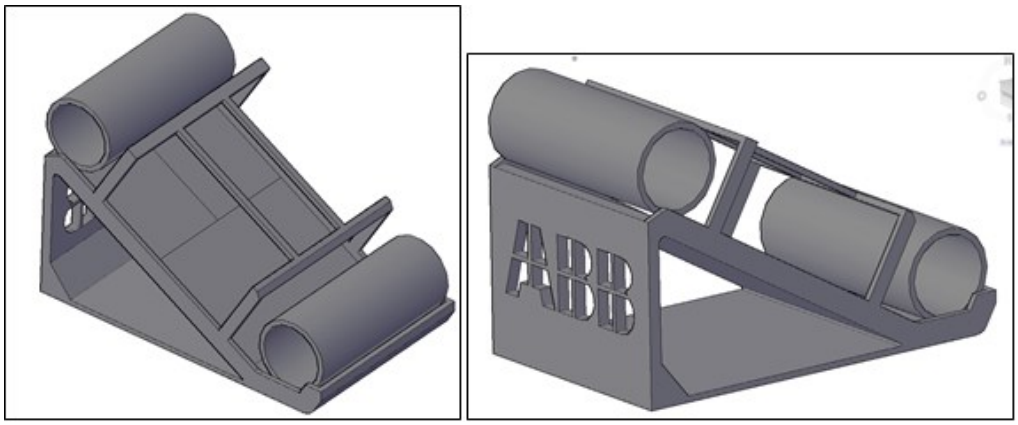


Figura 6.3 a) Modelo CAD del dispensador vista 1. b) Modelo CAD del dispensador vista 2 (Fuente: [20]).

#### 6.1.1.4 Cilindro

Como se ha mencionado los Cilindros son unos tubos que la pinza manipula para colocarlos en la parte superior del elemento dispensador.

La longitud de los cilindros es conocida siendo de 75 mm, el diámetro no es un dato influyente a la hora de simular la práctica, solo debe de ser tal que permita el paso del cilindro por el dispensador sin tener un choque. Se ha comprobado que 30 mm de diámetro exterior es una medida aceptable.

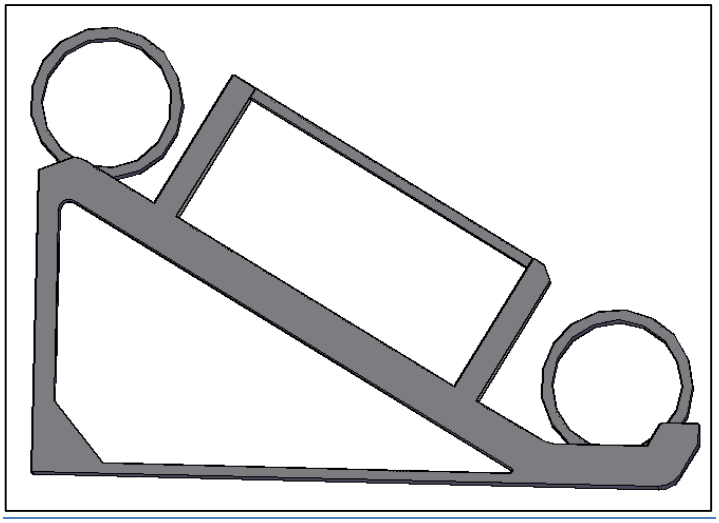

Figura 6.4 Modelo CAD dispensador con vista del diámetro de cilindros (Fuente: [20]).

6.1.1.5 Pinza

Para la pinza se usó un modelo en formato AutoCAD disponible en la propia página web de la empresa [19]. No fue necesario el uso de herramientas de escala ni ninguna herramienta externa.

6.1.1.6 Lápiz

Como se ha comentado el elemento lápiz es el conjunto de un rotulador y los accesorios que permiten manipularlo, Con este elemento se simulará una práctica en la que se ha de dibujar con el rotulador unas iniciales en un trozo de papel.

La herramienta Lápiz se compone de distintos elementos (Figura 6.5), de los cuales se disponen de planos, estos planos definen a los elementos en su totalidad.

Después de modelar estos elementos se unieron en uno solo, del que se hace uso para crear y configurar el componente inteligente SC_lápiz.

El valor de la medida que debe tener el último componente (el que representa al rotulador) es conocido y de 130 milímetros, esta medida es la medida del rotulador usado en el laboratorio, por tanto, si se cambiase de rotulador se debería cambiar esta medida.

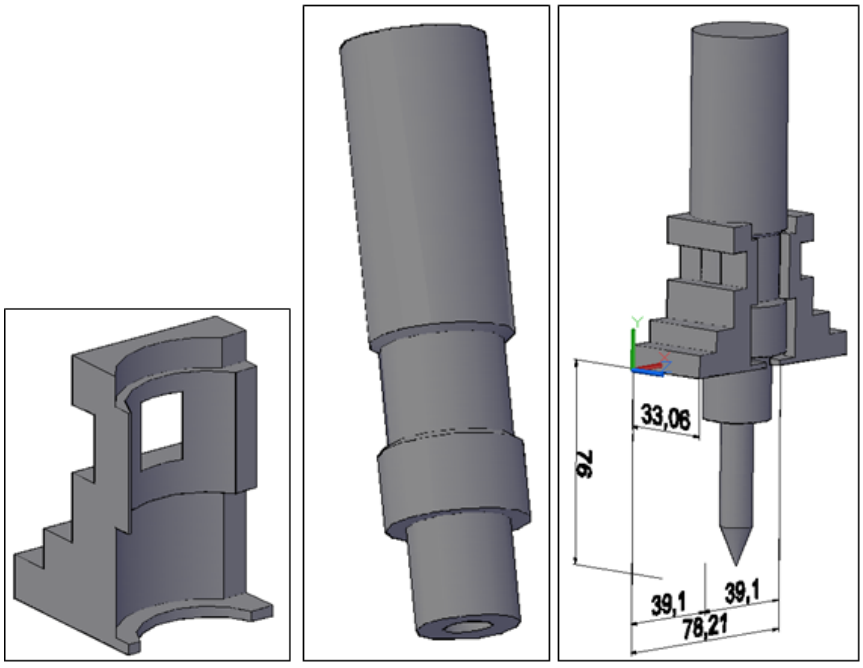


Figura 6.5 a) Modelo CAD de un componente lateral del lápiz. b) Modelo CAD del componente central del lápiz. c) Modelo CAD del componente lápiz completo (Fuente: [20]).

6.1.1.7 Cajas

Las cajas son unas geometrías sencillas creadas con el fin de simular una práctica que cambia la posición de los objetos para luego apilarlos de una forma determinada. La simplicidad de las cajas hizo posible la creación de este modelo tridimensional, desde el propio RobotStudio, el programa dispone de herramientas de modelado básicas (Figura 6.6).

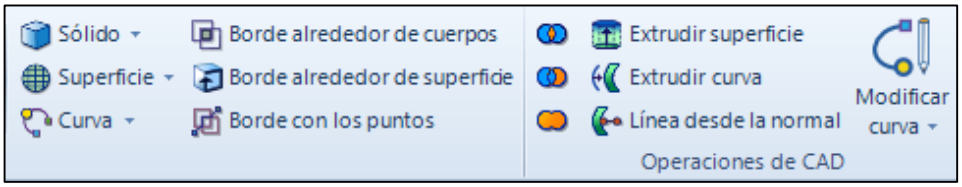


Figura 6.6 Menú de creación dentro de la pestaña Modelado.

Los cubos han sido creados usando la herramienta crear sólido/tetraedro, (aunque se puede observar que en realidad es un hexaedro) abriendo la ventana visible en Figura 6.7, se crearon tres tetraedros de base cuadrada de 60,75 y 80 mm de lado y una altura común de 40 milímetros.

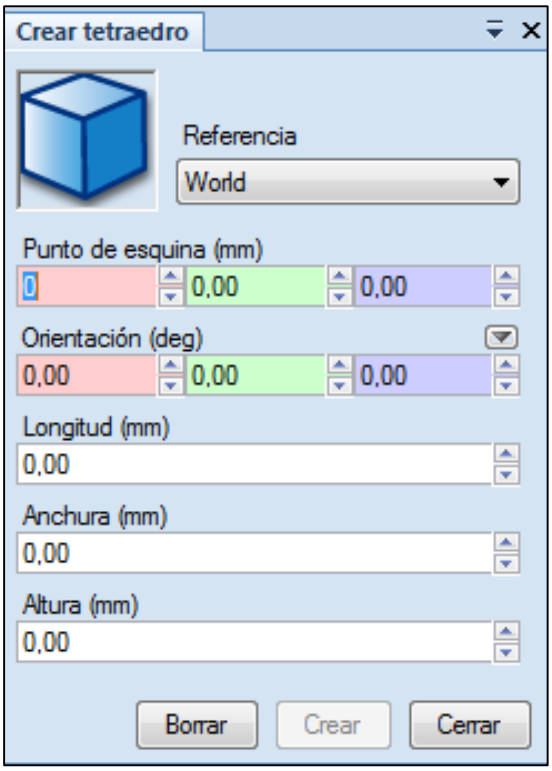


Figura 6.7 Ventana de creación de tetraedro (Hexaedro).

Para poder usar estos modelos como archivos independientes (que no deban de crearse cada vez que se desee añadirlos a una estación) se debe de exportar la geometría, para ello se debe de seleccionar la pieza creada y pulsar el botón derecho del ratón, entre las opciones disponibles debe hallarse una que ponga exportar geometría.

Al seleccionar exportar geometría se abrirá una ventana, en esta ventana será posible seleccionar el formato con el que guardaremos el modelo, se tomará en este caso el formato .obj (la elección de cualquier formato no debería afectar para nada la importación posterior del modelo).

#### 6.1.1.8 Accesorio de unión Brazo-Pinza:

En el laboratorio, la pinza está unida al brazo con una placa metálica, este accesorio ha sido medido en el laboratorio, con estas medidas se hizo un modelo tridimensional, pero se observó que esta pieza no tiene ninguna aplicación práctica salvo separar el extremo del brazo robótico con la pinza una determinada distancia, por esta razón se modificó el modelo para que fuera un rectángulo, como se puede ver en la Figura 6.8.

Esta modificación simplifica la selección del centro de las caras del accesorio, además se hizo un corte en una cara, este corte permite la selección de unas aristas que coinciden con unos puntos determinados de la pinza, y de esta forma la pinza queda completamente situada respecto al brazo.

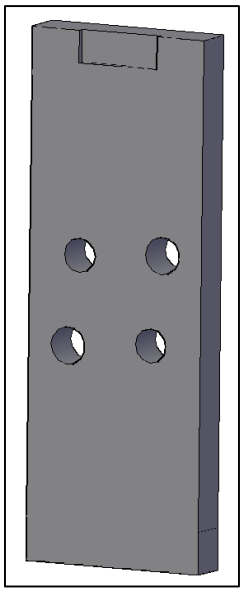

Figura 6.8 Modelo 3D Accesorio Pinza-Brazo.

### 6.1.2 Herramientas

En este apartado se procederá a comentar los pasos a seguir para crear una herramienta a partir de los modelos tridimensionales de pinza [12], [20]. Estos modelos han sido obtenidos de forma gratuita desde la página web del fabricante de la herramienta.

Sin embargo, por problemas al importar los formatos disponibles desde RobotStudio se tuvo que cargar el modelo descargado en AutoCad para después guardar el archivo en formato ".sat" (un formato que RobotStudio es capaz de reconocer), el cuerpo y los dedos de la pinza fueron separados desde AutoCad y guardados como archivos independientes de formato ".sat". Esta separación se debe a que al crear una herramienta virtual en RobotStudio las partes móviles de esta deben de ser elementos individuales

Para crear un mecanismo, primero se han de importar las tres geometrías de interés: el cuerpo y los dos dedos. Seguidamente se ha de ir a la pestaña de Modelado, ubicada en la esquina derecha el apartado de mecanismos, se pulsa en 'Crear mecanismo' (Figura 6.9).

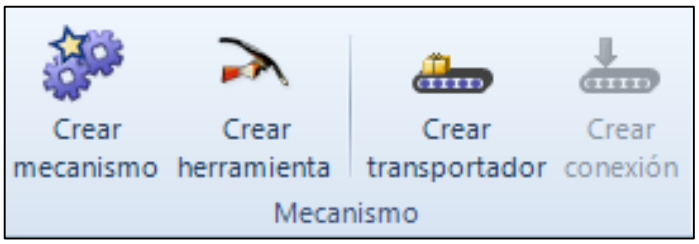


Figura 6.9 Menú de mecanismos de RobotStudio.

Se abrirá la pestaña que permitirá configurar un mecanismo (Figura 6.10), primero se debe determinar que nos encontramos en la configuración de una herramienta, para ello se ha de seleccionar la opción de herramienta en “Tipo de mecanismo”.

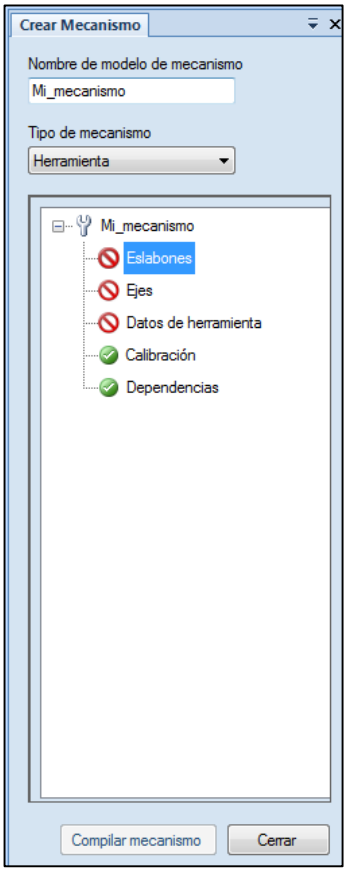


Figura 6.10 Ventana de creación de mecanismo.

Ya determinado el mecanismo como herramienta, se debe pulsar en el elemento desplegado “eslabones” (Figura 6.10), esto abrirá una ventana emergente (Figura 6.11), en

esta ventana se determina cuáles son los eslabones que tienen movimiento y cuales no lo poseen.

Se empieza con la base, que es la que no dispone de movilidad, para ello se selecciona la pestaña de “Establecer como eslabón base” (Figura 6.11), los otros dos se hacen de forma similar sin seleccionar “Establecer como eslabón base”.

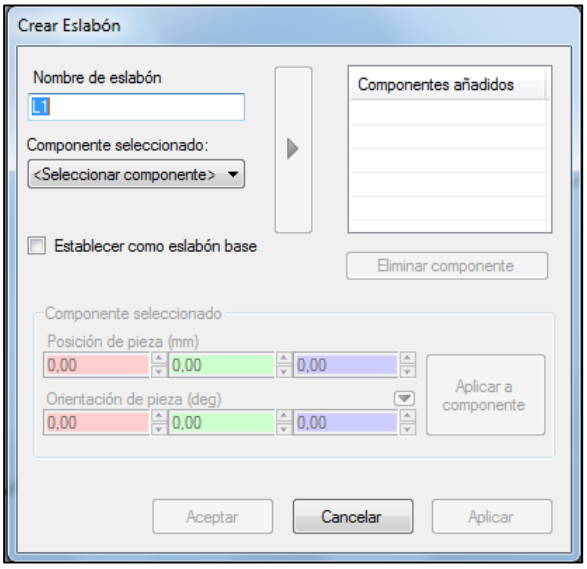


Figura 6.11 Ventana de configuración de eslabones.

Ya rellenado este apartado se debe cerrar la ventana y continuar con configurar en qué dirección y cuanta distancia tienen disponible los dedos de la pinza para realizar su movimiento. Se seleccionará “Ejes” abriéndose una ventana emergente (Figura 6.12).

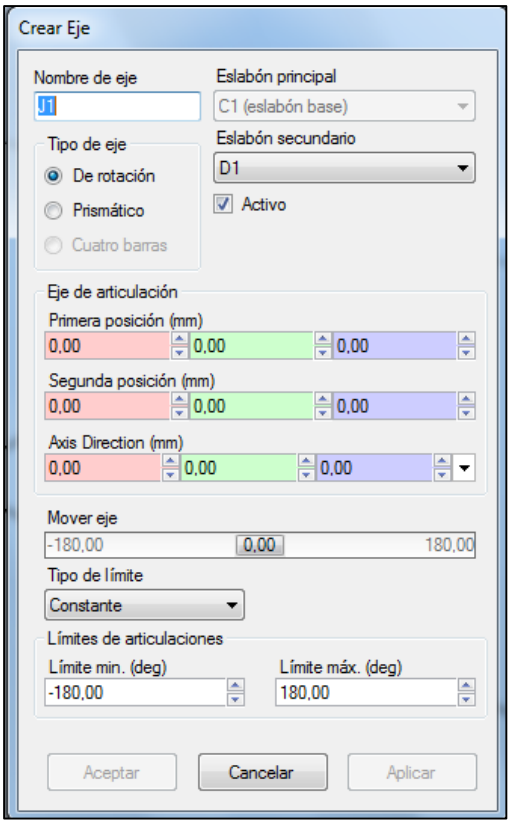


Figura 6.12 Ventana de configuración de ejes 1.

Los dedos de la pinza se mueven en un eje, con un movimiento prismático, por ello se debe seleccionar la opción de prismático.

En la Figura 6.13 se observa cómo se ha seleccionado el eje como la dirección del movimiento del eslabón, además se ha determinado que el rango de movimiento será de -7 a 0, este rango es conocido, ya que la apertura máxima de la pinza es de 14 milímetros, repartiendo equitativamente 7 milímetros en cada eslabón se obtiene esta apertura deseada. El otro eslabón “dedo” será similar, sin embargo, su rango de movimiento será de 0 a 7.

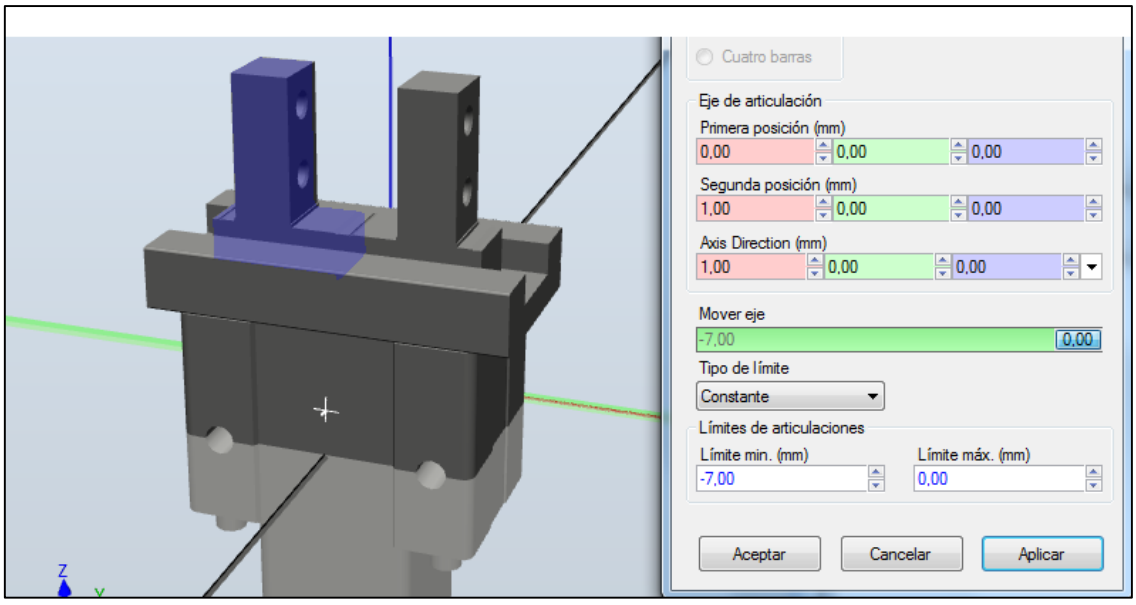


Figura 6.13 Ventana de configuración de ejes 2.

El siguiente paso será definir los datos de la herramienta, esto determina el punto que el software usará como referencia a la hora de mover la herramienta. Es importante elegir un punto del que sea fácil determinar distancias cuando se definen los puntos en la programación de los movimientos.

En este caso se ha seleccionado el punto medio de la cara en contacto con los dedos del eslabón cuerpo (Figura 6.14), este punto es equidistante con los dos dedos, una propiedad interesante en las pinzas. Es importante que se conozcan las direcciones de los ejes en el punto decidido, serán necesarios a la hora de determinar los puntos de agarre al tomar piezas. No es necesario seleccionar este punto en particular, solo es necesario tener en cuenta las distancias desde el punto seleccionado.

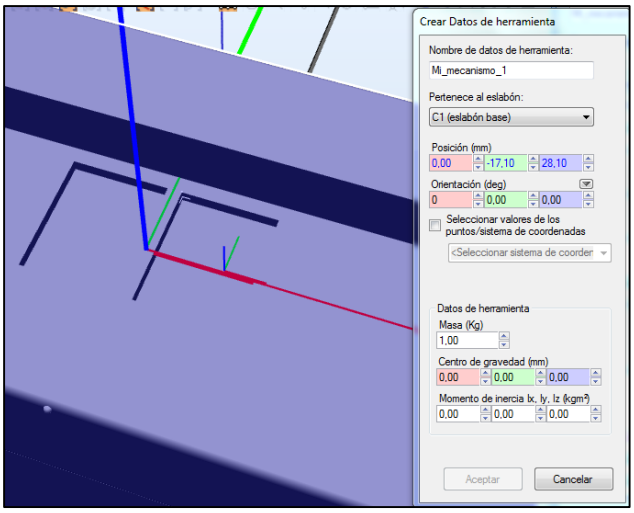


Figura 6.14 Ventana de "Datos de herramienta".

Como no es posible trasladar estas simulaciones de manera total al robot real el dato referente a la masa no es relevante.

El último paso es configurar las dependencias (Figura 6.15):

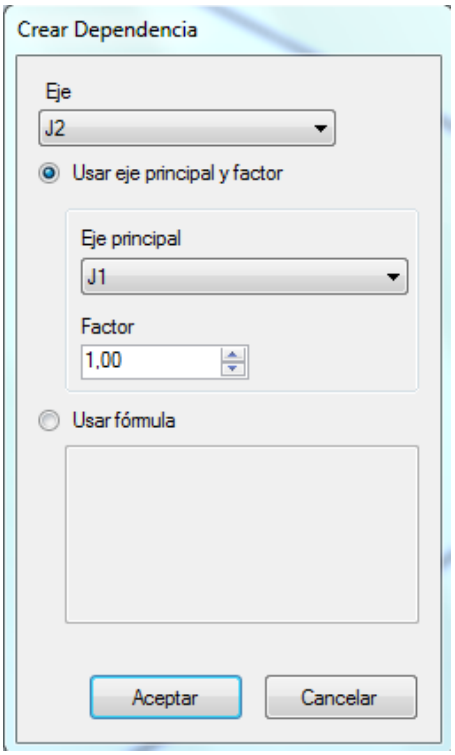


Figura 6.15 Ventanas de dependencia.

Se selecciona un eje y se hace “esclavo” del otro, de esta manera solo se tiene que aportar la información de uno de ellos para cerrar la pinza.

Para terminar el proceso y crear el mecanismo solo se debe pulsar el botón de compilar mecanismo. Pulsando la pestaña de diseño en la parte lateral de la pantalla se podrá observar que se encuentra nuestro mecanismo. Para guardar el mecanismo para futuros usos en otras estaciones se ha de pulsar el botón derecho del ratón y seleccionar “guardar en como biblioteca”, creando un archivo en el formato propio de RobotStudio.

### 6.1.3 Componentes inteligentes

En este apartado se explicarán los componentes inteligentes creados para este trabajo y sus funciones.

Un componente inteligente en RobotStudio es un modelo al que se le han añadido unas entradas y salidas de señales, las cuales interactúan con distintos elementos disponibles que simulan las funciones de elementos reales, ejemplo de esto son sensores, movimientos en una dirección, generación aleatoria de componentes, etc.

Estos elementos se relacionan con puertas lógicas como IF, NOT, NOR, también se permite comparar señales, comparar geometrías y muchas más funciones.

Todas las funciones que se usen en este trabajo serán explicadas a medida que sean necesarias.

Como paso inicial y común en la creación de un componente inteligente vacío, se debe ir a la pestaña de modelado en la parte superior de la pantalla y pulsar en “Componente inteligente” (Figura 6.16).

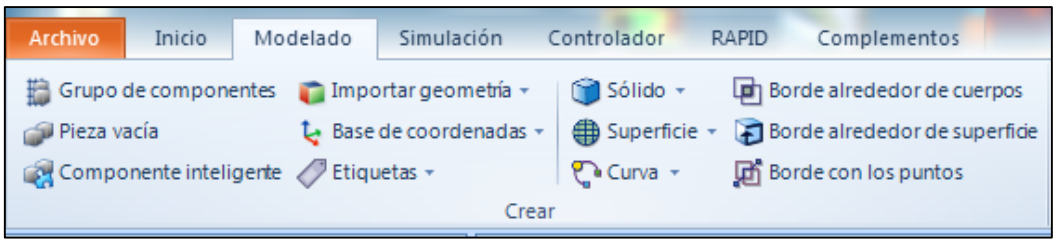


Figura 6.16 Menú de creación dentro de la pestaña Modelado.

Se abrirá en la ventana de visualización la ventana de componentes inteligentes, en ella se observa que es posible agregar geometrías tridimensionales. Como primer paso se importa el modelo 3D (importando la geometría) al que se quiere convertir en componente inteligente, esto es visible en la Figura 6.17 y Figura 6.18.

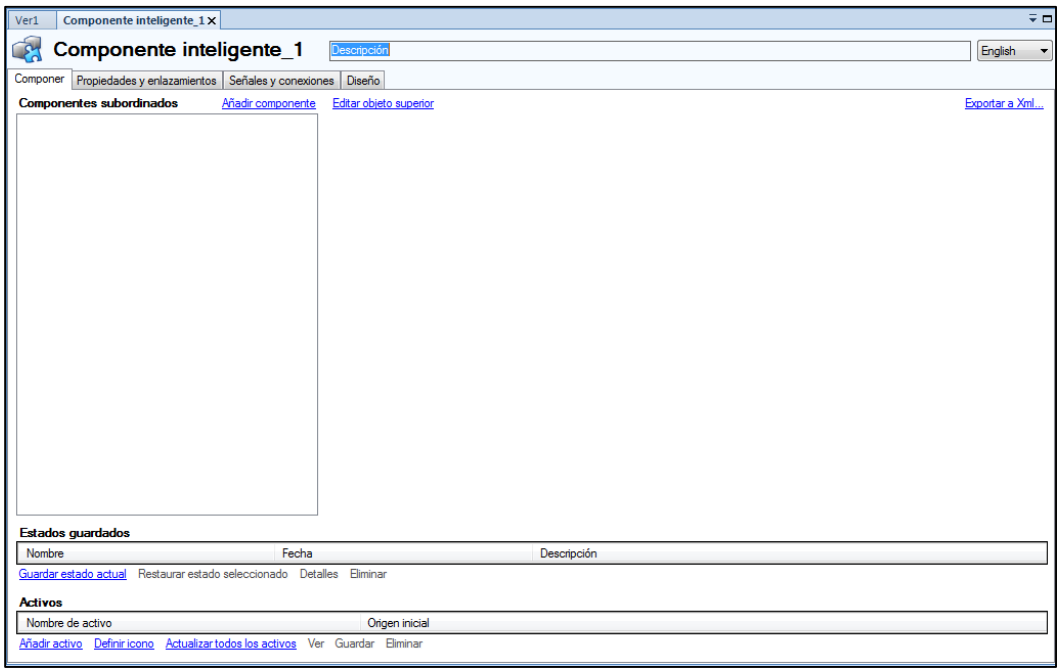


Figura 6.17 Ventana de Componente inteligente.

Hecho esto se puede añadir los elementos que se consideren necesarios para la configuración desde las pestañas de "señales y propiedades" (Figura 6.18).

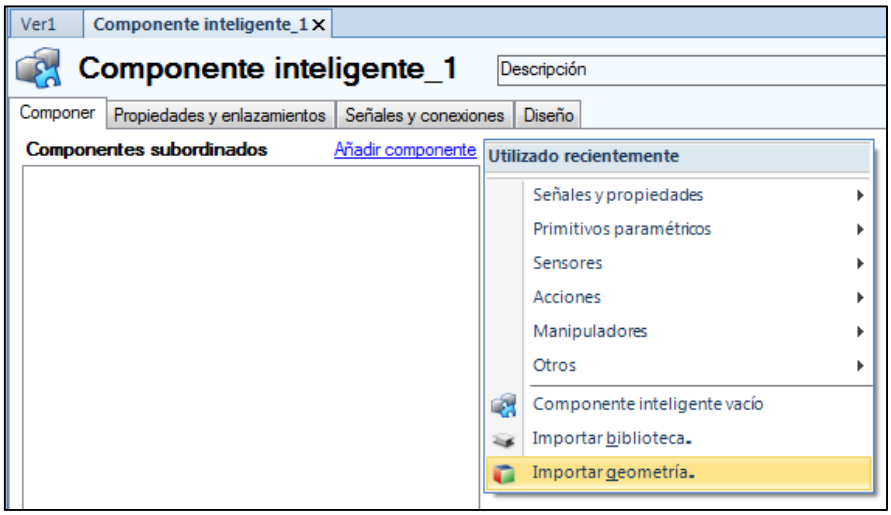


Figura 6.18 Añadir componente, componente inteligente.

Sin embargo, se ha observado que es más practico ir directamente a la pestaña de "diseño"', visible en la esquina superior derecha de la ventana de visualización (Figura 6.19).

En esta ventana será posible visualizar mediante cajas las conexiones entre los elementos creados, las cajas están unidas por líneas de colores (las líneas de color verde representan señales, las rojas representan propiedades).

Desde esta ventana no solo es posible visualizar el diseño, también es posible añadir las señales de estrada, salida (pulsando el símbolo + al lado de los cuadros de entradas y salidas) y los elementos. Para añadir elementos se debe de pulsar el botón derecho del ratón los que abrirá un menú visible en la Figura 6.19.

Este menú es similar al similar al visible en la ventana vista anteriormente.


Figura 6.19 Añadir componentes desde la pestaña de diseño.

### 6.1.3.1 Componente inteligente, Pinza

Este componente inteligente debe de tomar el archivo de herramienta creado con anterioridad para que el software lo identifique correctamente.

Este componente inteligente es, evidentemente, el más usado en este trabajo, en un principio se decidió crear este componente sin ningún tipo de retroalimentación mediante sensores (de esta forma era más parecido al disponible en el laboratorio).

Sin embargo, la retroalimentación es necesaria para que las simulaciones se llevasen a cabo, el programa debe de recibir información para poder saber en qué momento unir un objeto a la pinza o cuando soltarlo.

Esto obligó a incluir los sensores necesarios para la retroalimentación de la simulación en los objetos tomados por la pinza, decisión que acarrea ciertas complicaciones técnicas ya que el software no está diseñado para este planteamiento.

La Figura 6.20 presenta el diseño del componente inteligente de la herramienta pinza, este es el único caso en el que el componente inteligente no se hace sobre una geometría estática, al hacerlo con la herramienta definida anteriormente se permite usar los elementos de movimiento de ejes.

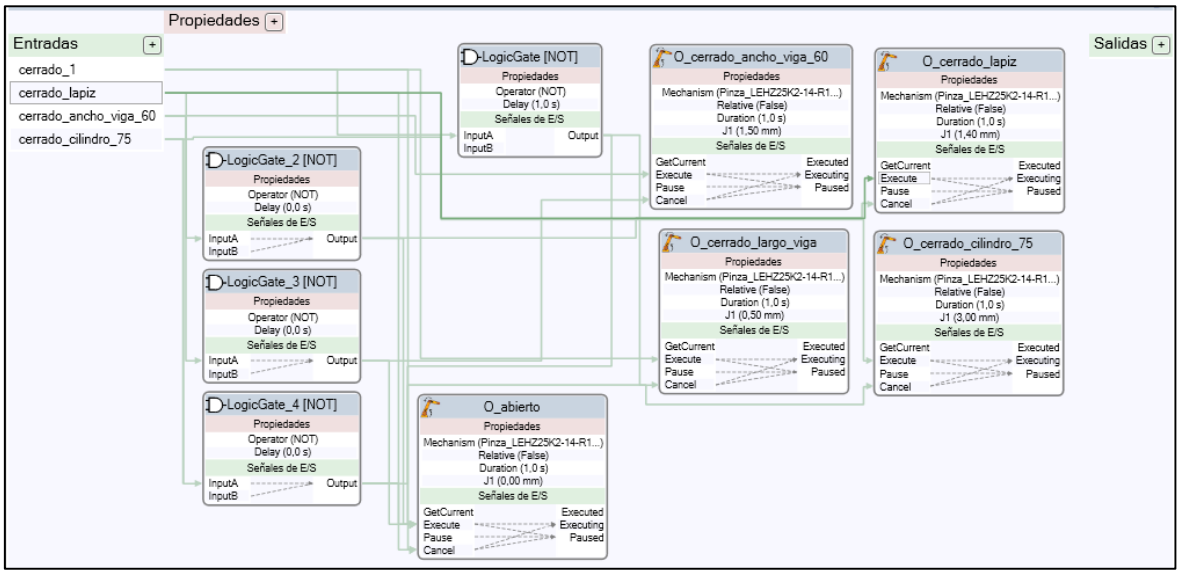


Figura 6.20 Diseño del componente inteligente SC_Pinza.

Los elementos visibles en este componente inteligente son:

- Puerta lógica NOT

Como su nombre indica actúa como la puerta lógica NOT

| Entrada | Salida |
|---|---|
| 0 | 1 |
| 1 | 0 |

Tabla 6-1 Entradas/Salidas puerta lógica NOT.

- MoveJoint

Este elemento permite el movimiento de un eslabón, para ello, primero hay que seleccionar el eslabón a mover, (en nuestro caso será el del dedo que se haya configurado como el maestro), también se debe imponer hasta que distancia el eslabón se mueve, para ello se ha de tener en cuenta que con esta orden se mueven dos eslabones, lo que implica que 0.5 mm es en realidad un cierre de 1 mm. (Figura 6.21).

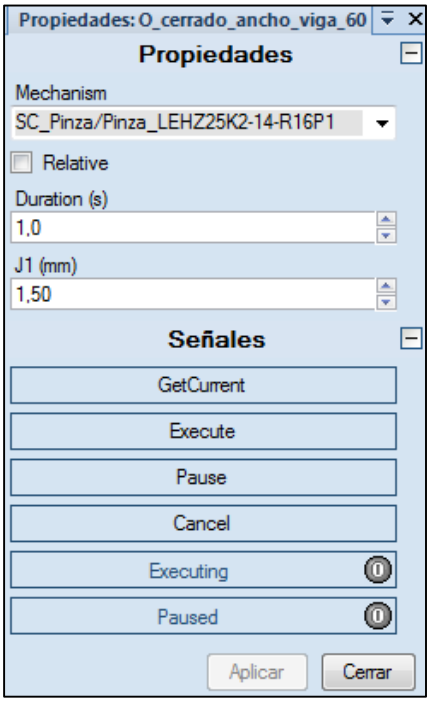


Figura 6.21 Ventana "Propiedades" de MoveJoint

Otra información relevante es la duración del movimiento, que permite una simulación más realista.

Hay un MoveJoint para cada cierre necesario en la realización de este trabajo y uno solo para apertura. Para la apertura solo hay que definir el punto al que se debe mover el eslabón maestro a 0.

RobotStudio trabaja de manera secuencial en su código, lo que significa que, si todas las señales están a 0, pero se cambia una a valor 1 la pinza se abrirá, aunque haya 3 señales de entrada en 0 que dan orden de apertura.

Los nombres de señales de entrada y salida de los componentes inteligente son útiles para simplificar la comprensión de un diseño de componente inteligente y de lógica

de estación posterior. Se recomienda elegir nombres que estén relacionados con la función que cumplirán.

Sin embargo, nada impide que una señal de entrada tenga cualquier otra nomenclatura. Pero se debe tener en cuenta que no se admite cifras como primer elemento del nombre. Por ejemplo, 1_entrada no es válido, en cambio práctica_1 sí es admitido.

Los nombres que se decidieron para las señales indican información de la situación en la que debe usarse, a excepción de la señal de cerrado_1, el nombre de la señal no muestra información alguna (esto se debe a que fue la primera en crearse), sin embargo, observando las demás entradas se puede deducir que debe de ser la de cierre para objetos de 80 milímetros.

6.1.3.2 Componente inteligente, Viga:

Para la viga hay se han creado dos versiones del mismo componente inteligente.

El primero es el que debe reorientarse en varias posiciones, por tanto, la viga se ha de agarrar en dos posiciones distintas, esto obliga a añadir varios planos de sensores, requiriendo la inclusión de dos “circuitos” dentro del componente inteligente, uno por cada sensor.

El segundo es el que se la viga debe de apilarse, en esta versión uno de los sensores es suprimido, la razón de esto es que al apilar las vigas estas se detectarían entre sí, por tanto, uno de los “circuitos” creados para la primera versión del componente inteligente se suprime.

El diseño de este componente inteligente es Figura 6.22:

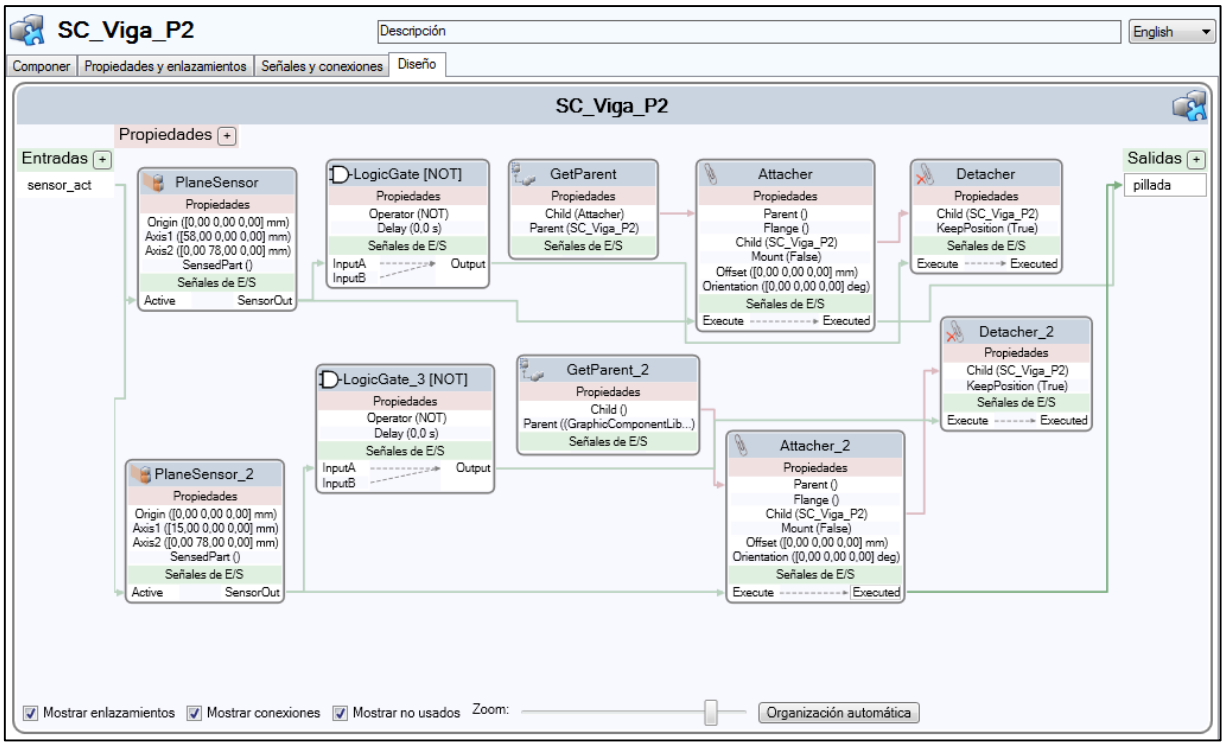


Figura 6.22 Diseño del componente inteligente Viga.

Este diseño incluye varios elementos nuevos:

- Plane sensor:

  Crea un plano sensor, que dará una señal 1 cuando un objeto lo atraviese y 0 cuando no, la Figura 6.23 muestra la ventana de configuración del plano.

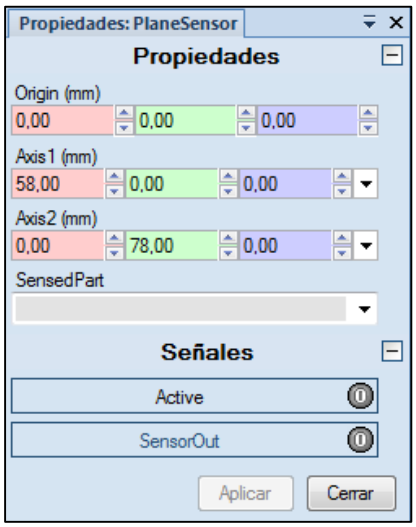


Figura 6.23 Ventana de configuración de sensor.

En “Origin” (Figura 6.24) se determina el origen del eje del plano, es útil para colocar en un primer momento el plano de manera aproximada, pero luego se vuelve irrelevante. Con Axis 1 y 2 se determina la anchura y altura del rectángulo que será nuestro sensor, para evitar problemas con el contacto con otros elementos o sensores se decidió que el área de los sensores de plano siempre sea menor que el área de la cara que ocupan, también se ha dado uso de la orden offset para mover el plano 0.1 mm de la superficie de la que sirven de sensor.

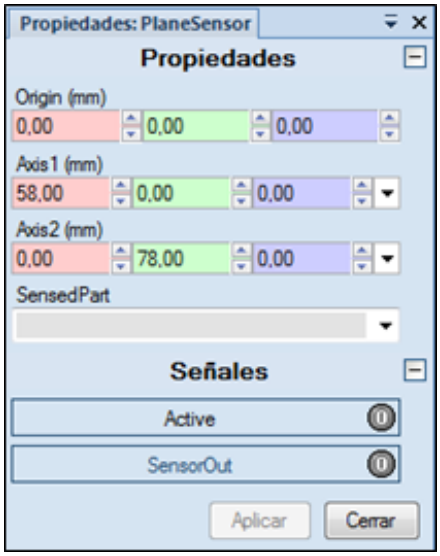


Figura 6.24 Ventana de configuración de sensor.

Para llevar esta orden se debe pulsar el botón derecho del ratón mientras se tiene seleccionado el plano, y seleccionar “Posición de offset…” (Figura 6.25 a), en la orden offset es importante tener en cuenta desde que referencia se hace el movimiento (Figura 6.25 b). (el código de colores usado en este software es el siguiente, el rojo representa el eje “x”, el verde el “y” y el azul el “z”, en caso de duda se insta a observar la esquina inferior izquierda de la representación virtual, en la esquina observaremos un eje de coordenadas con los colores.)

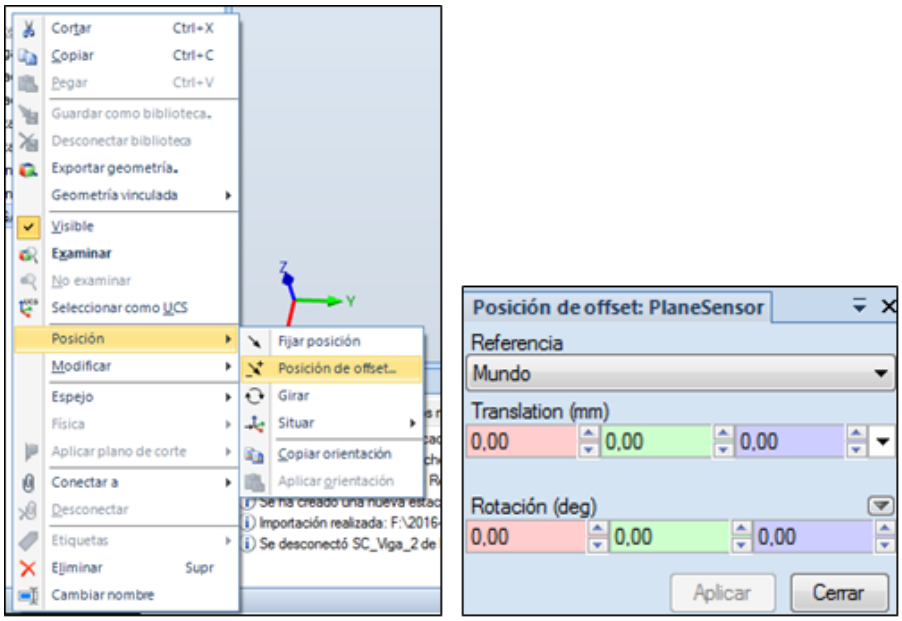


Figura 6.25 a) Selección de la orden Offset. b) Movimiento Offset.

- Attacher:

Este elemento, cuya ventana de configuración es visible en la Figura 6.26, es muy importante y se encuentra en todos los componentes inteligentes que se cojan con la pinza, y cuando se activa crea una orden de unión entre varios elementos.

Es el primer elemento mencionado que presenta “elementos” en la parte denominada propiedades, haciendo referencia a elementos dentro de la estación. Esto significa que no solo pueden ser seleccionados dentro de la configuración principal, sino que esta información puede provenir de otro componente del diseño (como se observa en la Figura 6.22).

Cuando este elemento ha ejecutado su función se activa una señal pulso en “executed”, que se ha unido a la señal de salida nombrada como “pillada”, de esta manera se ha podido crear una retroalimentación en la estación que permite al robot saber que una pieza ha sido tomada por la pinza.

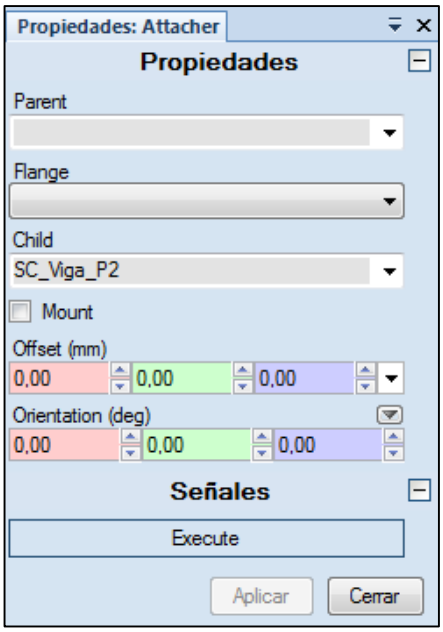


Figura 6.26 Ventana de configuración de Attacher.

La propiedad “Parent” de Attacher es el elemento al que se une el objeto manipulado al ejecutarse esta función. En este trabajo el “Parent” es el accesorio de la pinza al brazo, se hace así porque el elemento accesorio es una geometría sin ejes de movimiento que siempre se encuentra a la misma distancia de la pinza, se podría usar el cuerpo de la pinza, pero aumentaría la complejidad del diseño al ser una geometría dentro de un componente inteligente (obligando a usar otras funciones para poder seleccionarlo).

Los dedos de la pinza en cambio no podrían ser usados, al abrir la pinza para soltar el elemento el elemento unido seguiría al dedo en vez de solarse de él.

La propiedad “Child” es el elemento que se mueve, esta propiedad puede unirse con el de ‘Detacher’, haciendo que siempre sean el mismo.

- Detacher:

Detacher es el elemento complementario de "Attacher", permitiendo separar elementos unidos previamente Figura 6.27.

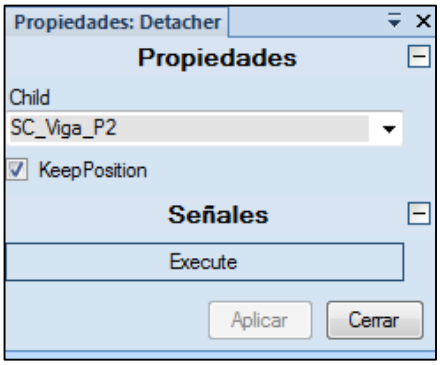


Figura 6.27 Ventana de configuración de Detacher.

"Keep position" es una opción que tiene como resultado que la orden de soltar la pieza tomada por la pinza esta no tenga como resultado la vuelta a ninguna coordenada origen, sino que se quede en la posición en la que se está soltando.

Como resultado del diseño elegido se tiene un componente inteligente que tiene una señal de entrada que activa unos sensores, mientras estos no detecten un elemento el componente inteligente no hará nada, pero si detecta algo el componente inteligente se unirá al accesorio de la pinza dando una señal tipo pulso a la señal de salida.

Se puede comprobar que este diseño hace que cualquier cosa que sea detectada por los sensores tiene como resultado la unión con el accesorio, aunque el elemento detectado no sean los dedos de la pinza, de manera que hay que tener cautela cuando se coloquen componentes inteligentes "Viga" entre sí.

Este problema como muchos otros podría resolverse si se hubiera colocado sensores en los dedos de la pinza.

Cuando se ha terminado el diseño del componente inteligente todos los componentes se unirán a la geometría para asegurar que los sensores y demás componentes que puedan verse afectados por el movimiento seguirá al elemento cuando este se mueva, y para ello se seleccionará todos los elementos que conforman el componente inteligente y se arrastraran a la geometría del componente inteligente como puede observarse en la Figura 6.28.

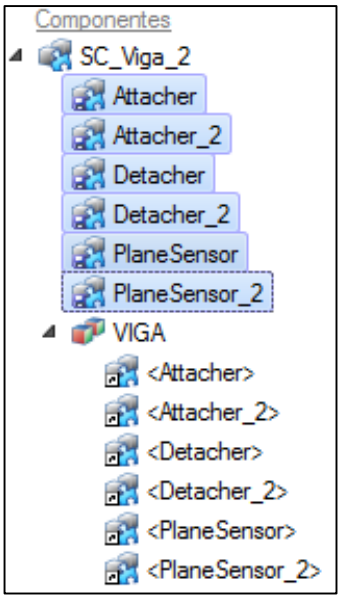


Figura 6.28 Unión de componentes a la geometría.

- Get Parent:

Get Parent, Figura 6.29 es un elemento que permite obtener el elemento al que pertenece otro, y se ha usado este elemento porque se observó que al abrir y cerrar el programa los componentes inteligentes tenían la tendencia de no recordar la propiedad padre del elemento Attacher. Usando el elemento Get Parent este problema desapareció.

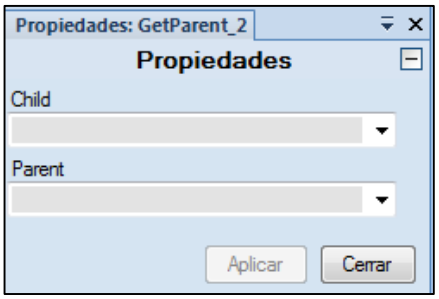


Figura 6.29 Ventana Configuración  Get Parent.

### 6.1.3.3 Componente inteligente Lápiz

El componente inteligente lápiz Figura 6.30 representa a varias piezas unidas incluyendo el rotulador con el que se dibuja en la práctica número 1.

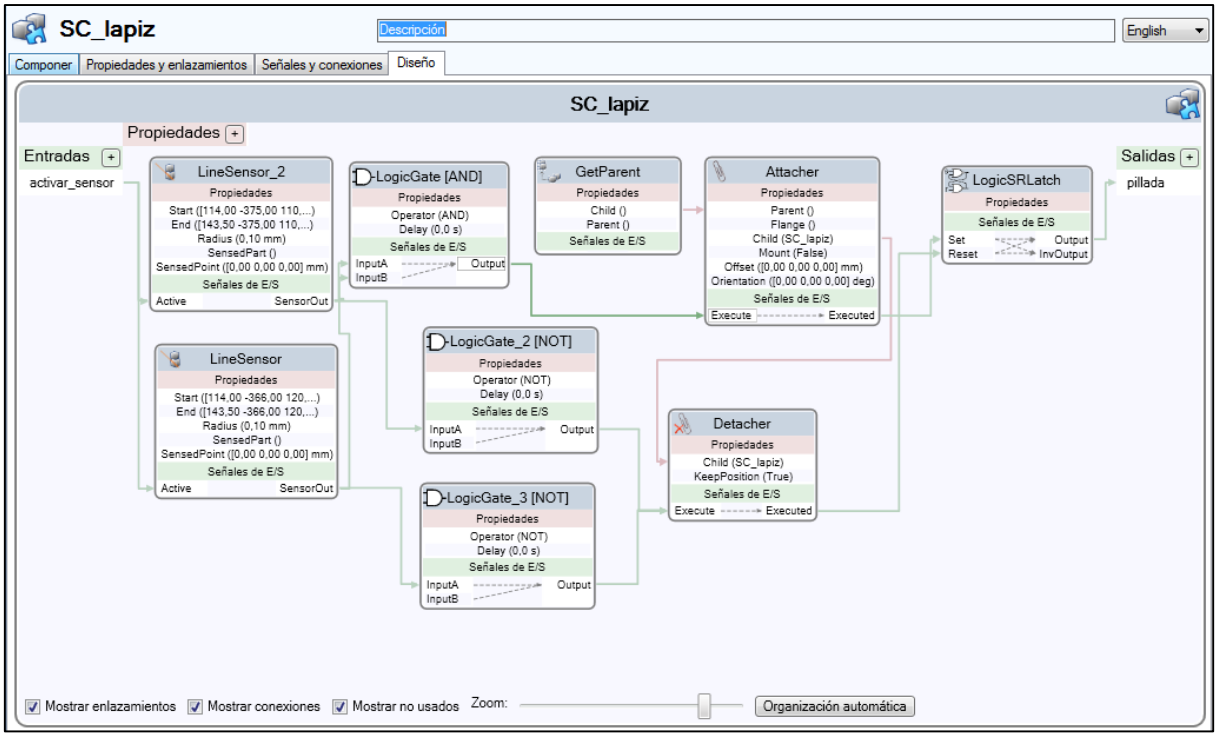


Figura 6.30 Diseño del componente inteligente Lápiz.

Los únicos elementos distintos en este diseño son uso de LineSensor y un LogicSRLatch:

- LineSensor:

  LineSensor, cuya ventana de configuración es visible en la Figura 6.31, se diferencia de PlaneSensor por ser una línea en vez de un plano, y de poder definirle un diámetro determinado.

  Se puede observar que se han de definir el origen y destino de la línea que define el sensor y el radio que tendrá el sensor.

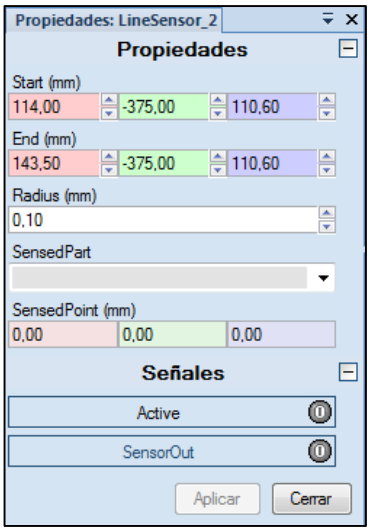


Figura 6.31 Ventana Configuración de LineSensor.

- LogicSRLatch:

Este elemento, visible en la Figura 6.32 permite convertir la señal pulso de salida de Attacher en una continua cuando se active la entrada Set y volver a dar una salida 0 cuando se reciba señal en reset.

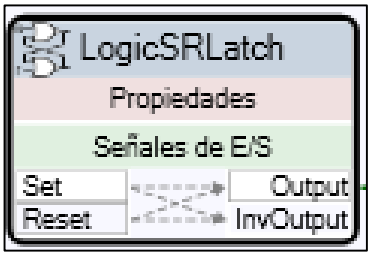


Figura 6.32 Elemento LogicSRLatch.

Existe además una inversa de la salida con valor 1 cuando la entrada set sea 0 y 1 cuando la entrada reset sea 0.

Este elemento es útil en caso de que se necesite mantener la información de una señal pulso como es el caso de la práctica nº5, no es necesario en las demás, aunque una correcta implementación no debería producir ningún problema.

6.1.3.4 Dispensador

El componente inteligente Dispensador, cuyo esquema es visible en la Figura 6.33, es un componente que debe simular el comportamiento de un elemento cuando sufre la acción de la gravedad. En esta versión de RobotStudio (o ninguna anterior) no se dispone de ningún motor de físicas, término con el que se denomina a herramientas en varios programas que simulan el comportamiento de elementos en un ambiente virtual.

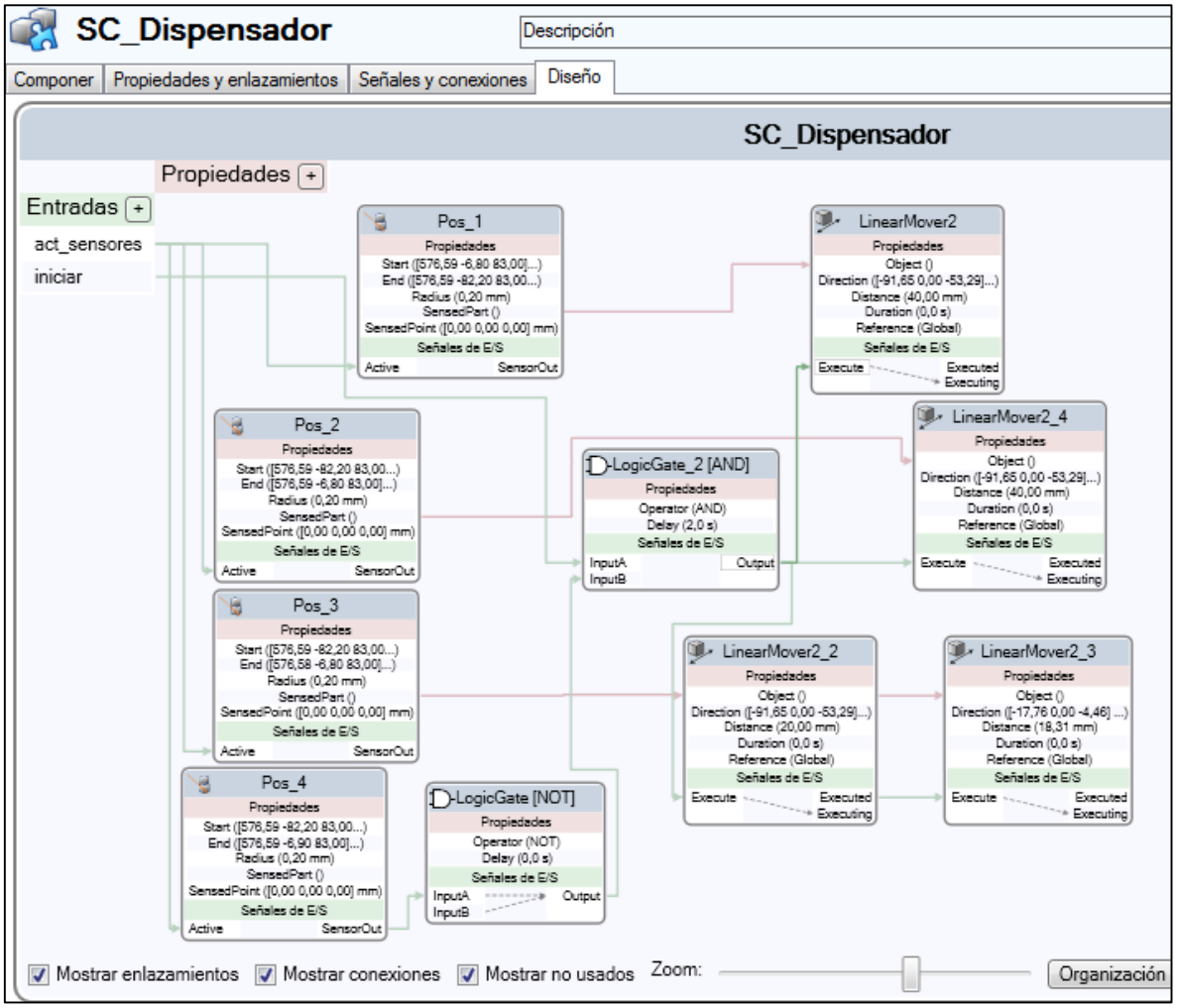


Figura 6.33 Diseño del componente inteligente Dispensador.

Los sensores colocados en el distribuidor muestran las posiciones en las que el cilindro debe pasar durante la simulación y, sirven para conocer si los cilindros están en su posición correspondiente.

La señal de entrada “iniciar” tiene un uso muy importante, ya que permite iniciar los sensores sin que se produzca ningún movimiento, ya que para que se realice cualquier orden

LinearMove2 deben de tener la señal iniciar activa. Al no disponer de un motor de físicas, se han tenido que usar herramientas y elementos propios de RobotStudio para simular los movimientos que sufren los cilindros.

Los elementos en el diseño que no han aparecido con anterioridad son:

- LogicGate AND

  Como su nombre indica es una puerta lógica AND

| Entrada 1 | Entrada 2 | Salida |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |

Tabla 6-2 Puerta lógica AND

- LinearMove2

  Este elemento, cuya ventana de configuración es visible en la Figura 6.34, tiene como función realizar un movimiento en una determinada dirección una distancia previamente dada de un objeto determinado previamente obtenido con el uso de sensores.

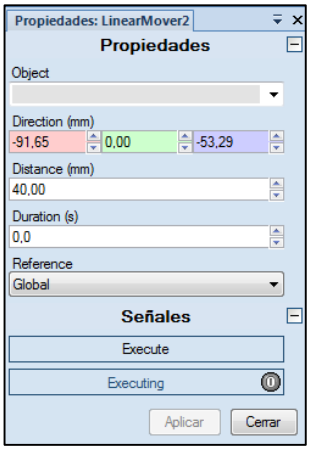


Figura 6.34 Ventana Configuración de LinearMove2.

En este caso el objeto está determinado por lo que detecte el sensor de una determinada posición, los cilindros (que son el objeto detectado en el distribuidor) van

cayendo de arriba abajo, lo que significa que la propiedad “objeto” del elemento LinearMove2 es variable.

La dirección que deben moverse los cilindros se ha tomado usando la herramienta de RobotStudio, Medir y seleccionar aristas (elementos que se encuentras en el visualizador de la estación).

Se ha de medir la distancia entre el punto medio de la arista que forma el principio de la pendiente del distribuidor y el punto medio de la arista del final de esta pendiente, RobotStudio dará como resultado no solo la distancia entre esos puntos, también informa de la distancia de los ejes.

Gracias a esto se conoce la dirección a la que deben moverse los cilindros sin que exista riego a cometer un error acumulativo que solo permita el realizar la simulación del movimiento de los cilindros un número limitado de veces antes que estos choquen con la geometría del dispensador.

La posición a la que deben ponerse los cilindros en el dispensador debe de ser tal que coincida con los sensores de línea colocados, estos sensores están a 40 mm entre sí menos el de la posición más baja.

Debido a no estar en un plano inclinado se determinó una posición que cumpliese la condición de que el cilindro colocado en la posición más baja no tocara al elemento distribuidor al manipular el cilindro.

Esta posición respecto del último sensor en la pendiente es: 20 mm en la misma dirección que los anteriores y aproximadamente 20 mm en dirección horizontal.

#### 6.1.3.5 Cilindro

En los cilindros se ha colocado dos planos sensores en un extremo. Estos planos paralelos entre sí, pero uno con un giro de 45 grados respecto el otro (ver Figura 6.35), formando una figura que se asemeja a la estrella de David, se ha hecho esto para asegurar que los dedos de la pinza son detectados.

El diseño de este componente Figura 6.36 es similar a la de la viga, ya que es un elemento inerte que se mueve con la pinza.

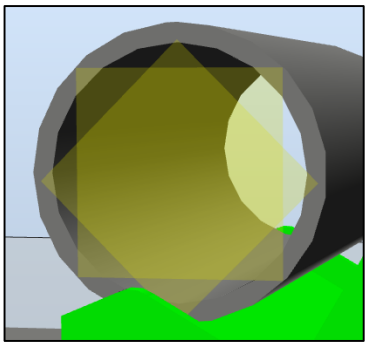

Figura 6.35 Vista de los sensores ubicados en el cilindro.

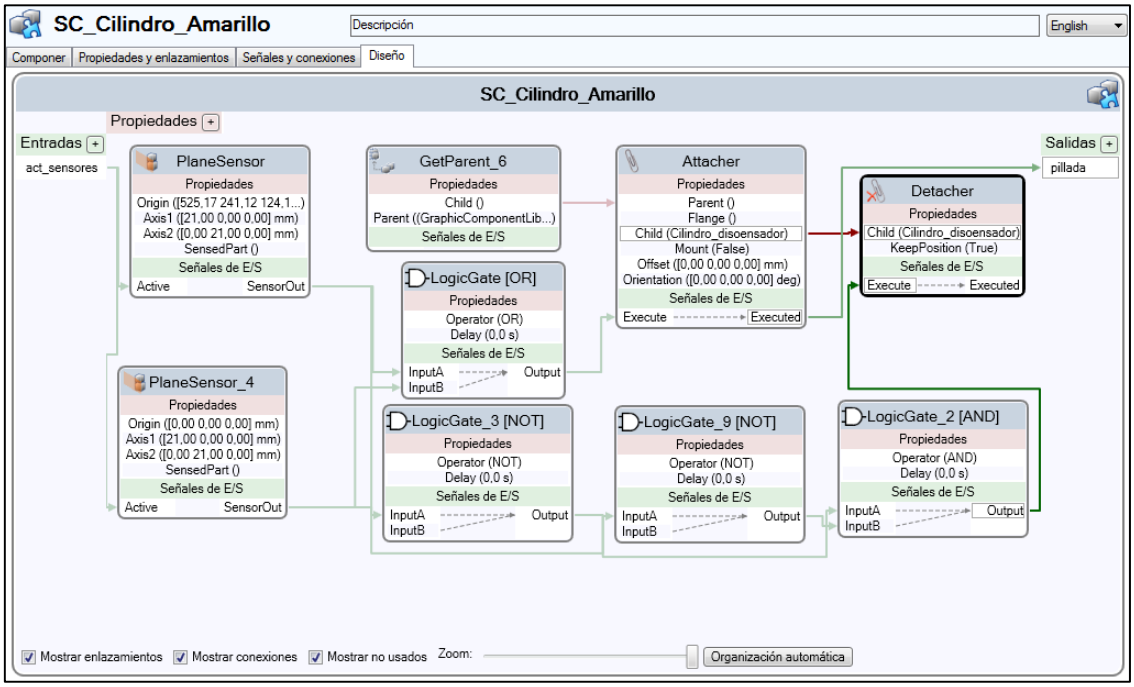


Figura 6.36 Diseño del componente inteligente Cilindro.

El único elemento no mencionado con anterioridad es el LogicGate OR.

- <u>LogicGate OR:</u>

Como su nombre indica es una puerta lógica OR

| Entrada 1 | Entrada 2 | Salida |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

Tabla 6-3 Puerta lógica OR

### 6.1.3.6 Componente inteligente, Sensores práctica 4

Este componente inteligente Figura 6.37 fue creado para la práctica nº 4, en la que se debe tomar en determinado orden vigas para luego apilarlos en una columna.

Para poder llevar a cabo esta práctica no es necesario la creación de este u otro componente, se podría incluir un contador en la programación al que se le sustrajese una unidad cada vez que apile una viga, sin embargo, estos sensores permiten al sistema conocer si aún quedan vigas sin apilar, por lo que es más fácil detectar fallos.

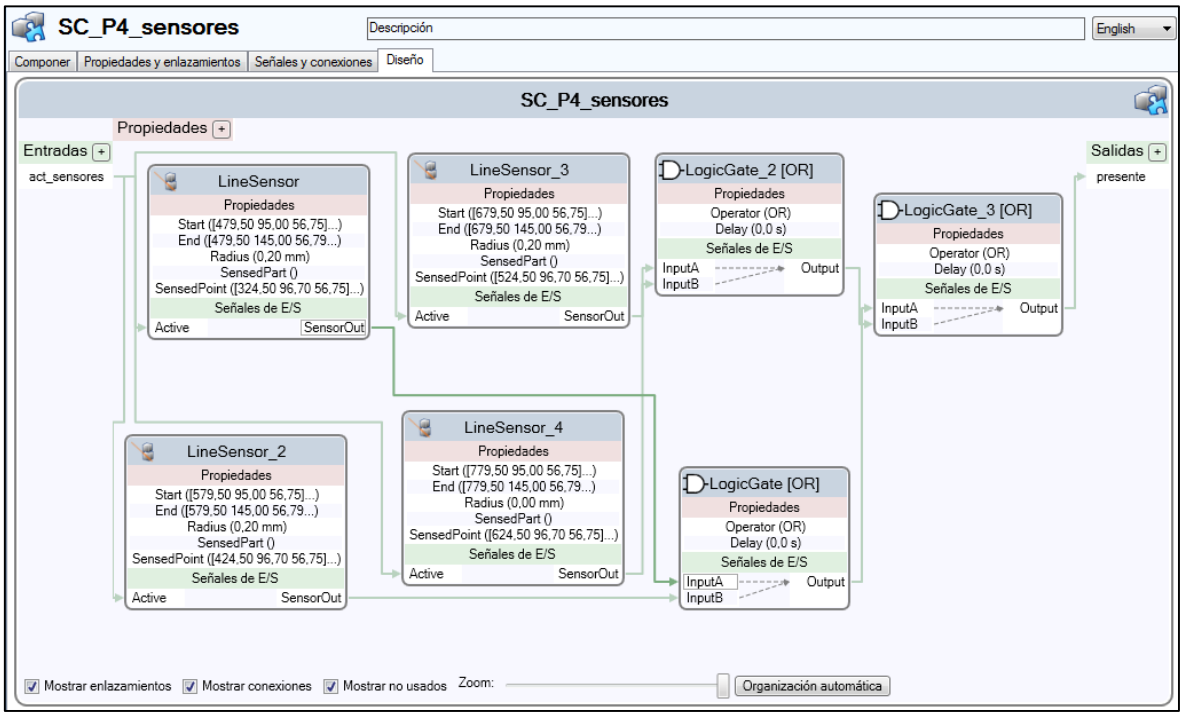


Figura 6.37 Diseño componente inteligente P4_sensores.

Todos los elementos que componen el componente inteligente han sido mencionados con anterioridad.

6.1.3.7 Componente inteligente, Cajas

Este componente inteligente (cuyo esquema es visible en la Figura 6.38) "Cajas" es el creado para la práctica propia de este trabajo, y en un principio este elemento (Este mismo componente se encuentra en tres tamaños distintos) aparecería en un punto determinado de forma aleatoriamente.

Esto obligaría al usuario a crear un sistema que debería identificar el elemento con sensores antes de poder tomarlo con la pinza, sin embargo, se comprobó que tal práctica sería inviable, y se debe a la decisión de que la herramienta pinza no tuviera sensor.

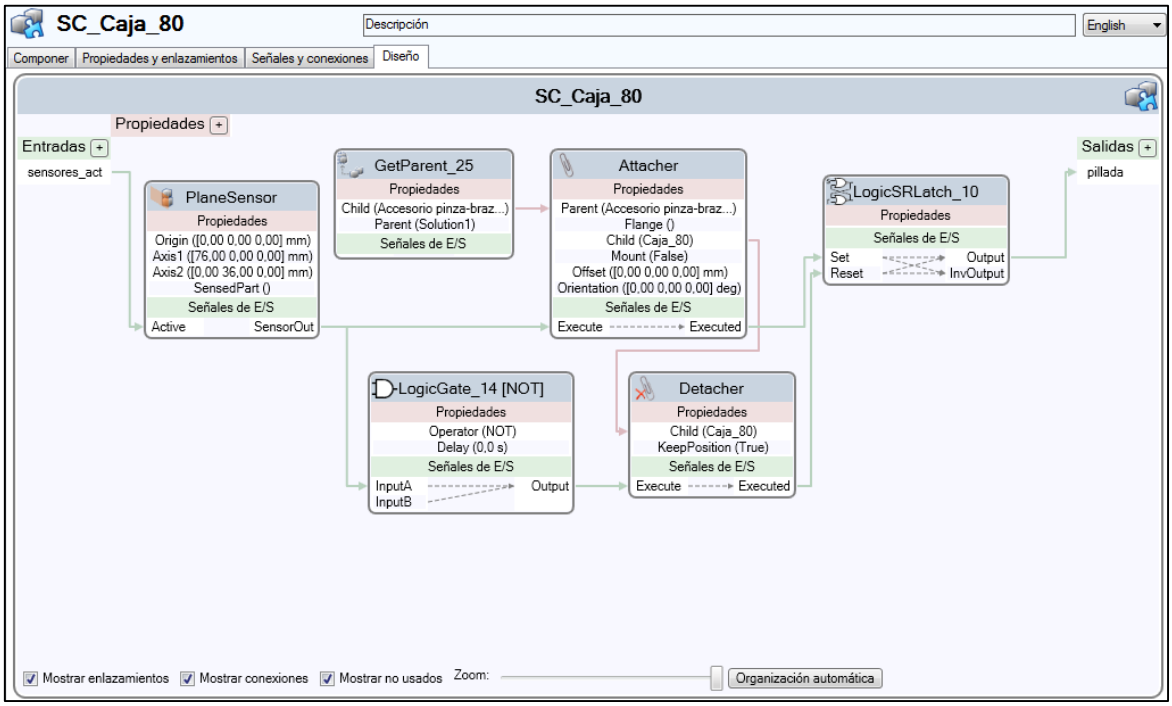


Figura 6.38 Diseño componente inteligente Cajas.

Todos los componentes inteligentes deben de estar conectados a la lógica de estación antes de ejecutar la simulación, intentar generar un componente inteligente de manera aleatoria durante la simulación impide conectarla al controlador del robot, lo que lo hace inerte en la simulación.

Se entrará en más detalles en cómo se ha dispuesto la práctica 5 más adelante en las guías de prácticas.

6.1.3.8 Componente inteligente. Sensores práctica 5

Ahora se va a proceder a describir el componente inteligente que junto al de las cajas compone la práctica 5. Este componente inteligente consiste en una serie de sensores y movimientos pre-definidos.

Debido al espacio que ocupa en pantalla la representación del diseño de este componente (Figura 6.41), se muestra dos fragmentos de la representación, siendo estos fragmentos los necesarios para la comprensión del funcionamiento de este componente.

La Figura 6.39 es la parte del diseño que representa como los componentes inteligentes Cajas pasan de la posición inicial (colocadas anteriormente en cualquier orden), a la posición de trabajo con la segunda línea de sensores.

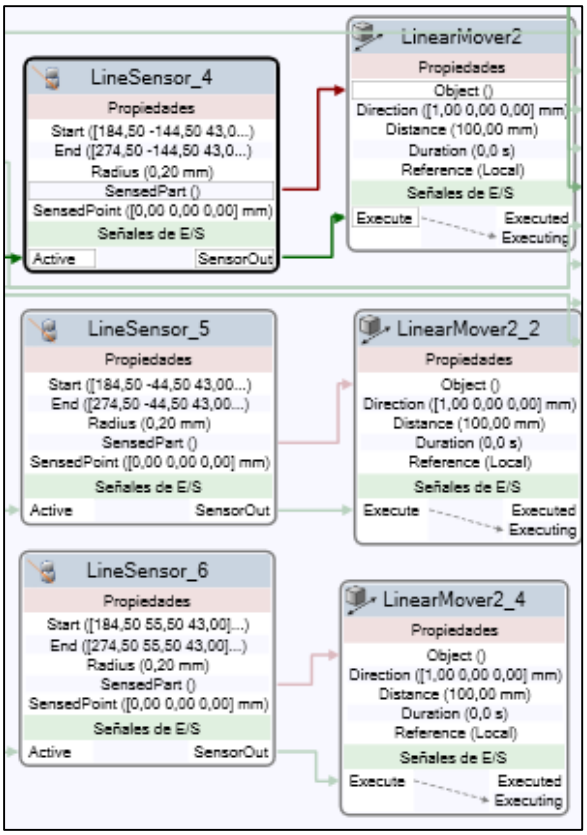


Figura 6.39 Zoom 1, diseño P5-componente.

La Figura 6.40 representa el bloque de elemento que puede discernirse en la esquina superior izquierda de la representación completa del diseño, este bloque tiene como objetivo identificar el elemento que detecta el sensor y activar una señal de salida particular para cada elemento posible, de esta manera se discierne que elemento se está detectando.

Este conjunto de funciones se ha repetido para las tres posiciones que ocupa una Caja.

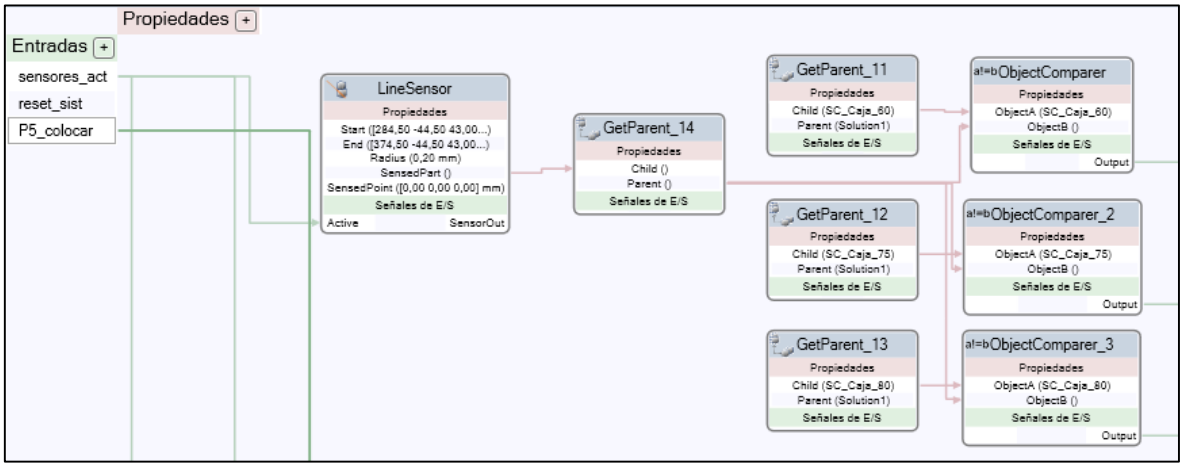


Figura 6.40 Zoom 2, diseño P5_componente.

El elemento no mencionado con anterioridad es ObjectComparer, como su nombre indica es un elemento que compara objetos, siendo uno de ellos el modelo a comparar y predefinido antes de la simulación

En nuestro caso no se podía seleccionar el objeto detectado en particular, (debido a que el software no detecta siempre los componentes inteligentes) lo que obligó a usar el elemento GetParent para convertir el elemento detectado por uno que se pudiera comparar, además por seguridad se aseguró que el elemento modelo se mantuviera estable definiéndolo con un elemento GetParent.

Las salidas de este componente representan todas las posibilidades que puede encontrarse al ubicar las cajas en distintas posiciones, la señal de entrada reset_sist no es relevante, es una señal residual de una configuración inicial que usaba elementos de LogicSRLatch, sin embargo, el elemento comparador de objetos no tiene una salida tipo pulso como el elemento Attacher, lo que hace el uso del elemento LogicSRLatch innecesario.

A continuación, se muestra una imagen completa del diseño del componente inteligente.

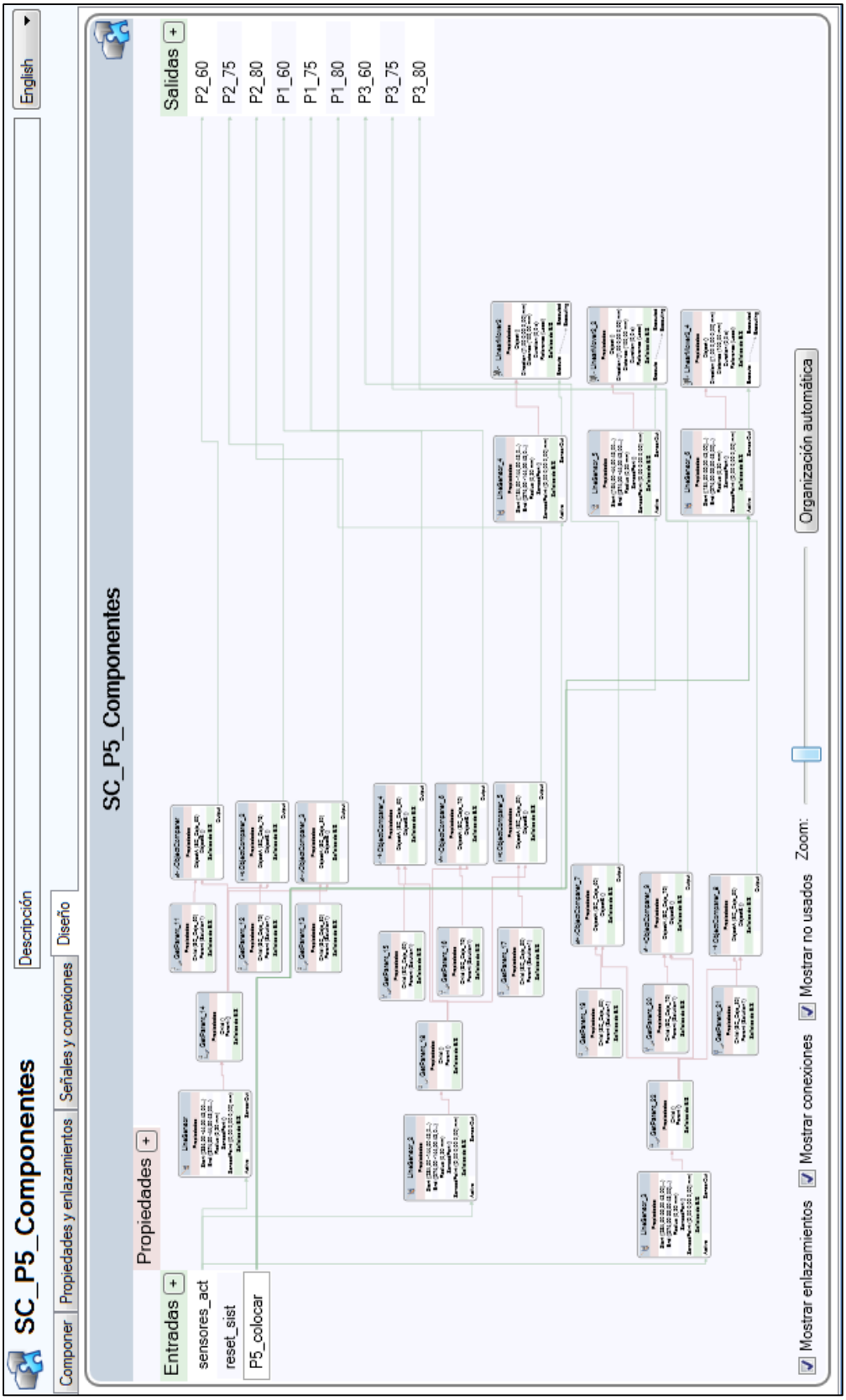


Figura 6.41 Diseño componente inteligente P5_componente.

## 6.2 Guía de prácticas

En esta parte se mostrará paso a paso como se ha recreado virtualmente la célula disponible en el laboratorio, permitiendo al alumno que lea este texto adquirir conocimientos básicos del uso del software RobotStudio Se ha usado tanto el manual de usuario y RAPID [11], [21] como video-tutoriales [16].

### 6.2.1 Practica nº 0: Preparación de la estación.

En esta primera práctica denominada 0 se explicará cómo se inicializa una estación, a partir de esta se realizarán las demás prácticas. En primer lugar, se debe abrir el programa RobotStudio, se puede ver su icono en la Figura 6.42 (La versión con la cual se ha realizado este trabajo es la versión 6.04)


Figura 6.42 Icono RobotStudio.

Ahora se debe seleccionar la pestaña de 'Solución con estación y controlador del robot' esto creará una carpeta en la que se guardarán todos los elementos de la estación Figura 6.43, Si se elige otra opción es muy posible que el alumno tenga problemas en el futuro debido a que los ordenadores de la Universidad borran datos al finalizar una sesión.

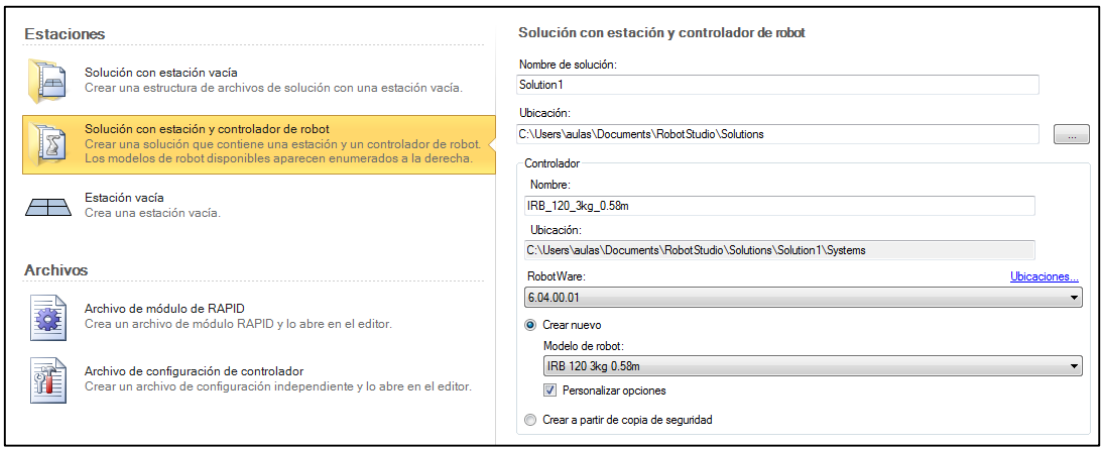


Figura 6.43 Página de inicio del programa RobotStudio.

Es importante seleccionar la pestaña seleccionada en la imagen, abrirá un menú de personalización de robot que es indispensable a la hora de crear una estación en la que se podrá añadir una “Lógica de estación”.

El usuario deberá decidir el modelo de robot que se utilizará en la estación en este momento y en este trabajo será el modelo IRB 120 (Figura 6.44 a). Hecho esto se procede a crear la estación.

Como se ha seleccionado la pestaña de personalización del controlador se mostrará el siguiente el menú comentado con anterioridad (Figura 6.44 b) y se ha de seleccionar la opción DeviceNet dentro de Industrial Nertworks.

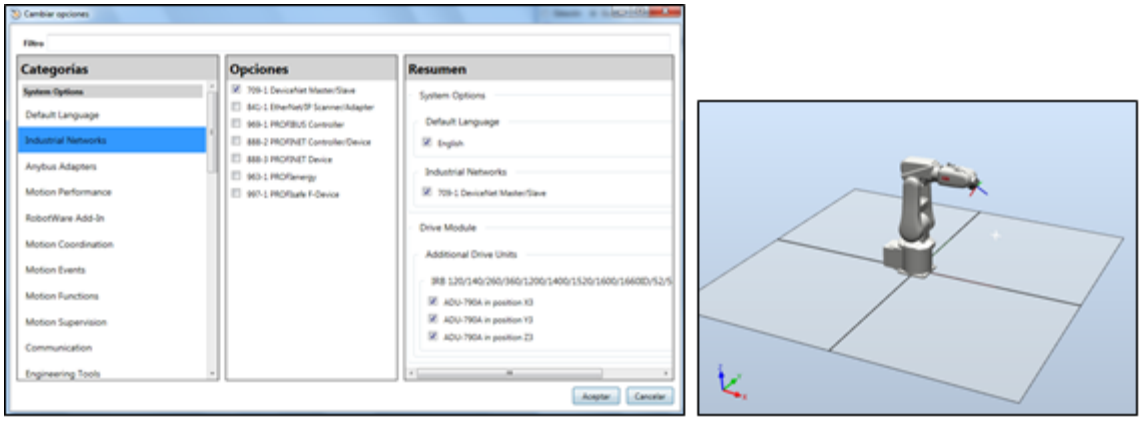


Figura 6.44 a) Página de inicio del programa RobotStudio. b) IRB 120 virtual.

Ahora se debe empezar a recrear la mesa de trabajo usada en el laboratorio, para ello previamente se ha creado un modelo 3D de la mesa, si bien se observa que no es igual, las medidas entre los planos y las distancias entre los mismo si lo son.

Se seleccionará la ventana de importar geometría, visible en la Figura 6.45, y seleccionar el archivo Mesa.sat. Este formato de archivo se crea mediante el comando ACISOUT desde el software AutoCAD.

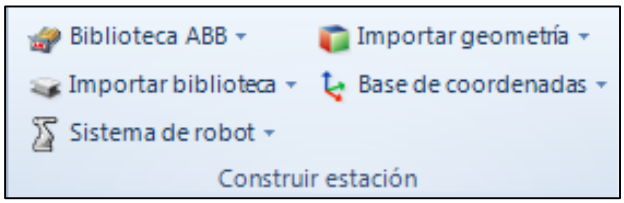


Figura 6.45 Menú RobotStudio, Pestaña Inicio.

Ya cargada la mesa se nos presenta el primer obstáculo, que es la ubicación del objeto Figura 6.46.

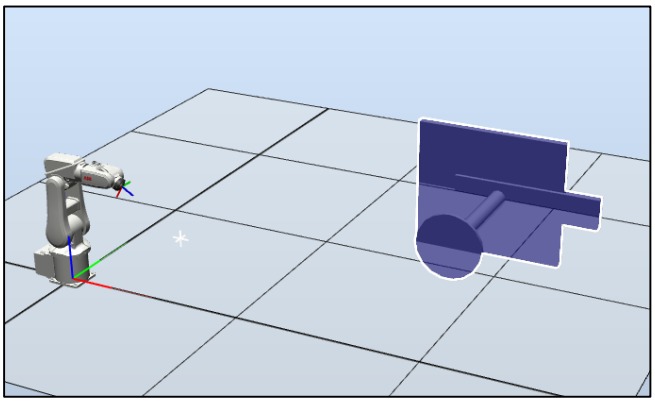

Figura 6.46 Posición Mesa.sat inicial.

Para ubicar un objeto en RobotStudio se hará clic derecho en el modelo y se hará uso de las herramientas para situar objetos Figura 6.47.

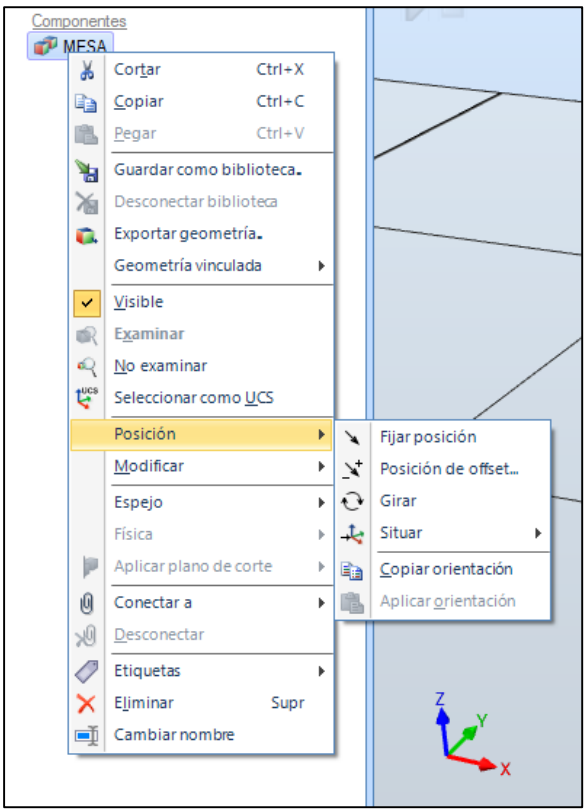


Figura 6.47 Herramientas, situar objetos.

Las herramientas para mover objetos son [11]:

- Fijar posición: Establece la posición de una pieza con respecto el sistema de coordenadas especificado.

- Posición offset: Se seleccionará el eje de coordenadas a tener como referencia y se podrá añadir o sustraer una determinada distancia o ángulo para modificar la posición

- Girar: Se seleccionará un punto de referencia para el giro, seguidamente se dará un valor del ángulo de giro habiendo elegido el eje sobre el cual se hará mediante una pestaña para 'X', 'Y' y 'Z' respectivamente.

- Situar:
    - 1 punto: Seleccionándolo se tendrá que elegir un punto de origen con respecto al eje de coordenadas local del objeto y uno de destino, esto hará que el elemento se mueva desde una posición a otra sin variar la orientación del objeto. Si se quiere cambiar la orientación se debe especificar rellenando las pestañas de orientación.
    - 2 puntos: Similar al de un punto, pero tomando dos orígenes y dos destinos.

    - 3 puntos: El objeto se moverá para coincidir con el primer punto y a continuación girará para coincidir con el tercer punto.

    - Base de coordenada: De una posición a una posición de objeto o base de coordenadas, cambiando simultáneamente la orientación del objeto de acuerdo con la orientación de la base de coordenadas. La posición del objeto cambia de acuerdo con la orientación del sistema de coordenadas del punto de destino.

    - 2 bases de coordenadas: Realiza el movimiento de una base de coordenadas de referencia a otra

En el caso particular de la mesa mostrada primero se girará para darle la orientación deseada, esta es, en la que el rectángulo de menor tamaño será la base del robot. Después con la orden de situar en un punto se seleccionará el centro del rectángulo antes mencionado y se dejarán las coordenadas destino como 0, 0,0.

RobotStudio dispone de herramientas muy útiles para seleccionar puntos dentro de un modelo Figura 6.48:

Figura 6.48 Herramientas de selección de puntos.

Empezando por la izquierda tenemos [6]:

- Ajustar a objetos, ajusta el cursor al punto central, medio o final más cercano.
- Ajustar a centro, ajusta el cursor al punto central.
- Ajustar a línea media, ajusta el cursor a la línea media.
- Ajustar a final, ajusta el cursor al punto final o a la esquina que esté más cerca.
- Ajustar a arista, ajusta el cursor a los puntos de arista.
- Ajustar a gravedad, ajusta el cursor al centro de gravedad.
- Ajustar a origen local, ajusta el cursor al origen local de un objeto.
- Ajustar a cuadricula, ajusta el cursor a los puntos de cuadrícula UCS.

Los que más usados en este trabajo son los dos primeros.

Ahora solo queda añadir la herramienta al brazo, pero antes hay que incluir un accesorio rectangular, visible en la Figura 6.49 para que las medidas coincidan con la realidad de laboratorio.

Figura 6.49 Accesorio brazo-pinza.

Este accesorio ha sido modificado para hacer más sencillo su acoplamiento con la muñeca del robot.

Se le ha añadido además una hendidura en un extremo, estas tienen como propósito ser aristas que permitan una selección más sencilla cuando se deba de añadir la herramienta pinza.

Se actuará como se hizo con la mesa, exportar el modelo, modificar la orientación y por último usar la herramienta situar para colocar el accesorio en donde le corresponde. Para hacer esta tarea más sencilla se hará uso del control que se tiene de los ejes del robot para modificar la postura predefinida (Figura 6.50 a) a una que deje la muñeca en una posición completamente horizontal (Figura 6.50 b).

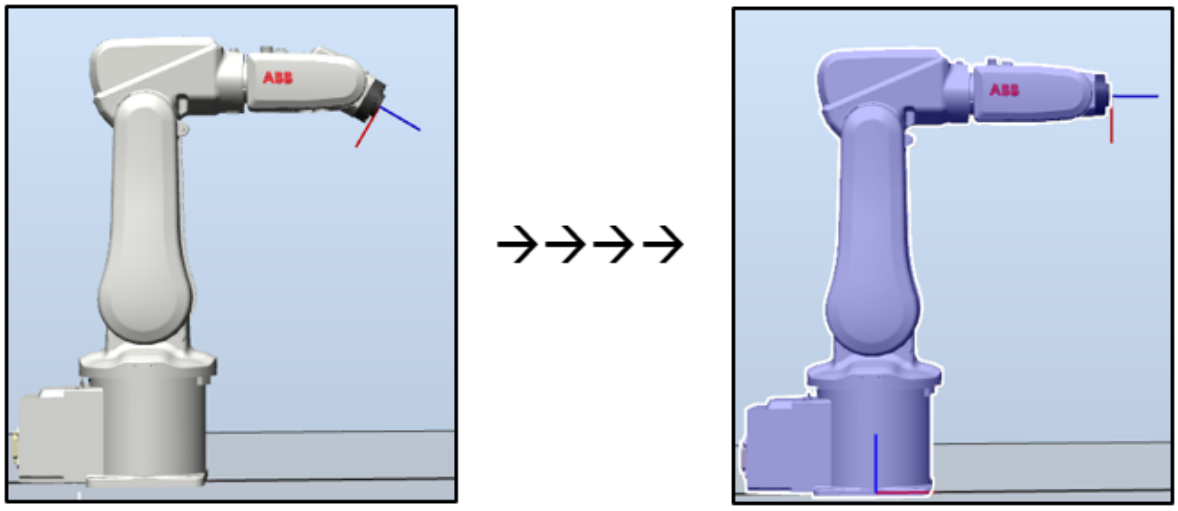

Figura 6.50 a) Brazo antes del movimiento de ejes. b) Brazo después del movimiento de ejes.

Para realizar este giro de ejes se seleccionará el robot en la pestaña de diseño y pulsando el botón derecho del ratón en el mismo, se abrirá un menú en el cual podrá observarse la opción, “ejes del mecanismo” (Figura 6.51).

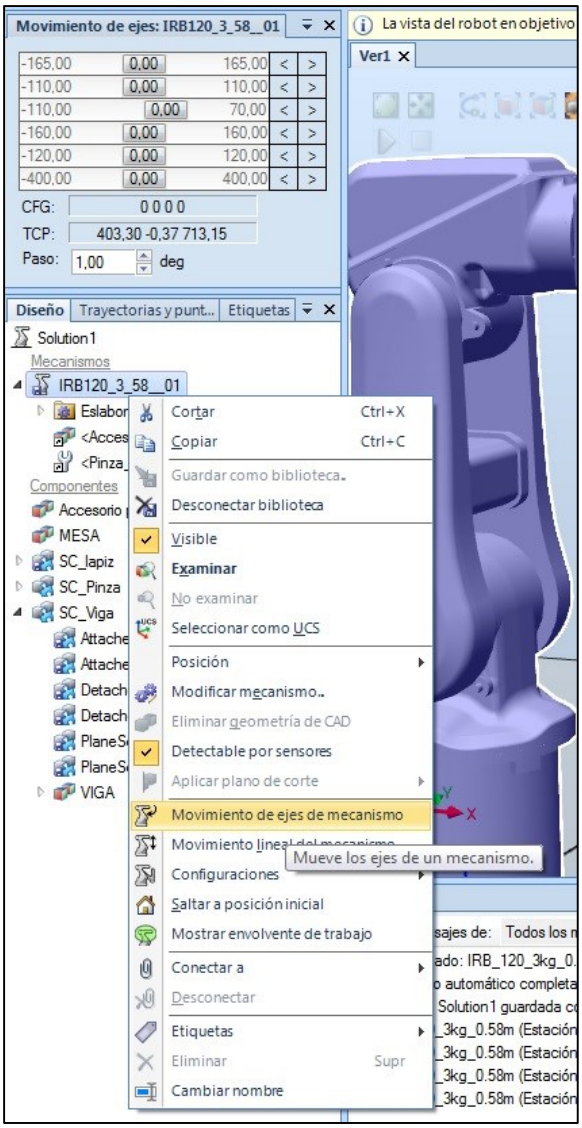


Figura 6.51 Ventanas de opciones del brazo.

En esta nueva pestaña se tiene control de los ejes de manera exacta. Para que el robot tome la postura horizontal deseada se tienen que dar valores 0 a todos los ejes. Realizada esta operación se podrá colocar el accesorio sin problema.

Para que el robot pueda moverse y que además el accesorio (y en el futuro las herramientas) se mueva con el brazo se debe usar la orden “conectar a.…”. La orden puede verse al hacer clic derecho en el modelo del accesorio, se nos darán varias opciones, el usuario deberá seleccionar a IRB 120, como es observa en la Figura 6.52.

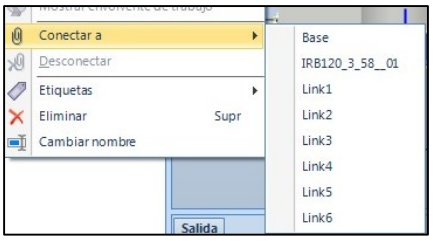


Figura 6.52 Opción "Conectar a...".

El siguiente paso es hacer lo propio con el modelo de la pinza, pero a diferencia de la mesa y el accesorio la pinza es una herramienta, no un modelo. Ya se ha creado un llamado "componente inteligente" de la pinza, dándole propiedades propias de una herramienta, para usar este elemento tenemos que importarlo desde la biblioteca (Figura 6.53).

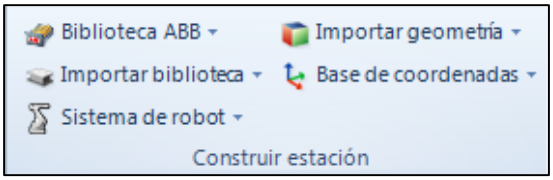


Figura 6.53 Menú Construir estación.

Como no es un elemento del grupo ABB no se encontrará en la carpeta que saldrán por definición, solo hay que buscar el fichero llamado SC_Pinza. Este fichero fue creado en apartados anteriores y el usuario tendrá acceso a él de manera directa.

Importando el archivo disponible SC_Pinza se mostrará el modelo de la Figura 6.54.

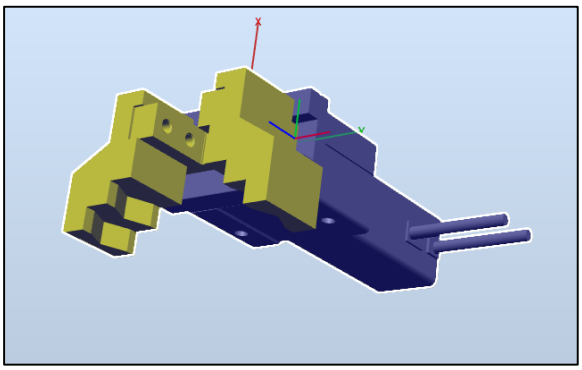

Figura 6.54 Pinza importada al programa.

Para este robot, se dispone de unas mordazas de agarre genérico de superficie plana con dos hendiduras de carácter triangular, añadidas a los dedos mediante dos tornillos, de esta forma se aumenta el rango de piezas a manipular, además de mejorar el agarre.

Estas mordazas son visibles en la Figura 6.54. Con la finalidad de facilitar posibles modificaciones en un futuro se ha preparado la pinza sin estos accesorios.

Para el posicionamiento de la pinza primero se cambiará su orientación para coincidir con la mostrada en la imagen Figura 6.55, después de situar con dos puntos, los puntos origen son los mostrados en la imagen Figura 6.56 a, los de destino serán las aristas anteriormente mencionadas del elemento accesorio Figura 6.56 b.

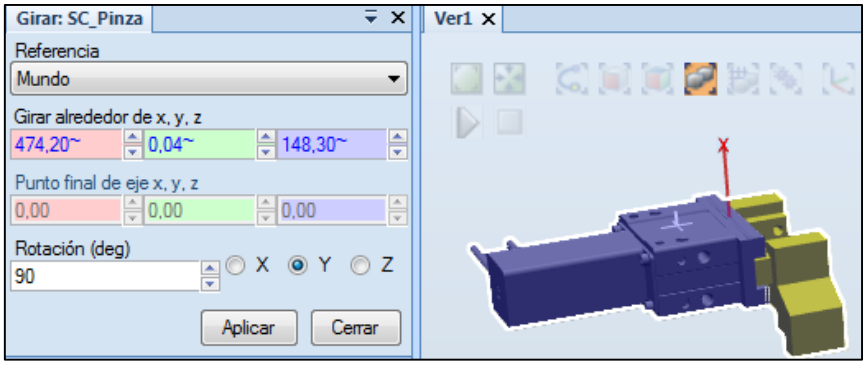


Figura 6.55 Orientación de la pinza.

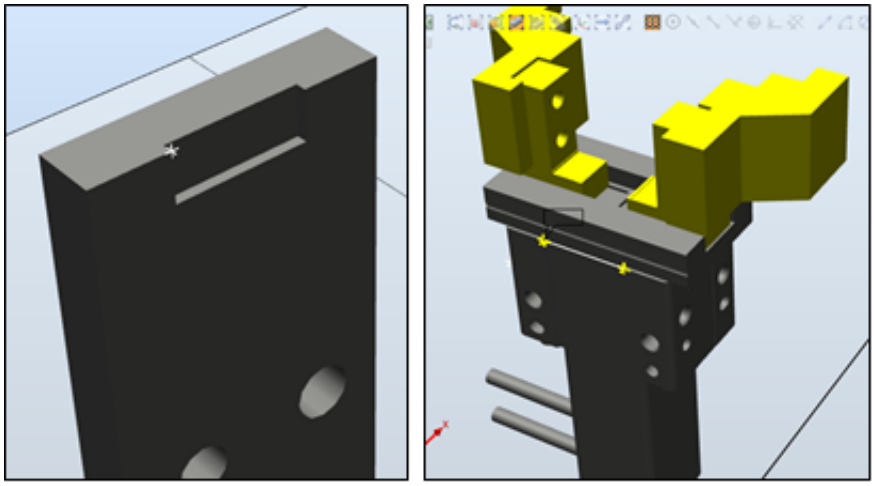

Figura 6.56 a) Puntos de unión en el accesorio brazo-pinza. b) Puntos de unión en la herramienta pinza.

Para finalizar se conectará la herramienta al robot, pero hay que tener en cuenta que el elemento que hay que conectar es la herramienta, no el componente inteligente. Si no se hiciera así el software no detectará a la herramienta y no se podrá seleccionar en la pestaña de selección de herramientas dentro del menú parámetros (Figura 6.57 a). Con este último paso se debería tener una estación con el aspecto visible en la Figura 6.57 b.

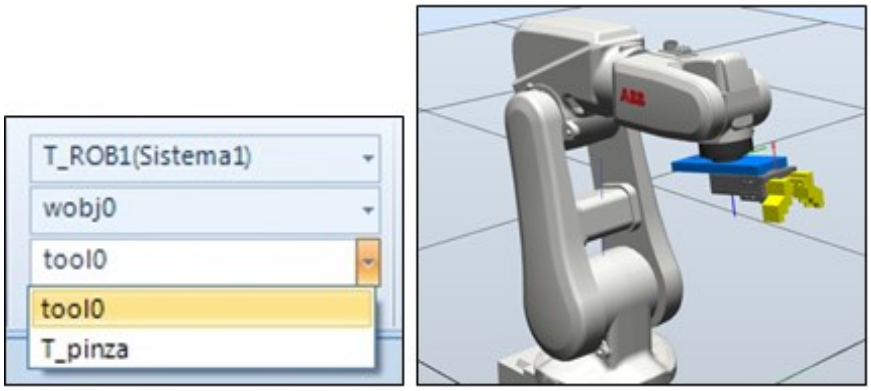


Figura 6.57 a) Selección de Pinza como herramienta. b) Brazo y herramienta unidos.

Para poder visualizar mejor los elementos se han cambiado de color usando la opción de "cambiar color" dentro de la pestaña de modificar Figura 6.58.

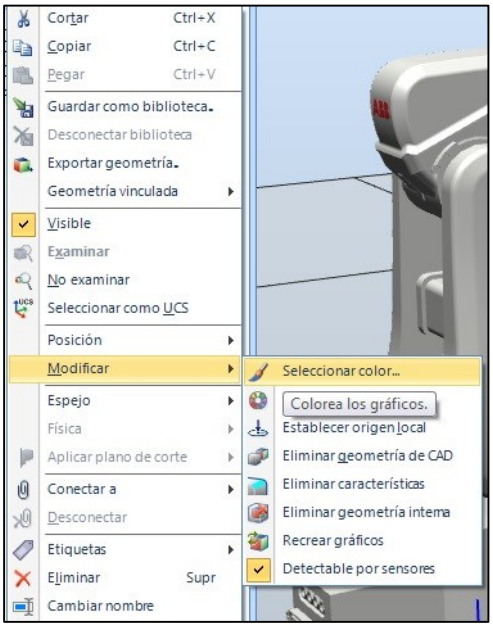


Figura 6.58 Opción cambiar color de geometría.

Este último paso completa la práctica 0, que es el paso inicial que se debería de hacer en caso de querer simular cualquiera de las siguientes prácticas.

Sin embargo, esta estación no es funcional, para completarla hay que añadirle las señales de entradas y salidas necesarias para poder configurar una lógica de estación, esto permitirá al robot manejar las señales de entrada y salida de los componentes inteligentes.

### 6.2.2 Práctica nº1: Dibujar en plano.

Esta es la primera práctica funcional que se va a realizar en este trabajo y en ella se simulará el movimiento de un conjunto de piezas que sostienen un rotulador, siendo el resultado de estos movimientos la escritura de nuestras iniciales en un folio de papel previamente colocado en la mesa de trabajo. En el ejemplo que se va a detallar las iniciales son J.A.

Como primer paso se deberá tener una estación preparada de la forma que se explicó en el apartado Práctica nº0, ver Figura 6.59 a.

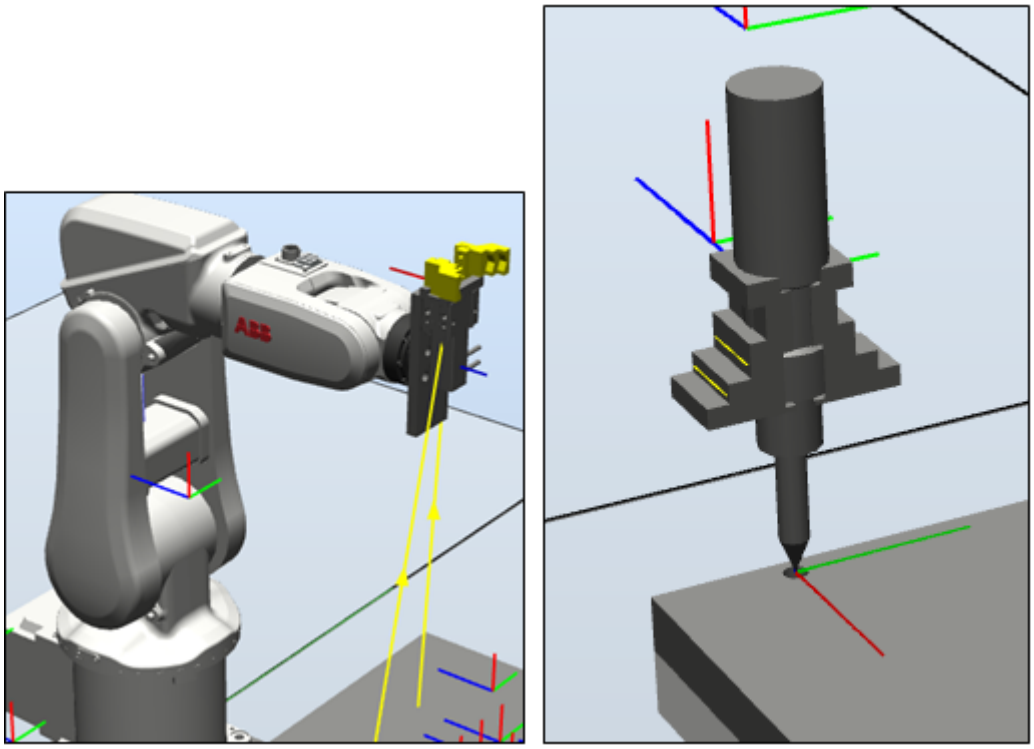

Figura 6.59 a) Brazo configurado. b) CI Lápiz en su posición de inicio.

Preparado ya el brazo robot con la herramienta Pinza lista para su uso (Figura 6.59 b) se deberá importar el archivo SC_Lapiz, este archivo ha sido creado con anterioridad.

Debido a la naturaleza del elemento "Attacher" dentro del componente inteligente, cada vez que se importa un componente inteligente a la estación se debe de indicar el "accesorio pinza-brazo". Para ello se seleccionará el componente inteligente dentro de la pestaña de diseño en el menú "Inicio", tal y como se muestra en la Figura 6.60.

Si no es posible seleccionar “Editar Componente”, se debe a que el archivo se encuentra guardado en la biblioteca y no permite cambio dentro del mismo. Para poder modificar esto primero se debe de desconectar el componente inteligente de la biblioteca pulsado “Desconectar biblioteca”.

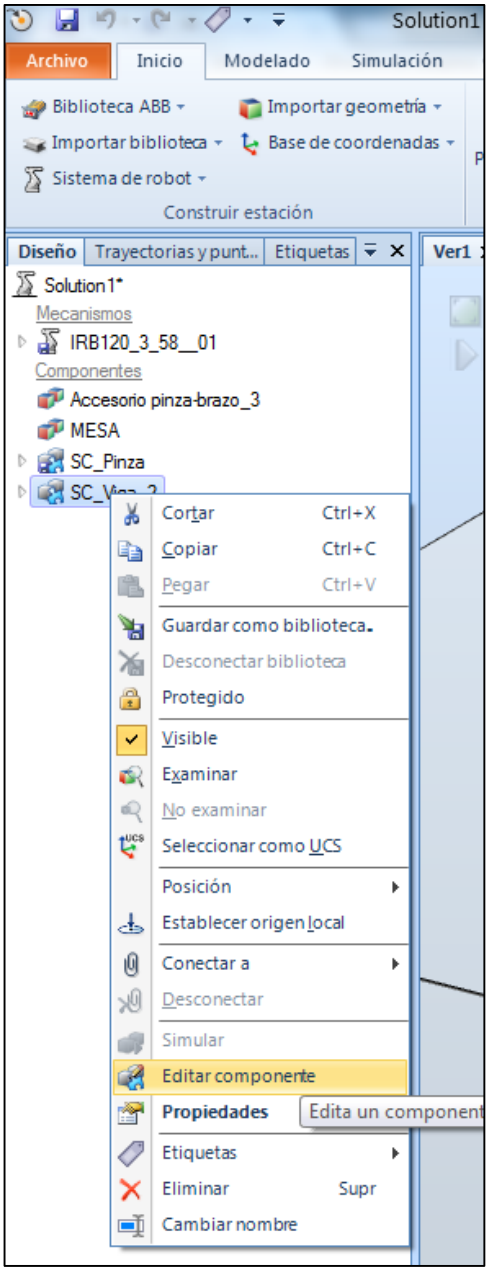


Figura 6.60 Selección editar componente.

Cuando se pulse “Editar Componente” se verá la misma ventana que cuando se creó el componente, solo hay que ir a la pestaña de diseño y añadir el accesorio en el elemento “Get Parent”.

Este es otra de las desventajas que se han encontrado al ubicar los sensores en los elementos cogidos y no en la pinza, si los sensores se encontraran en los dedos de la pinza, nunca se perdería la propiedad de “accesorio pinza-brazo” ya que, en un principio, nunca se eliminaría la pinza de la estación, lo que produce este problema.

Es muy probable que el propio archivo al ser importado a la estación se encuentre en la posición de inicio, se debe a que al guardar el archivo se guardan también datos como la posición del objeto.

Esta posición es en la que la punta del lápiz está en contacto con el centro del punto de referencia realizado en la mesa, es decir el punto (0,0,0) del plano de trabajo. En caso de no estar en esta posición se usarán las herramientas disponibles de movimiento explicadas en la práctica nº0.

Colocado ya el componente inteligente se procederá a crear un plano de trabajo, este plano es donde se determinarán los puntos que forman esta práctica, y para crear un plano de trabajo se debe ir a la sección de la pantalla llamada “Programación de trayectorias” (Figura 6.61).

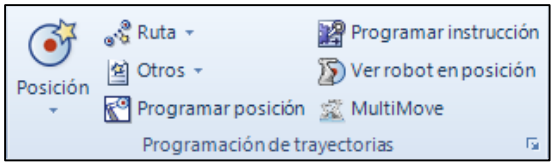


Figura 6.61 Menú Programación de trayectorias.

El pequeño triángulo visible en varias de las opciones destaca que existe un submenú, se debe seleccionar el submenú de “Otros”, desde donde se verán las opciones mostradas en la Figura 6.62.

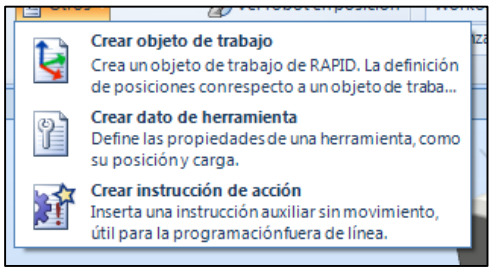


Figura 6.62 Opciones de creado en programación de trayectorias.

La primera opción es la que se necesita, como indica la breve explicación visible dada por el programa. Al seleccionar crear objeto de trabajo, se abrirá la ventana emergente

visible en la Figura 6.63 que permite la configuración de un plano de trabajo. En el caso de este trabajo lo único que se ha modificado es el nombre del mismo, ya que la posición del origen del plano se modificará más tarde usando comando de posicionamiento.

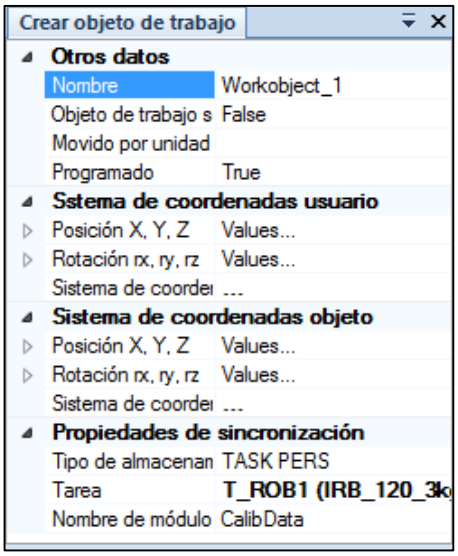


Figura 6.63 Configuración de plano de trabajo.

La posición inicial de un plano de trabajo es el punto de origen predeterminado del programa. Se recuerda que este punto se eligió como la base del robot (Figura 6.64).

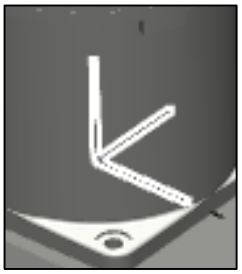

Figura 6.64 Punto de creación de planos pre-definido.

Como se ha mencionado anteriormente el origen del plano de trabajo se encuentra en el centro del orificio de la mesa, con el uso de los comandos de posicionamiento se trasladará el plano de trabajo para que coincidan en este punto Se tiene que además hacer que los ejes X, Y, Z coincida con los representados en la Figura 6.65.

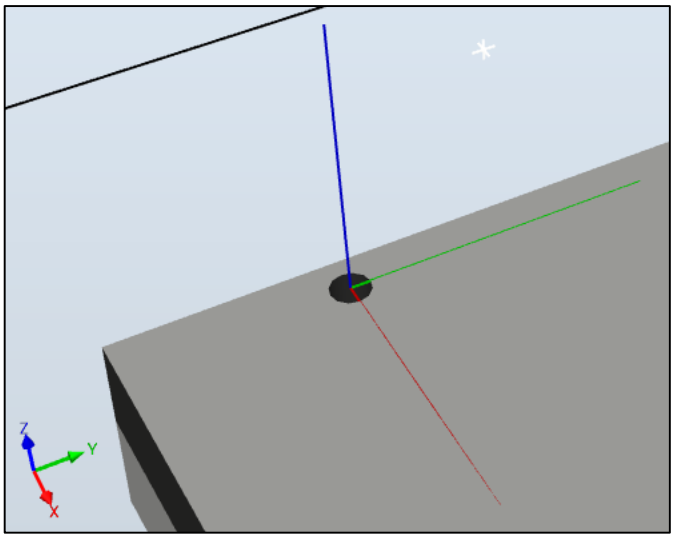


Figura 6.65 Posición del plano de trabajo definitivo.

Hasta ahora no se ha hablado del código de colores que usa RobotStudio para definir los ejes, ya que es visible en todo momento en la esquina inferior izquierda de la visualización del brazo. Este código es el siguiente, el rojo representa el eje X, el verde representa el eje Y, y el azul representa el eje Z.

Realizado ya este paso se deben determinar los puntos necesarios para poder simular esta práctica, primero se explicará cómo añadir un punto al plano, para luego explicar los distintos puntos que hay que añadir en esta práctica en particular y la razón por la que se han añadido.

Para definir un punto en un plano en concreto hay que antes activar el plano, hay dos maneras de activar un plano.

La primera consiste en ir al menú visible en la parte superior de la pantalla Figura 6.66 a, en este menú, se puede seleccionar el plano de trabajo con la segunda opción.

La segunda forma de activar un plano consiste en activarlo desde el propio plano. Para ello primero hay que seleccionar el plano de trabajo, que se encuentra en a la parte izquierda de la pantalla y en “trayectoria y puntos”, se pulsará el botón derecho del ratón en el plano deseado abriendo el menú que se muestra en la Figura 6.66 b, siendo la primera opción disponible la de activar el plano.

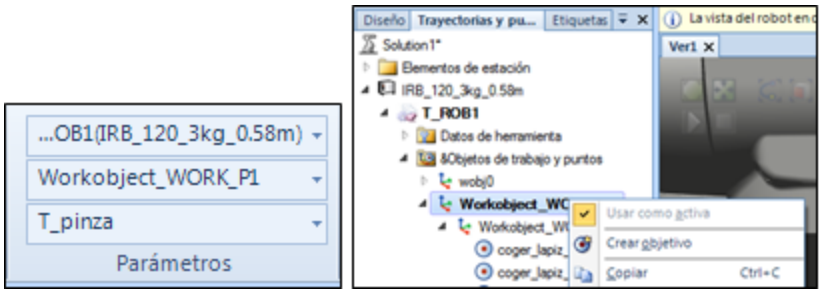


Figura 6.66 Menú de Parámetros. b) Activación del plano de trabajo.

En la Figura 6.66 b también es visible la opción de "Crear Objetivo", su selección abrirá la ventana emergente con la que crearemos los puntos (Fig [6.80]), otra manera de llegar a esta ventana es pulsando "Posición" en el menú "Programación de trayectorias".

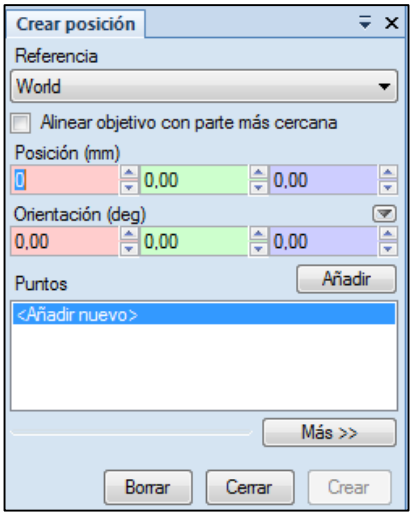


Figura 6.67 Ventana de configuración de objetivo.

La referencia determina respecto de que plano de trabajo se están definiendo los puntos, se debe seleccionar el plano de trabajo, para que la posición (0, 0, 0) sea el centro del orificio de la mesa.

Por ejemplo, para crear un punto a 200 mm de distancia en X e Y en el plano de trabajo, la ventana visible en la Figura 6.67 debe de quedar con un valor de 200 en la pestaña roja, la de verde 200 y la azul 0. Se pulsará Añadir, lo que permite pre-visualizar el punto (Figura 6.68 a), para completar la creación del punto se debe de pulsar crear (Figura 6.68 b).

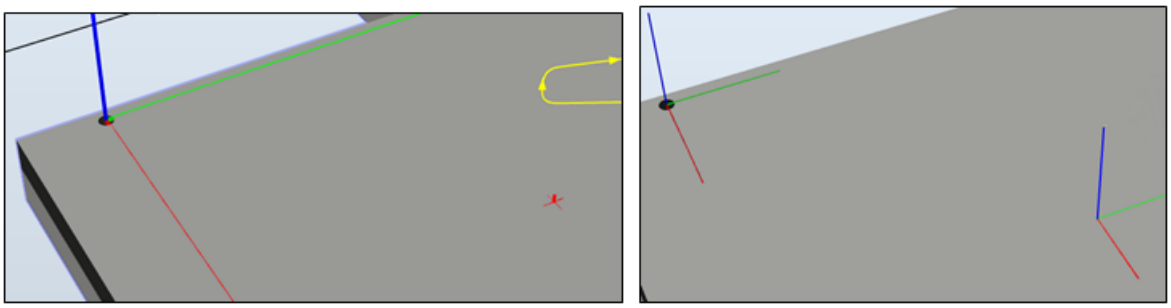

Figura 6.68 a) Pre-visualización de punto b) Punto creado.

Si se deseara modificar l aposición de un punto ya creado solo hay que usar las herramientas de posicionamiento.

A continuación, se comentarán los grupos de puntos creados y más importantes, como se ha creado el punto de cogida de elementos:

- Puntos de aproximación.

Estos puntos son todos los que determinan pasos intermedios, como aproximaciones al lápiz o a la "hoja de papel" y el punto de reposo.

Son como su nombre indica puntos de aproximación para tener un control mayor de los movimientos del Robot.

- Puntos de toma de Lápiz.

En realidad, es solo un punto, siendo de gran importancia, hay que tomar varias medidas en los modelos 3D, primero en la herramienta pinza, y después en la pieza que se quiera coger.

Estas medidas son necesarias para determinar que distancias ha de tenerse en cuenta para que el punto configurado tenga como resultado el que se observa en la Figura 6.69.

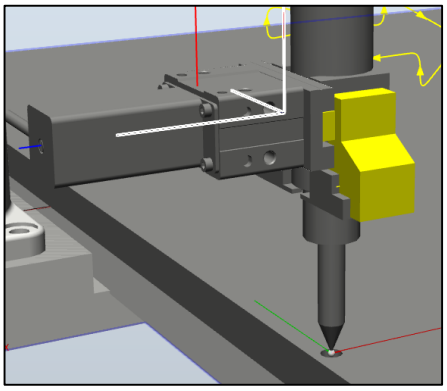

Figura 6.69 Pinza colocada.

En este caso todos los planos del accesorio agarre toman a la pieza, sin embargo, esto no se cumple en las demás prácticas, cada pareja de planos del accesorio agarre tienen distintas distancias máximas y mínimas distintas, distintas piezas se toman con distintos planos (Figura 6.70).

Piezas en la que la pinza deba abrirse entre 81mm y 67mm se toman con la última área de agarre y piezas entre 63 mm y 49 con la que se encuentra en medio.

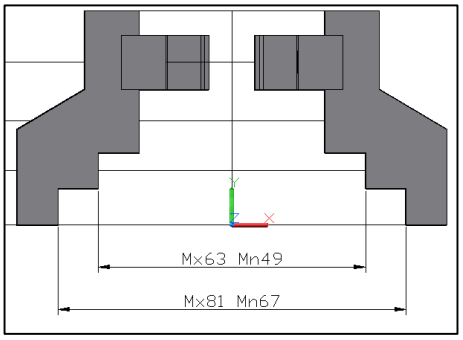


Figura 6.70 Esquema de los accesorios de agarre.

Observando el modelo 3D de los dedos con el accesorio agarre se han determinado varias medidas de interés, 21,5mm, 36mm y 24mm.

La medida de 21,5mm proviene de medir la distancia menor entre el punto dato de herramienta con el plano en el que se encuentran los puntos medios del plano de agarre, esta medida es visible en la Figura 6.71.

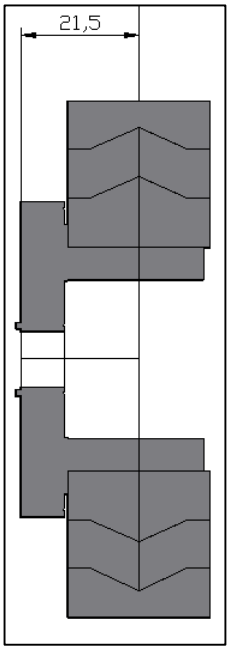


Figura 6.71 Vista de agarre 1.

La medida de 36 mm corresponde con la menor medidas posibles entre el punto dato de herramienta con el plano formado por los puntos que conforman la cara más alejada del dato herramienta (Figura 6.72).

La medida de 24 mm es similar, es la medida más cercana entre el punto medio de la segunda área de agarre.

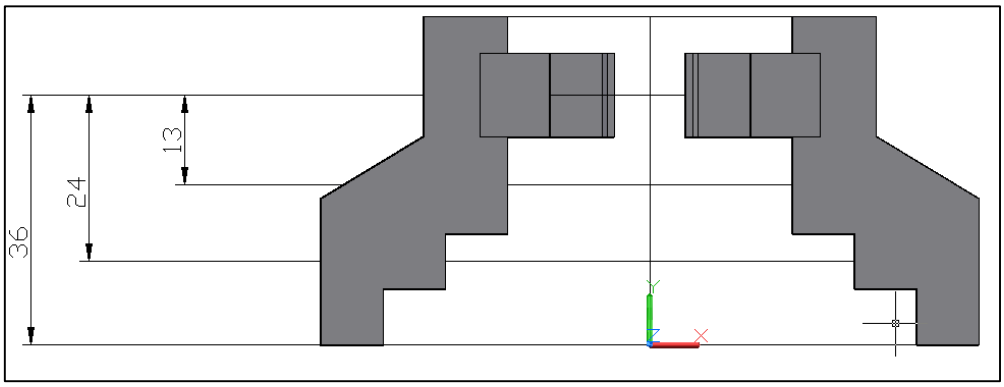


Figura 6.72 Vista de agarre 2.

Conociendo la relación de estas medidas se podrá decir que en caso de que se agarre con la última cara del accesorio la altura que se debe de agregar al punto de interés es de 36mm-7,9mm, (La medida de 7,9 proviene de los 8mm de altura del área de agarre, de esta manera no hay contacto no deseado entre los dedos de la pinza y el objeto).

En caso de que el área de contacto sea la que se encuentra en medio del accesorio agarre, la altura que se debe de agregar es de 20,1 mm en vez de 28,1 mm (36-7,9).

Se puede llegar a esta conclusión observando que 24 mm menos la mitad de 8 mm con el 0.1 mm de seguridad dan ese mismo resultado (24mm-3.9mm=21,5mm).

Sin embargo, este primer caso (el lápiz) es particular, ya que la toma de un punto al que agregar las distancias mencionadas no es de fácil selección, pero realizando medidas se comprueba que la punta de lápiz (cuya selección si es sencilla) dista de la zona de agarre 76mm, pero no es simétrico.

En este caso la medida de 21,5mm no es la correcta, como es sin embargo similar se planteó en un primer momento esa distancia, para luego con una orden offset modificar el punto hasta que se evitara colisión.

Se creará un punto en la punta del lápiz para después usando la herramienta offset añadir en Z; 7 6mm+36 mm y luego en X; 21.5 mm, para poder visualizar cuantos milímetros

de deben de añadir en el eje X (Estos ejes son en referencia al plano Mundo) para evitar choques, se usará la opción dentro del punto de “ver herramienta en posición”.

Esta opción permite pre-visualizar la pinza en el punto, es posible que se deba girar el punto para que la herramienta pinza se encuentre en posición de agarre, lo cual implica que la distancia que se planea añadir se debe de hacer usando como referencia al plano mundo.

” Ver herramienta en posición” se encuentra seleccionando el punto de interés en el menú de “Trayectorias y puntos”, se debe a continuación pulsar el botón derecho del ratón y seleccionar la opción visible en la Figura 6.73.

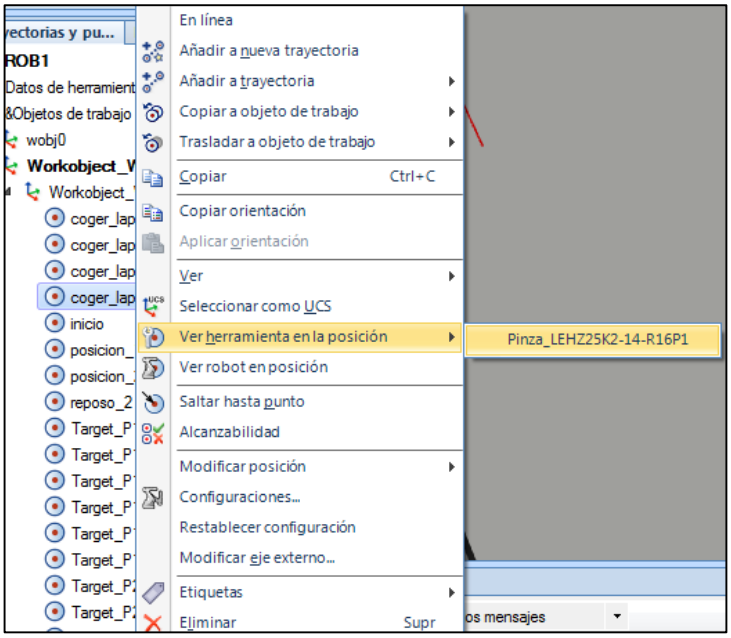


Figura 6.73 Ver herramienta en posición.

En el caso de este trabajo se añadieron 5 mm. Es aconsejable tomar medida de las medidas agregadas, serán necesarias cuando se creen la malla de puntos del folio.

- Puntos de “hoja de papel”.

Este trabajo continúa un trabajo anterior en el que solo se usó la programación gestual disponible en los productos de ABB (Usando el FlexPendant), siendo esta programación gestual es poco práctica a la hora de crear puntos.

Por tanto, para que varios alumnos pudieran programar sus iniciales en una sola clase se creó esta malla, con el uso de los puntos disponibles y comandos offset posibles en la programación mediante FlexPendant.

En el caso de la programación con RobotStudio esta malla pierde su propósito de ahorrar tiempo, ya que, como se ha comprobado, crear puntos en RobotStudio es una tarea rápida y sencilla.

Sin embargo, la creación de esta malla permite practicar la creación de puntos, por lo que se mantuvo en este trabajo.

La malla de puntos se encuentra dentro del área que ocupa un folio en posición horizontal, con la esquina superior izquierda en una posición de X=465mm Y=490 (Figura 6.74).

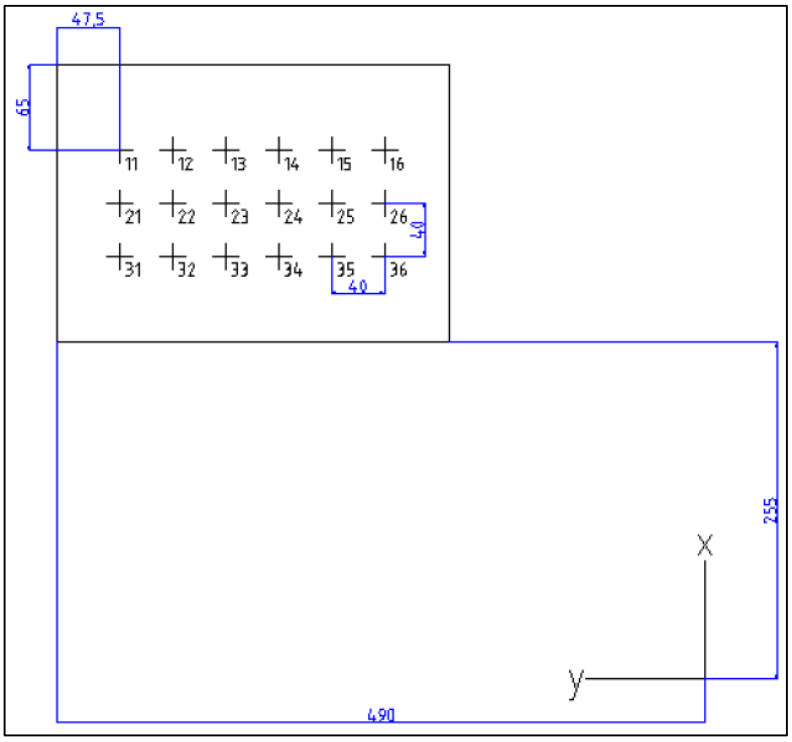


Figura 6.74 Esquema de posiciones en el plano de trabajo P1 [20].

Primero se creará los puntos en el plano de la mesa de trabajo de la forma que se ve en la Fig [6.75], para después modificarlos todos con las medidas realizadas para el agarre del lápiz, también es posible realizar operaciones antes de crear los puntos para crear una malla en la que directamente que haga el cambio para el agarre.

Para facilitar la identificación de los puntos se cambia la nomenclatura que el programa produce automáticamente, esto se hace en la opción “cambiar nombre” visible al pulsar el botón derecho previa selección del punto.

El resultado final tendrá un aspecto similar a la Figura 6.75.

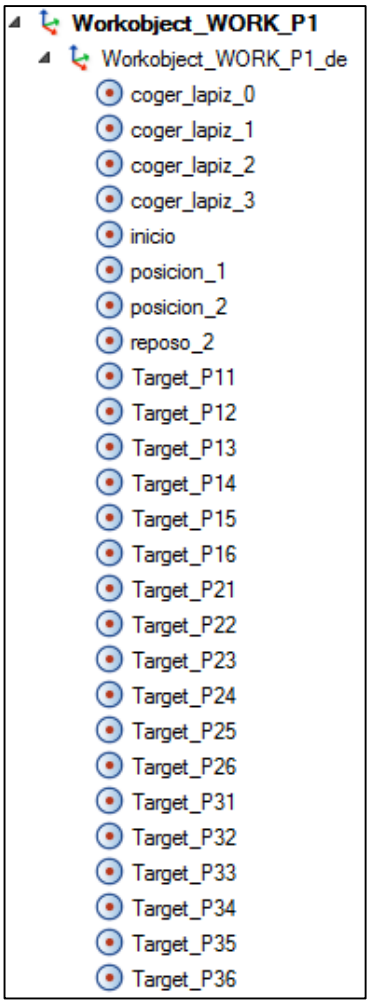


Figura 6.75 Lista de puntos.

Coger_lápiz_0, 1, 2 son aproximaciones desde el punto de reposos hasta el punto de agarre Coger_lápiz_3, Inicio es el punto es donde se empieza la aproximación al folio, y posición_1,2 son puntos cercanos al folio, pero elevados para poder cambiar del movimiento a una inicial a otra, y los Target, son los puntos que forman la malla.

La lista inicialmente creada no tiene el aspecto mostrado, tendrán una imagen de un asterisco, esto indica que el robot no tiene ninguna configuración de las articulaciones asignadas al punto, para asignar al punto solo hay que seleccionar el punto y pulsar el botón derecho del ratón para abrir el menú de opciones en el que será visible “Configuraciones” (Figura 6.76 a), al pulsarlo, una ventana emergente nos permitirá seleccionar una configuración viable, si no se puede pulsar esta opción es posible que ese punto no sea alcanzable en caso de duda, en el mismo menú en el que es visible “Configuraciones” se puede pulsar “alcanzabilidad” (Figura 6.76, b) para comprobar si el punto es alcanzable.

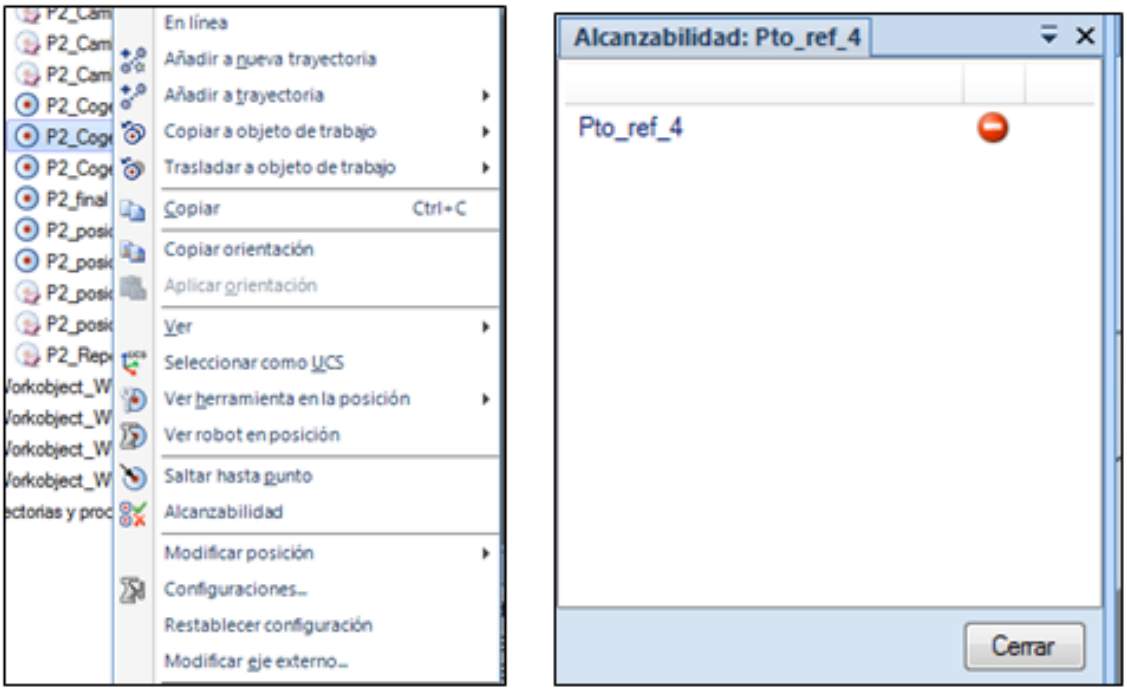


Figura 6.76 a) Menú de un punto b) Alcanzabilidad del punto.

Ahora tenemos una estación con una herramienta con una serie de posibles entradas, y un componente inteligente con unas salidas de entrada y de salida, para que el Robot comprenda estas entradas y salidas hay que configurar el controlador.

Para preparar el controlador debemos ir a la pestaña de Controlador, visible en la parte superior de la pantalla (Figura 6.77).

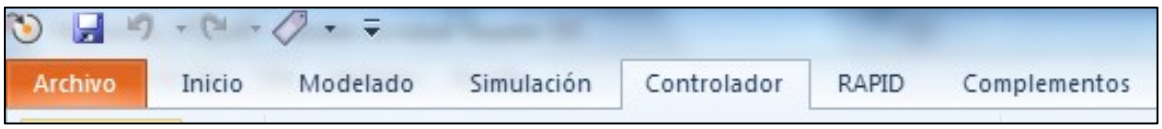


Figura 6.77 Pestaña de menús.

En la parte izquierda de la pantalla solo será visible una pestaña, “controlador” (Figura 6.78), en esta abriremos “Configuration” y pulsaremos “I/O System”, cambiando la visualización del brazo por la de la Figura 6.79.

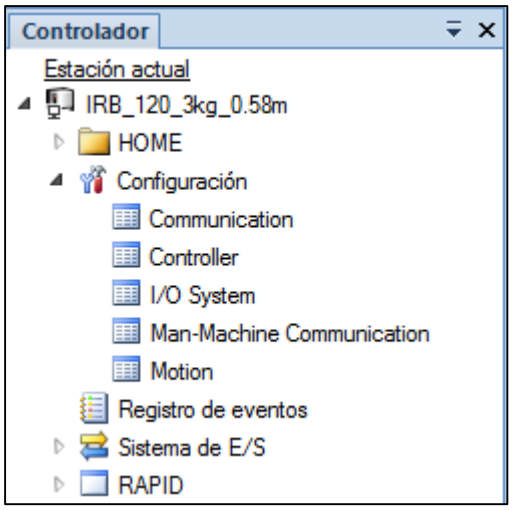


Figura 6.78 Ventana del menú controlador.

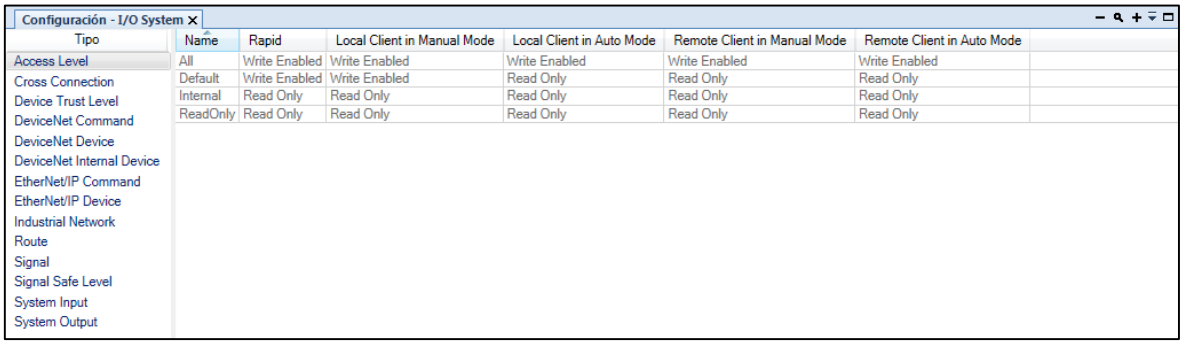


Figura 6.79 Ventana de entradas y salidas del controlador.

Se recuerda que en la práctica nº0 no se agregó un elemento al controlador, se configuró un controlador con la capacidad de tener un “Device Net” (Necesario para la realización del trabajo). Si no se encuentra entre las opciones “DeviceNet Device” significa que no se configuró correctamente el controlador como se instó en la práctica nº0.

Primero hay que configurar un “DeviceNet”, el “DeviceNet” será en donde incluiremos nuestras señales y salidas, para crear uno se debe de ir a “DeviceNet Device” y pulsar el botón derecho del ratón y seleccionar “Nuevo DeviceNet”.

La ventana emergente que aparecerá a continuación es la vista en la Figura 6.80.

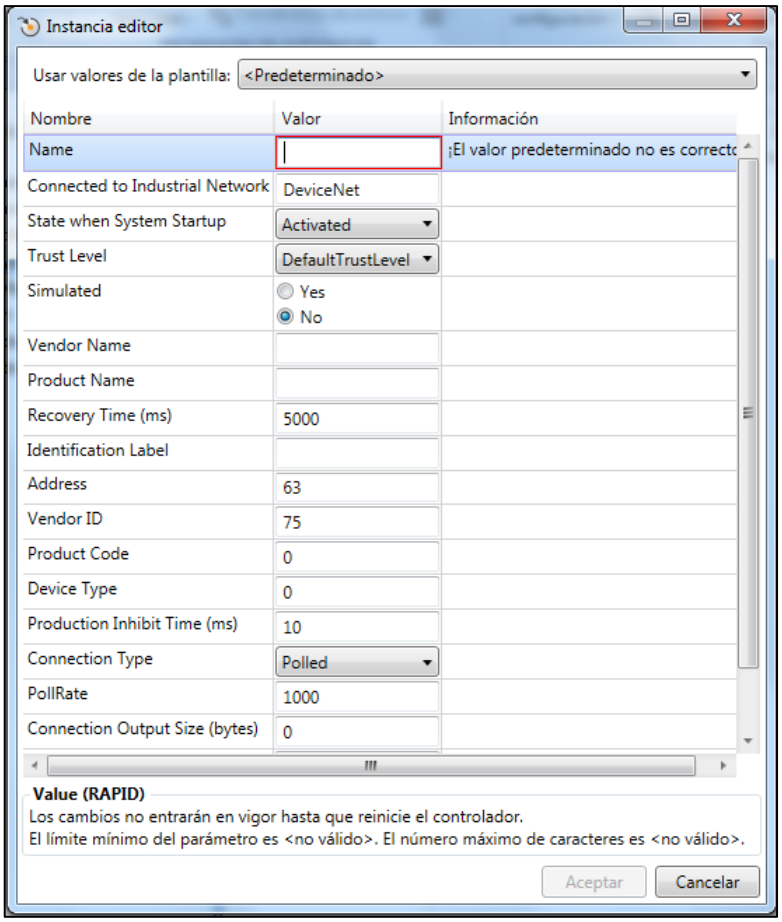


Figura 6.80 Ventana creación de DeviceNet.

El nombre del "DeviceNet" puede ser cualquiera siempre y cuando no empiece con un número (1_nombre no es válido, nombre_1 sí). Los valores usados son los predeterminados, así que al elegir un nombre solo hay que pulsar acepta.

Ya creado el "DeviceNet" se crearán las señales, para crear señales de entradas y salidas al controlador se deberá pulsar en "Signal" visible en la Figura 6.81.

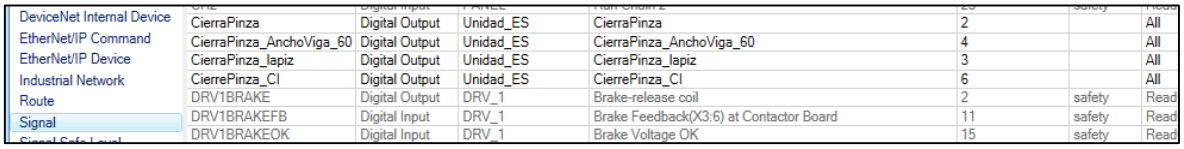


Figura 6.81 Señales en el controlador.

En esta Figura 6.81, se observa varias señales ya creadas, son señales de salida que se corresponden con las que recibe la herramienta.

Para crear una señal se pulsará el botón derecho del ratón previa selección de "Signal", al pulsar Nueva señal aparecerá la ventana emergente Figura 6.82.

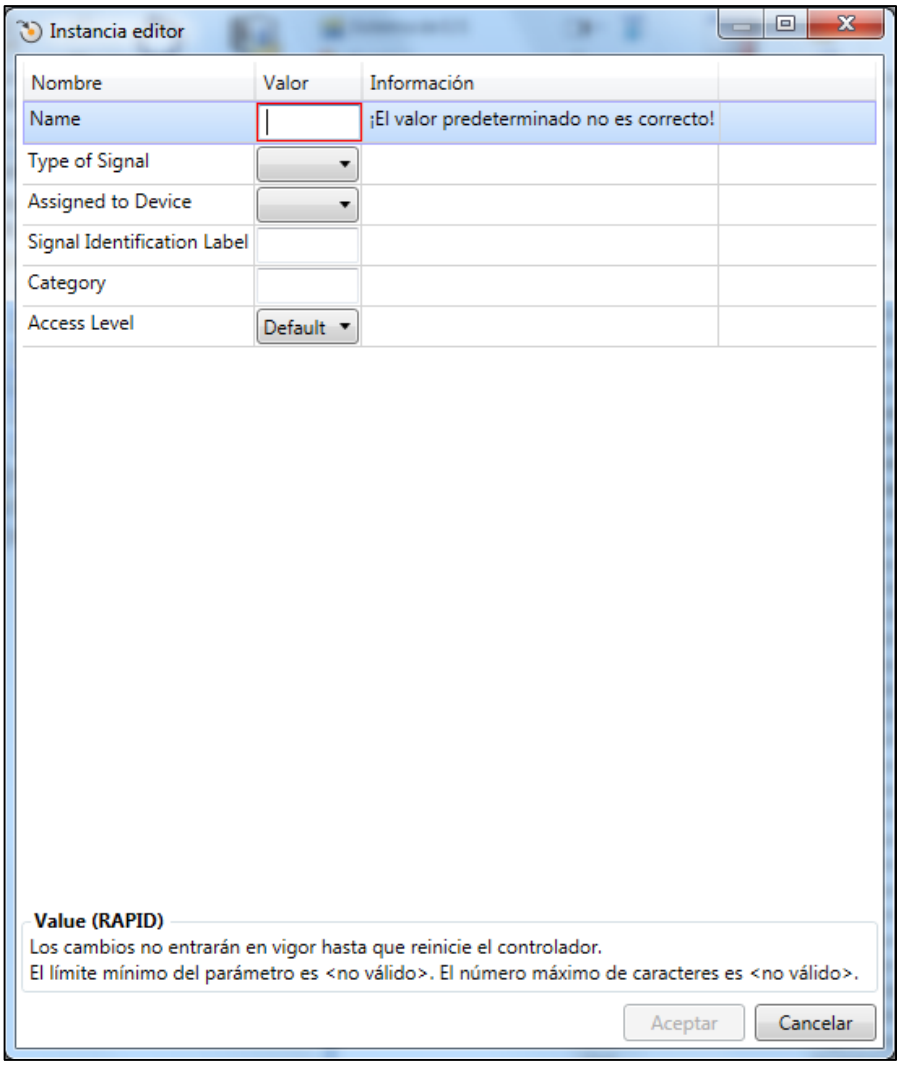


Figura 6.82 Ventana de creación de señales en controlador.

Primero se explicará cómo crear una señal de salida del controlador, que coinciden con las señales de entrada de los componentes inteligente.

Primero se determinará un nombre que identifique la señal de forma clara, luego en "Type of signal" se seleccionará "Digital OutPut" (Se elige esta opción porque la señal que se requiere es 1/0, en caso de que sea de otro tipo la señal la requerida, el tipo de señal deberá ser distinta, en este trabajo no se ha estudiado esta posibilidad porque no ha sido necesario).

En la opción "Assigned to Device" se seleccionará el "DeviceNet" añadido por el usuario.

La selección de estos elementos modifica la ventana, de manera que ahora es como muestra la Figura 6.83.

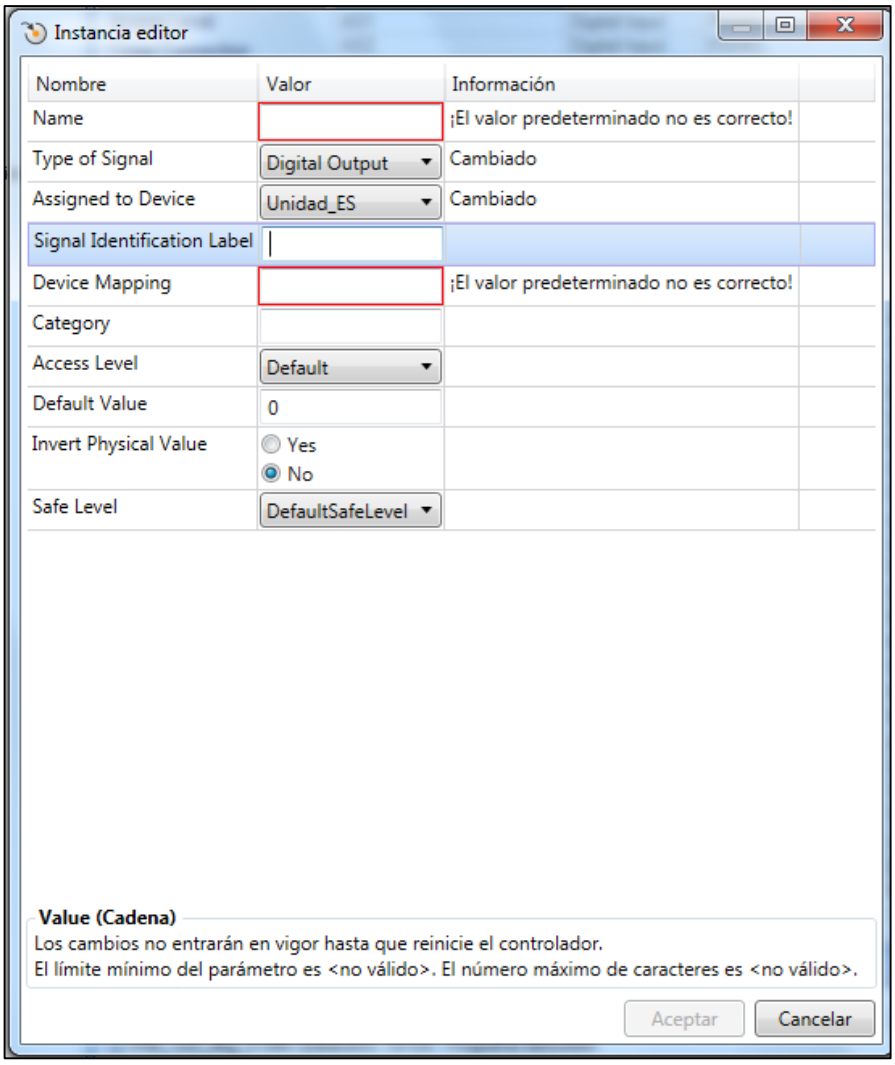


Figura 6.83 Ventana creación de señal de salida en controlador.

"Signal Identification Label" no es necesaria, pero para mantener la máxima homogeneidad en la nomenclatura de señales se le ha puesto el mismo nombre que se ha rellenado con anterioridad.

"Device Mapping" es en cambio muy importante, porque es la identificación que comprueba el software, siendo esta un número, empezando por el 1, para evitar confusión se insta a que se apunte de forma separada las identificaciones numéricas de las señales a medida que se creen.

En "Access Level" se debe cambiar de "default" a "All" y por último se pulsa aceptar.

Como ejemplo de una señal de entrada se muestra la Figura 6.84, en esta figura se observa la señal que se conecta con la orden de apertura para agarres de 80mm de la pinza.

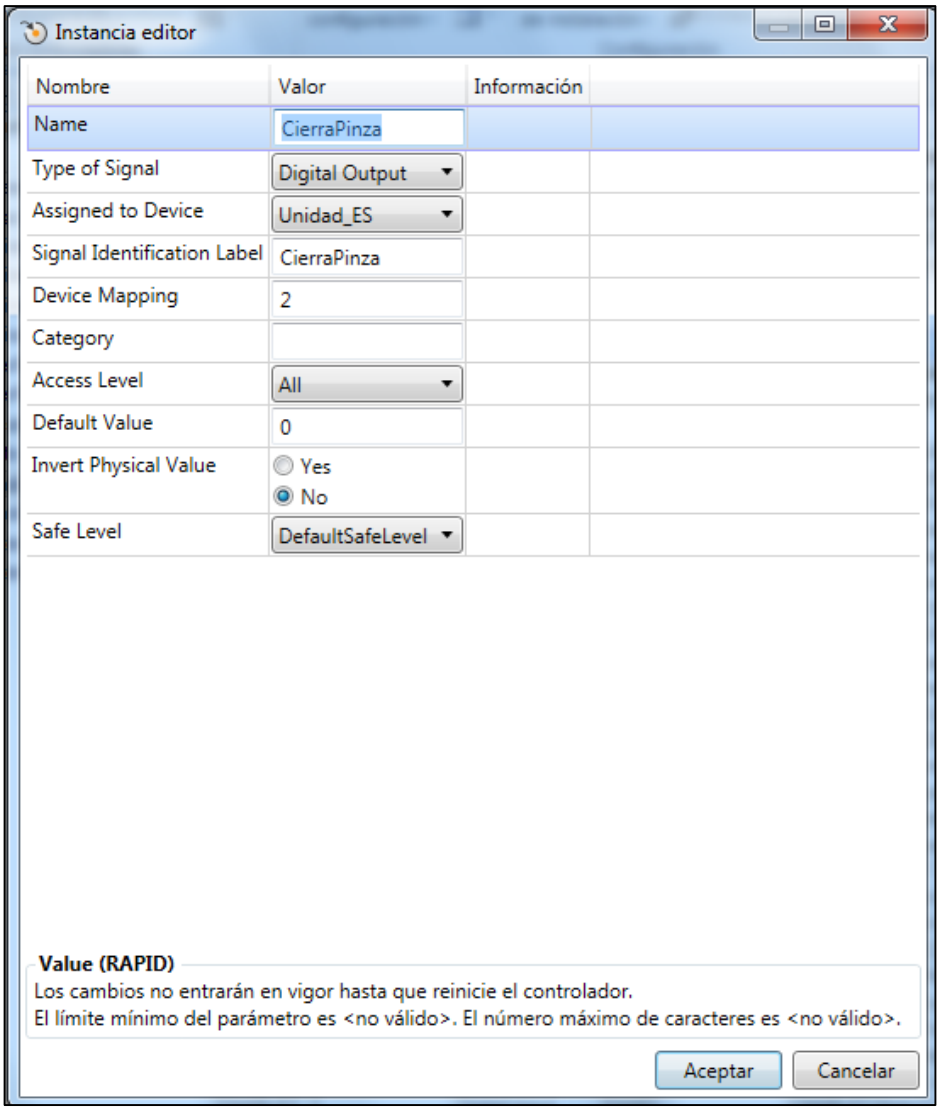


Figura 6.84 Ejemplo de señal en controlador.

Las principales señales de salida del controlador en este trabajo son: La señal de activación de sensores, y las señales de apertura de pinza para distintas posiciones.

También se han tenido que crear otras señales de salida del controlador para otras prácticas, pero se hablará de ellas más adelante.

Explicada ya las señales de salida del controlador pasaremos a las señales de entrada, excepto en la última y penúltima práctica la única señal de entrada del controlador ha sido la señal "presente", no es necesario crear una señal de "presente" para cada elemento que tenga sensores en el trabajo.

Para crear una señal de entrada para el controlador se debe seguir un proceso similar al mencionado anteriormente, con la diferencia de la selección de tipo de señal en "Type of

Signal", en el caso de señales de entada se debe seleccionar "Digital Input" (Como se ha comentado anteriormente, en este trabajo se usan elementos con señales 1/0, si se usara un elemento cuya salida no fuese un 0 o 1 el tipo de señal de entrada sería distinto).

El "Device Mapping" de las entradas y salidas es independiente, lo que implica que hay señal 1 tanto en la entrada como salida del controlador.

Creadas las señales de entrada y salida del controlador pasaremos a unirlas con sus respectivos componentes inteligente, para ello saldremos de la pestaña en la que nos encontramos y pasaremos a la pestaña de Simulación, en esta pestaña encontraremos un menú llamado Configurar (Figura 6.85).

Pulsaremos Lógica de estación, cambiando la ventana de visualización por la ventana mostrada en la Figura 6.86.

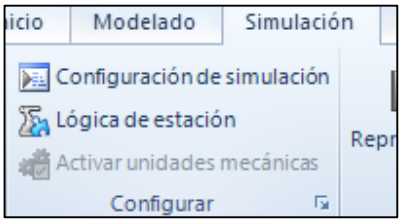


Figura 6.85 Menú configura, ventana simulación.

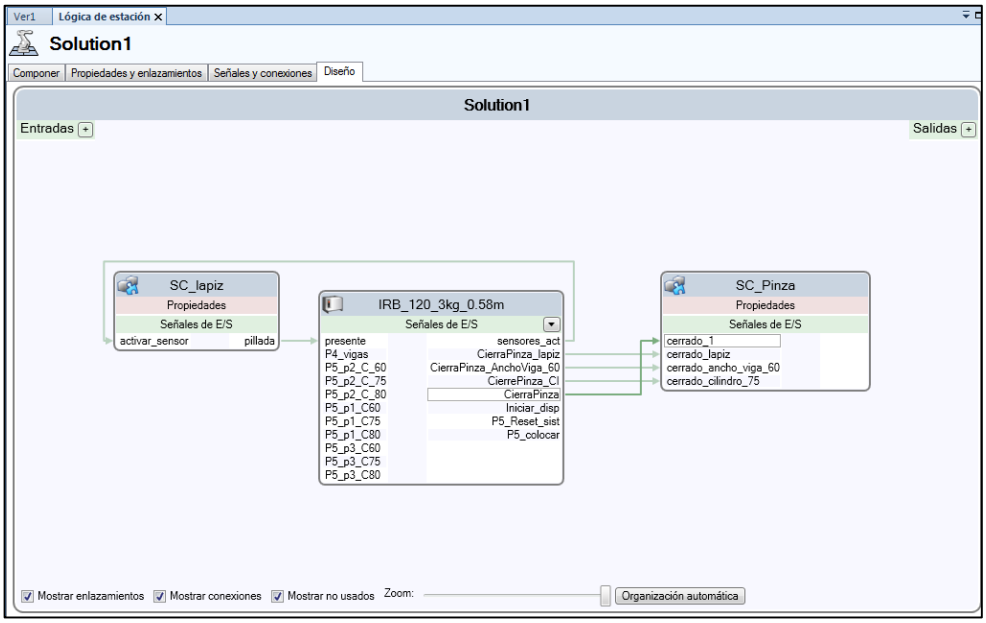


Figura 6.86 Ventana lógica de estación.

La Figura 6.86 representa una lógica de estación que ya ha sido preparada, en un principio no había señales de entrada y salidas visibles en el controlador (Elemento Central), para añadirlos se pulsa la pestaña visible al lado de “Señales de E/S” (Figura 6.87).


Figura 6.87 Representación del controlador en la lógica de estación.

Aparecerá la lista de todas las señales, disponibles incluyendo las señales internas del controlador, solo tenemos que buscar las que se han creado anterior mente y seleccionarlas, esto hará que aparezcan como se ve en la imagen.

Cuando las señales estén añadidas al controlador solo tenemos que pulsar con el ratón la señal y arrastrar hasta el destino, esto creará una conexión representado con una flecha de color verde.

Otra forma de crear esta conexión requiere ir a la pestaña de “Señales y conexiones”, en esta ventana al pulsar “Añadir conexión de E/S” emergerá una ventana en la que se puede manualmente crear una conexión (Figura 6.88).

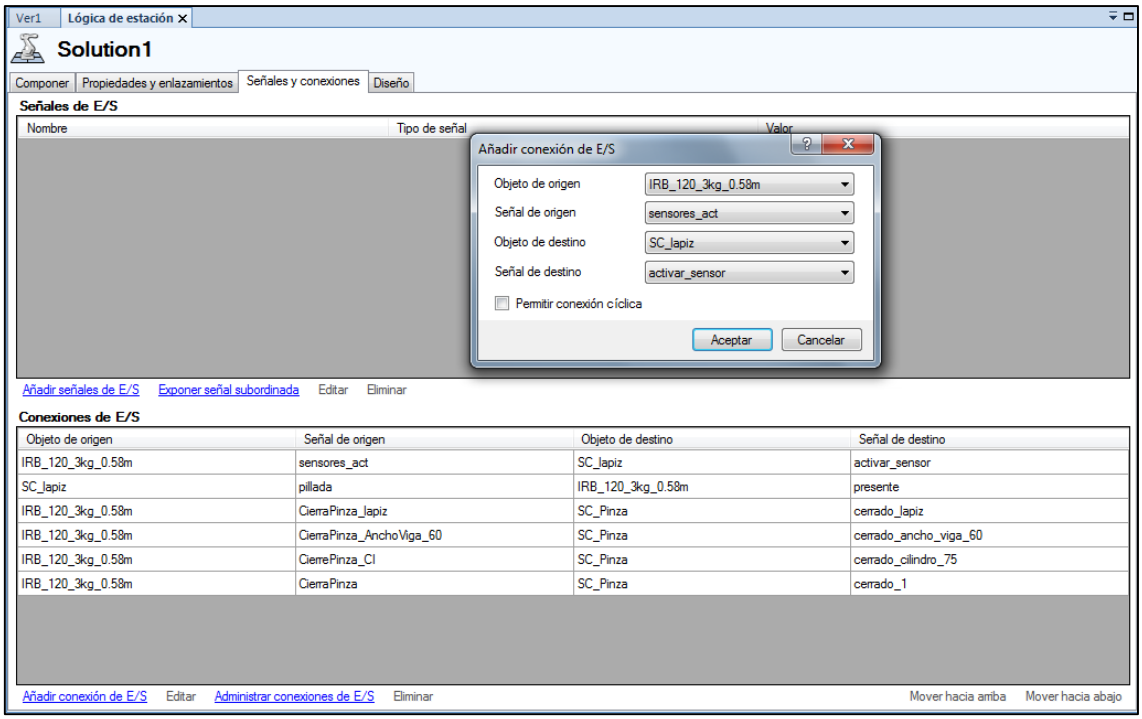


Figura 6.88 Unión de señales manual.

Creadas ya las conexiones de entradas y salidas y los puntos podemos crear una trayectoria y programar los movimientos de la práctica.

El Programa informático permite la creación de la trayectoria con los puntos antes de crear un módulo de programación, también se puede primero crear un módulo vacío para luego incluirle una trayectoria, la forma más sencilla y rápida es la primera.

La creación de una trayectoria vacía es sencilla, en la parte superior de la pantalla deberemos pulsar la pestaña de “Inicio” y en el menú de “Programación de trayectorias” pulsaremos la opción de “Ruta”, esto creará una trayectoria vacía con el nombre “Path_10”, esta trayectoria y todas las demás en el futuro se encontrarán en la carpeta de “trayectorias y procedimientos”, visible en el lado izquierdo de la pantalla, (Figura 6.89, a).

Para añadir los puntos de la práctica deberemos seleccionar todos los puntos pulsando en ellos mientras mantenemos presionado el botón Ctrl del teclado (O de una en una) y arrastraremos al elemento recientemente creado “Path_10”.

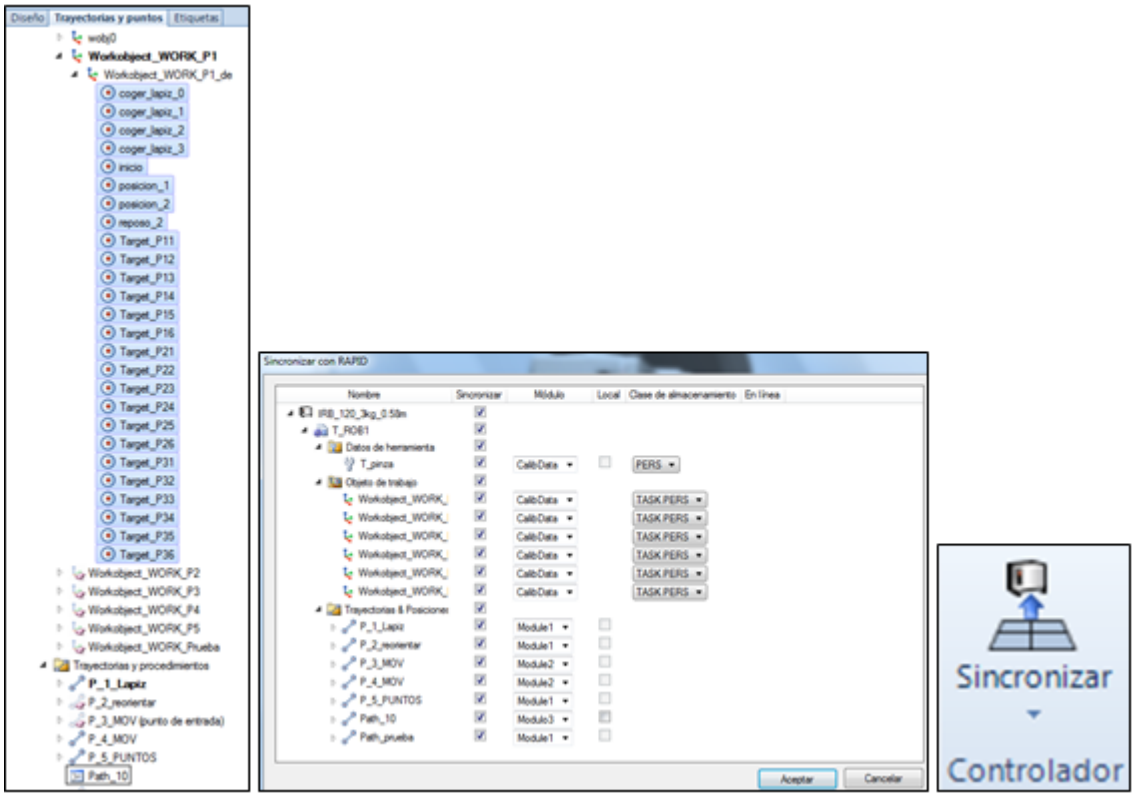


Figura 6.89 a) Selección de puntos creados. b) Ventana para sincronización. c) Icono sincronizar.

El próximo paso creará una primera trayectoria, pero también nos permitirá crear un módulo de programación, en la parte superior de la pantalla, dentro del menú “Controlador” hay un único elemento en el que se lee “sincronizar” (Figura 6.89 c), al pulsarlo aparecerá la ventana emergente (Figura 6.89, b).

En la imagen Figura 6.89 b solo aparecerá la trayectoria recientemente creada, ya que es la única que se ha hecho, al haber solo una trayectoria todas las pestañas de la Figura 6.89 b deberán tener tic (cuando se añadan más trayectorias no se deben sincronizar las de prácticas anteriores).

A la derecha de la celda con tic de la trayectoria “Path_10” recientemente creada (trayectoria que no debe necesariamente tener esa nomenclatura) se selecciona el modulo en el que se quiere escribir el programa, como no tenemos un módulo creado, seleccionamos el triángulo que indica que hay varias opciones disponibles y seleccionaremos “user”, “user”

tendrá un pequeño dibujo de una lápiz al lado, esto indica que puedes cambiarle el nombre, en el caso de este trabajo el primero se llamó Modulo1, como se muestra en la Figura 6.90.

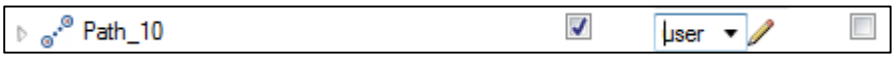


Figura 6.90 Modulo de programación para trayectoria.

Es muy importante que en caso de que se deba modificar la trayectoria (añadiendo un punto) o modificando el punto ya definido se debe de volver a sincronizar con el módulo (al hacer esto se borra el programa escrito, si se debe hacer una sincronización de una trayectoria con una programación compleja se aconseja copiar el texto del programa antes), seleccionando los elementos modificados.

Como ejemplo, la Figura 6.91 representa un caso en el que se ha tenido que modificar la posición de un punto de la trayectoria y se ha añadido uno nuevo a la trayectoria.

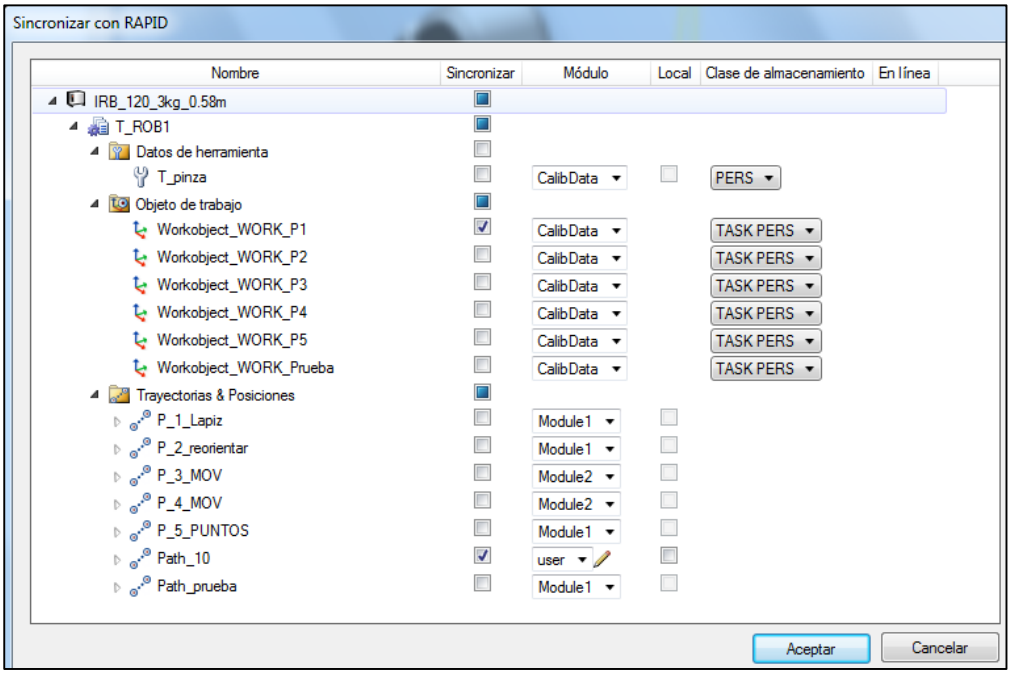


Figura 6.91 Ventana de sincronización

Con estos pasos realizados la estación se encuentra en disposición de que se escriba su programa.

Primero se abrirá el menú “RAPID” y dentro de este buscaremos el menú del módulo como muestra la Figura 6.92 a.

Seleccionaremos el módulo en el que se va a escribir el programa (El que tiene la trayectoria de interés) y hacemos doble pulsación o pulsaremos el botón derecho del ratón y pulsaremos “Editor RAPID” (Figura 6.92 b).

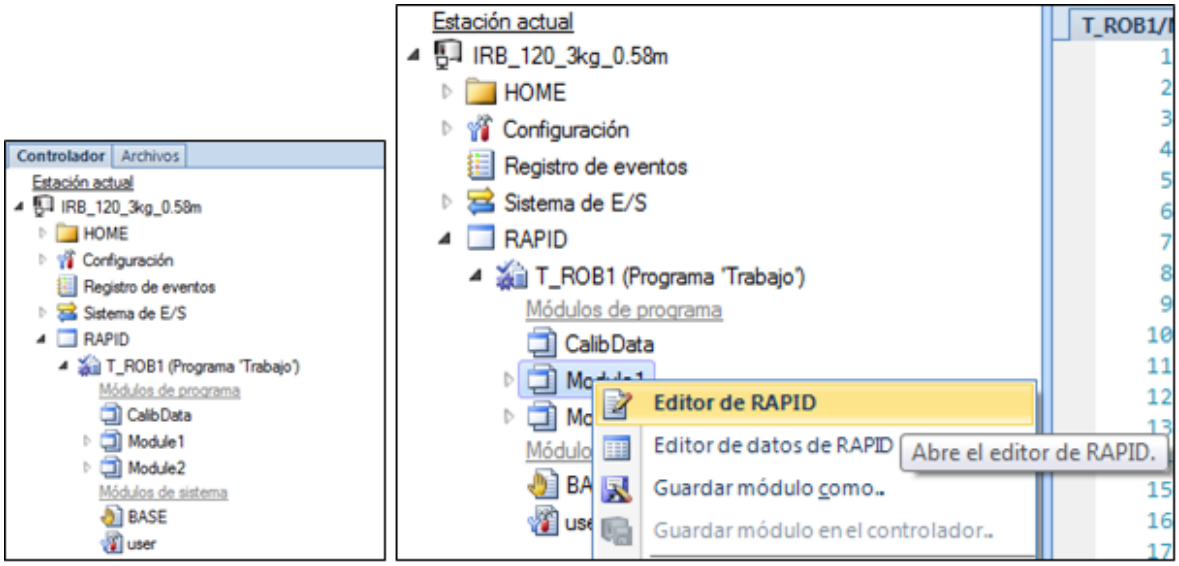


Figura 6.92 a) Menú RAPID b) Editor RAPID.

Esto abrirá una ventana donde se observa que las primeras líneas de código son los puntos que forman parte del módulo (Figura 6.93).

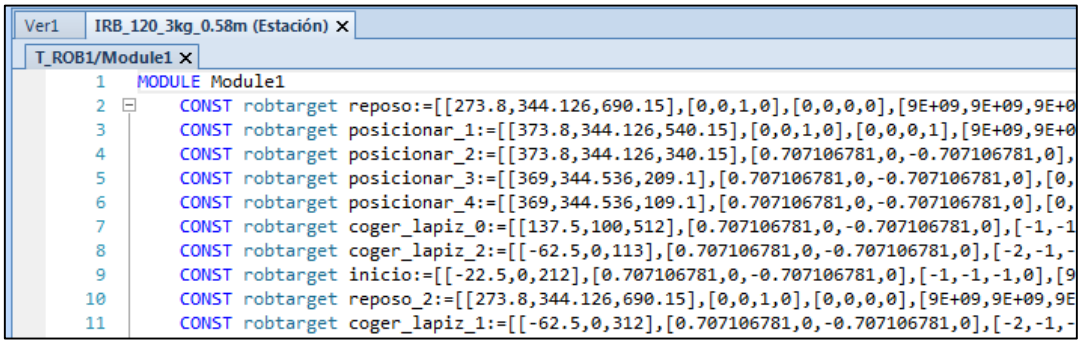


Figura 6.93 Ventana editor RAPID.

Después se ve que la trayectoria que se ha añadido al módulo con todos sus puntos añadidos uno tras otro, esto es útil ya que la línea de código que indica movimiento es larga y es más sencillo modificar las generadas automáticamente que escribirlas de manera manual (Figura 6.94).

```
75
76      PROC P_1_Lapiz()
77          SetDO CierraPinza_AnchoViga_60, 0;
78          SetDO CierraPinza, 0;
79          SetDO CierrePinza_CI, 0;
80          SetDO CierraPinza_lapiz, 1;
81          SetDO CierraPinza_lapiz, 0;
82          SetDO sensores_act, 0;
83          SetDO sensores_act, 1;
84          MoveJ reposo_2,v200,fine,T_pinza\WObj:=Workobject_WORK_P1;
85          MoveJ coger_lapiz_0,v100,z50,T_pinza\WObj:=Workobject_WORK_P1;
86          MoveJ coger_lapiz_1,v100,z50,T_pinza\WObj:=Workobject_WORK_P1;
87          MoveL coger_lapiz_2,v100,fine,T_pinza\WObj:=Workobject_WORK_P1;
88          MoveL coger_lapiz_3,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
89          WaitTime 1;
90          SetDO CierraPinza_lapiz, 1;
91          WaitDI presente, 1;
92          WaitTime 2;
93          MoveL inicio,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
94          MoveJ posicion_1,v50,z50,T_pinza\WObj:=Workobject_WORK_P1;
```

Figura 6.94 Código RAPID para la práctica 1.

En la Figura 6.94 se observa el código creado para programar la práctica nº1 el código completo se encuentra en el anexo al final de este trabajo, a continuación, se va explicar los elementos necesarios para la programación de esta práctica. [11], [16], [21].

Solo son necesarios cuatro tipos de comandos para realizar el movimiento desde el reposo al componente inteligente lápiz, cogerlo, mover el conjunto lápiz-brazo al plano del folio, mover el conjunto para que simule escribir unas iniciales, volver a depositar el componente inteligente lápiz en su posición original y volver a colocar el brazo en la posición de reposo.

Estos cuatro comandos son:

- SetDO.

En inglés Set significa fijar y DO representa Digital Output, que significa salida digital, este comando permite controlar las señales de salida del controlador y fijar su valor en 0 o 1.

La estructura de este comando es la siguiente:

SetDO #####, 0/1;

Siendo ##### el nombre de la señal salida, y 0/1 el valor que se desee dar a la señal.

- MoveJ

MoveJ es uno de los comandos de movimiento donde J proviene de Joint, que significa articulación, este comando permite al Robot mover las articulaciones con libertad, solo se le indica el destino del punto.

La estructura de este comando es:

MoveJ #####, v#, z#, Herramienta\plano de trabajo;

Siendo ##### el nombre del punto de destino, v# la velocidad a la que se mueve el elemento terminal del brazo (la pinza), en este trabajo se ha decidido darle un valor de 50 cuando se tiene tomado un objeto y superior cuando no se encuentra cogiendo o muy próximo de alguna geometría.

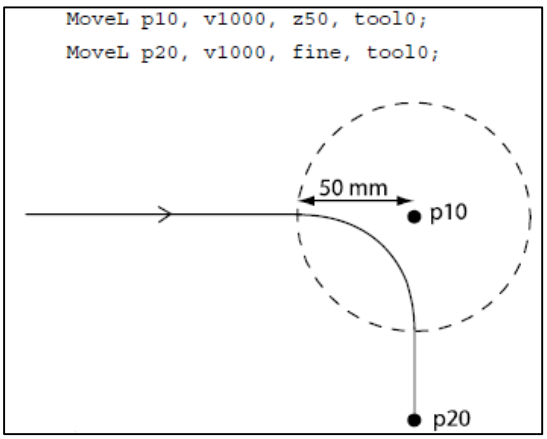


Figura 6.95 Esquema de movimiento en RAPID.

Z# representa precisión, si se requiere que la herramienta pase por el punto en concreto se usará la línea fine en vez de z# (Figura 6.95).

- MoveL

MoveL es uno de los comandos de movimiento donde L proviene de Linear, que significa Lineal, este comando condiciona al brazo a que debe de realizar un movimiento lineal (y no curvo como en MoveJ), en posiciones límite el brazo puede encontrarse en una circunstancia en la que no es posible realizar un movimiento recto.

La estructura de este comando es larga y es:

MoveL #####, v#, z#, Herramienta\plano de trabajo;

Este movimiento la misma estructura que el comando MoveJ previamente explicado.

- WaitTime

WaitTime es un comando que como su nombre en inglés indica hace que el programa espere en la línea de comando en la que se encuentre este código un determinado número de segundos, esto es útil para permitir que el brazo se mueva de un lado a otro o que el controlador pueda finalizar una tarea.

La estructura de este comando es larga y es:

WaitTime #;

Donde # representa los segundos que el programa esperará en esa línea, si se desea esperar menos de un segundo la se expresará en 0 punto #.

- WaitDI

WaitDI es un comendo que indica que el programa esperará hasta que una señal de estrada (Digita Input) del controlador pase a ser de un valor determinado.

La estructura de este comando es larga y es:

WaitDI #####, 0/1;

Siendo ##### el nombre de la señal de entrada del controlador y 0/1 representa cual es el valor de la señal que debe tener para que el programa continúe.

Cuando el código del programa se encuentre escrito se deberán aplicar los cambios realizados, para ello hay que pulsar aplicar en el botón que se encuentra en la parte superior de la pantalla, mostrado en la Figura 6.96.

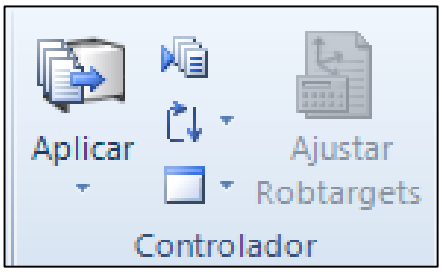


Figura 6.96 Menú controlador.

Aplicados ya los cambios solo queda configurar la simulación y probar el programa. La configuración de la simulación se hace desde la pestaña “Simulación” visible en la parte superior de la pantalla, después se pulsará “Configurar simulación “dentro del menú ’Configurar” (Figura 6.97).

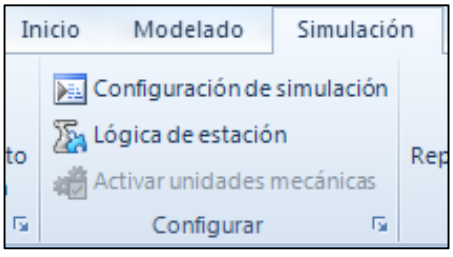


Figura 6.97 Menú configurar.

La ventana que aparecerá será la Figura 6.98.

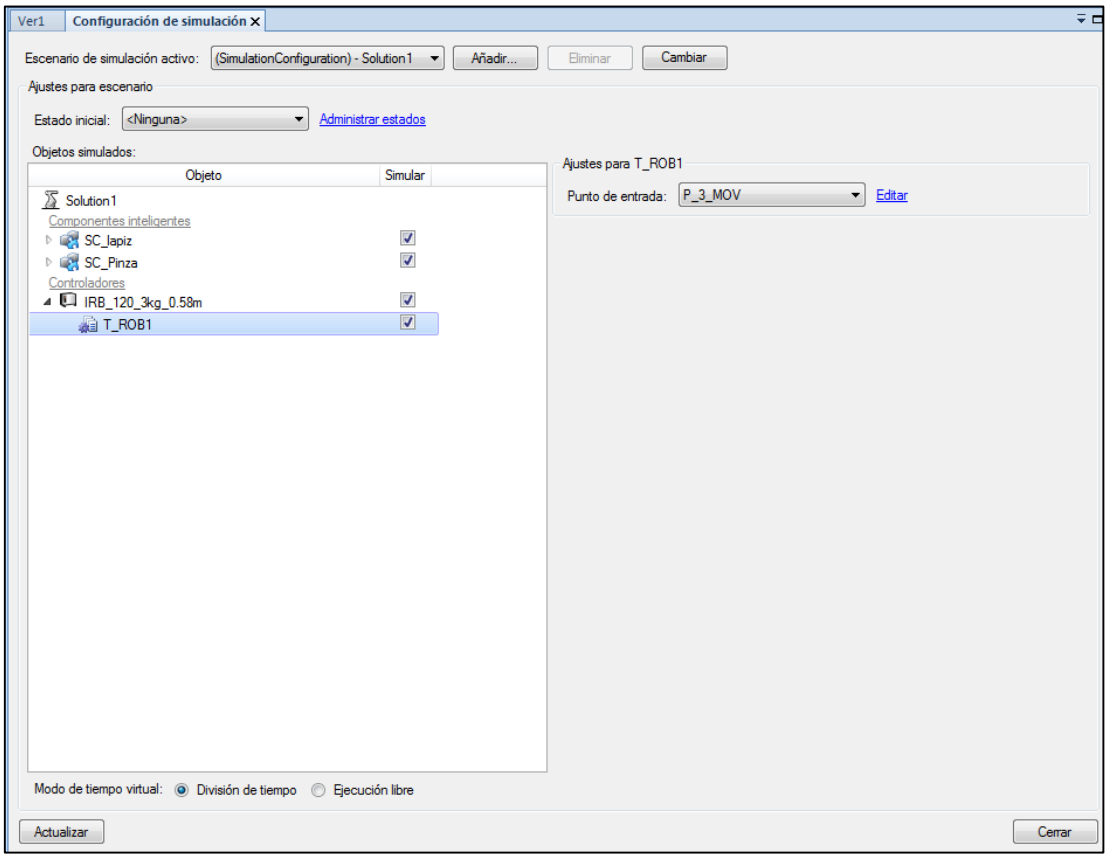


Figura 6.98 Ventana de configuración de simulación.

Dentro del grupo "Controlador" seleccionaremos el único robot de la estación, en Punto de entrada se selecciona que programa se lee dentro de un módulo, pulsaremos en el triángulo para mostrar las opciones y seleccionaremos el programa que se desee, en este caso el que contenga el programa de la práctica nº1, pulsaremos "Actualizar" en la esquina inferior izquierda para comprobar que se ha realizado el cambio adecuado y se podrá cerrar la ventana, la estación ya está configurada para realizar la práctica nº1.

Antes de empezar la simulación se aconseja que se divida la pantalla de visualización en dos partes, una con la vista de la estación y otra con el código escrito.

De esta forma si el programa falla en alguna línea será más fácil detectar el fallo, para hacer esta división se debe de volver a la pantalla editor del programa en RAPID, y en la pestaña en la que se lee el nombre del Robot (si no se ha modificado debería de ser T_ROB1) y el nombre del módulo (Figura 6.99).

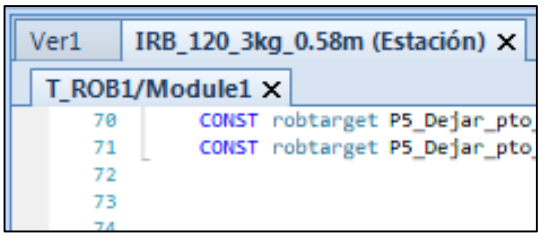


Figura 6.99 Nombre del robot y módulo.

Se deberá pulsar el botón derecho del ratón y entre las opciones que se volverán visibles se pulsará en "Nuevo grupo de pestañas verticales", quedando la pantalla como se muestra en la Figura 6.100 a.

Para revertir a la configuración de pantalla anterior solo hay que repetir el proceso y pulsar en "Ir a grupo de pestañas anterior" (Figura 6.100, b).

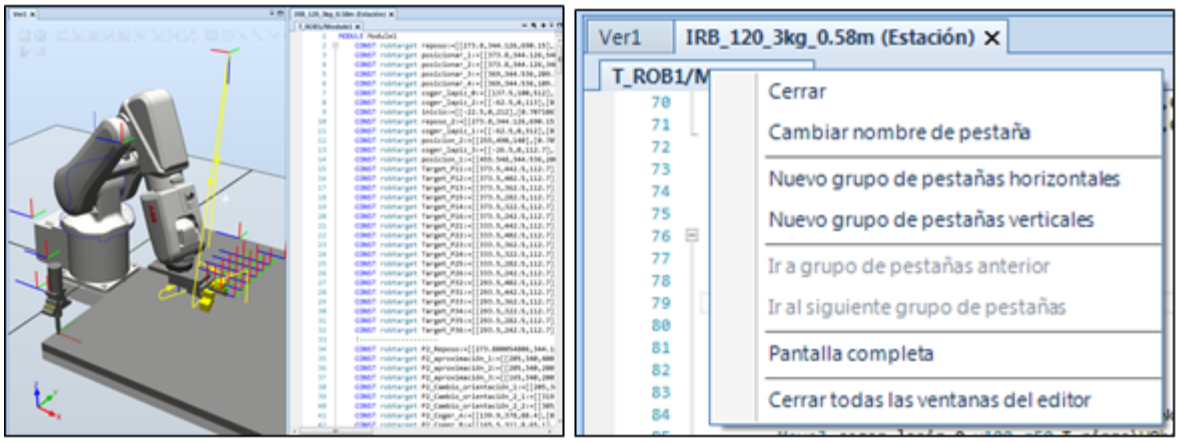


Figura 6.100 a)Ventana de visualización dividida verticalmente. b) Menú de opciones visualización ventanas.

Como ya se ha indicado anteriormente la estación se encuentra en disposición de simular el programa de la práctican°1, para ello se pulsará el símbolo de reproducción que es visible en la parte superior de la representación 3D de la estación (Figura 6.101).

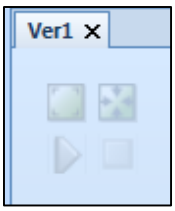


Figura 6.101 Botón de inicio de simulación.

### 6.2.3 Práctica nº2: Reorientar pieza.

En esta práctica se cambiará la orientación del componente inteligente Viga respecto al brazo de la forma mostrada en la Figura 6.102. En el trabajo anterior está práctica estaba diseñada para que el alumno creara una serie de movimientos que deshiciese los cambios realizados que se le mostraban al inicio de una clase, el objetivo del alumno era pasar de la posición C a la A, en este trabajo en cambio la practica ha resultado más útil para practicar el agarre en distintas posiciones y además introduce la novedad de usar dos señales de cierre.

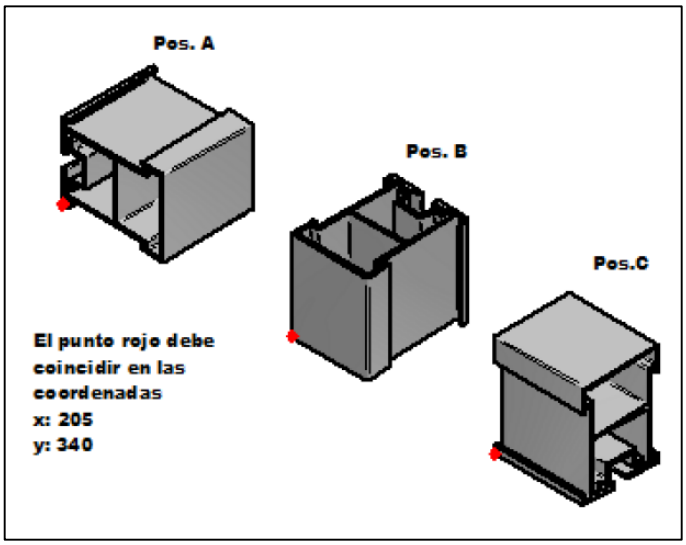


Figura 6.102 Esquema de posiciones en P2 [20].

Primero se creará una estación como se explicó en la práctica nº0, se importará el componente inteligente Viga, se creará un plano de trabajo, etc. Todos esos pasos se han explicado en la práctica nº1.

Para ayudar en el proceso de colocar geometrías tridimensionales es útil crear un punto en el plano de trabajo que sirva como guía, en la práctica nº1 ese punto era el origen del plano de trabajo, en el caso de la práctica nº2 ese punto será el (205, 340, 0), este punto no se añadirá a la trayectoria, ya que su uso será el de asistir en el posicionamiento del elemento Viga, este punto de guiado se muestra en la Figura 6.103.

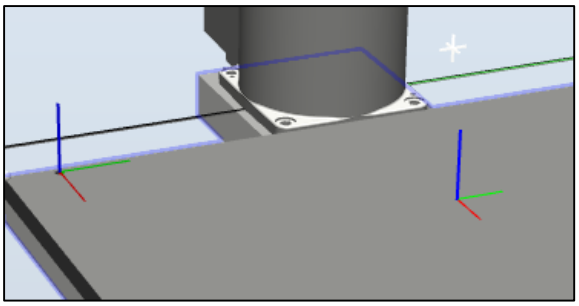

Figura 6.103 Punto de posicionamiento de vigas.

Como se puede observar en la Figura 6.102, el punto (205, 340, 0) siempre está en contacto con una esquina del elemento Viga, sin embargo el elemento Viga tiene las esquinas redondeadas, para este trabajo se ha colocado la Viga visualizando el elemento como un paralelepípedo, por tanto a la hora de colocar la viga en posición hará falta tener constancias de distintas medidas de la viga, (Conocidas y mostradas en apartados anteriores), gracias a las órdenes de "offset" y "situar en un punto" se podrá colocar la viga en la posición correcta.

La posición A quedará como muestra la Figura 6.104 a, en la Figura 6.104 b puede observarse el punto de agarre de esa posición con una pre-visualización de la herramienta.

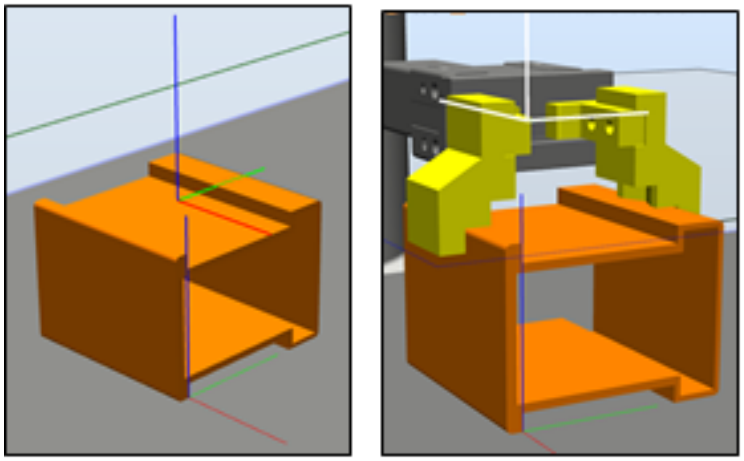

Figura 6.104 a) Posición A. b) Posición A con herramienta.

Para crear este punto haremos uso del punto de referencia creado, primero se creará un punto que ocupe la misma posición que el punto de referencia Figura 6.105 a, luego se usará la orden de offset para mover ese punto hasta la cara superior del paralelepípedo imaginario Figura 6.105 b, a este punto debemos de además añadirle las distancias correspondientes a un cierre de 80 mm, como se observa en la Figura 6.106, este agarre se efectúa con la superficie de agarre más alejada del accesorio agarre.

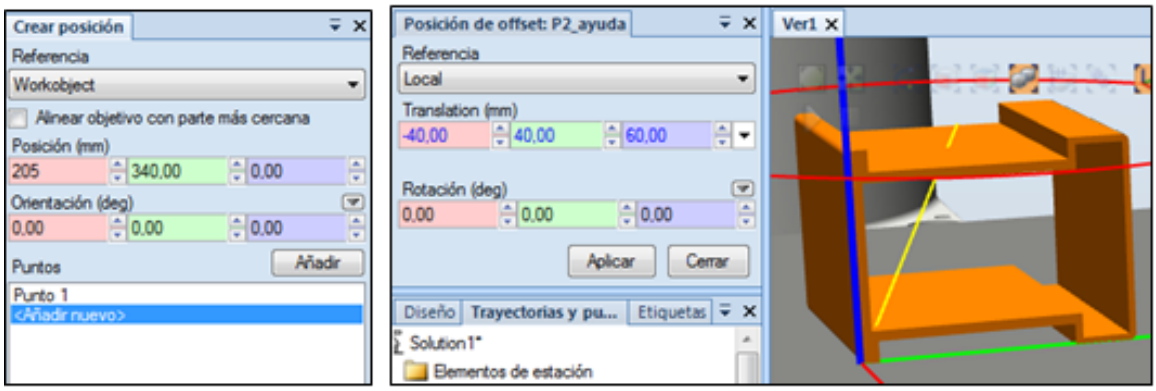


Figura 6.105 a) Posicionamiento punto de agarre paso 1. b) Posicionamiento punto de agarre paso 2.

Se ha de tener en cuenta que para realizar el segundo offset se debe de borrar las distancias introducidas para la primera.

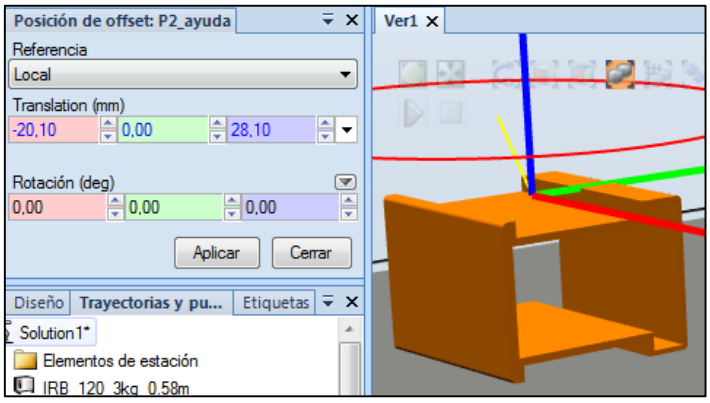


Figura 6.106 Posicionamiento punto de agarre paso 3.

En este caso se observó más adelante que el 0,1mm de seguridad elegidos al calcular la distancia 28.1mm no fueron suficientes en este caso, no respecto a las colisiones, pero si respecto al plano sensor de menor tamaño, este se encuentra en esta cara de la viga, y también se encuentra alejada de la geometría para evitar que detecte al propio elemento viga, por tanto, se tuvo que modificar este punto elevándolo 0,5mm más, se insta que se haga ese cambio en este paso.

Para los demás puntos de agarre primero se situó la geometría de la viga en la posición en la que se iba a agarrar, con el fin de facilitar la pre-visualización del paralelepípedo, facilitando también la selección de distancias a añadir respecto al punto de referencia, en la posición B se tomó a la viga como se muestra en la Figura 6.107 a.

La posición C tuvo en un primer momento una posición de agarre que mostro no ser alcanzable, como se muestra en la Figura 6.107 b.

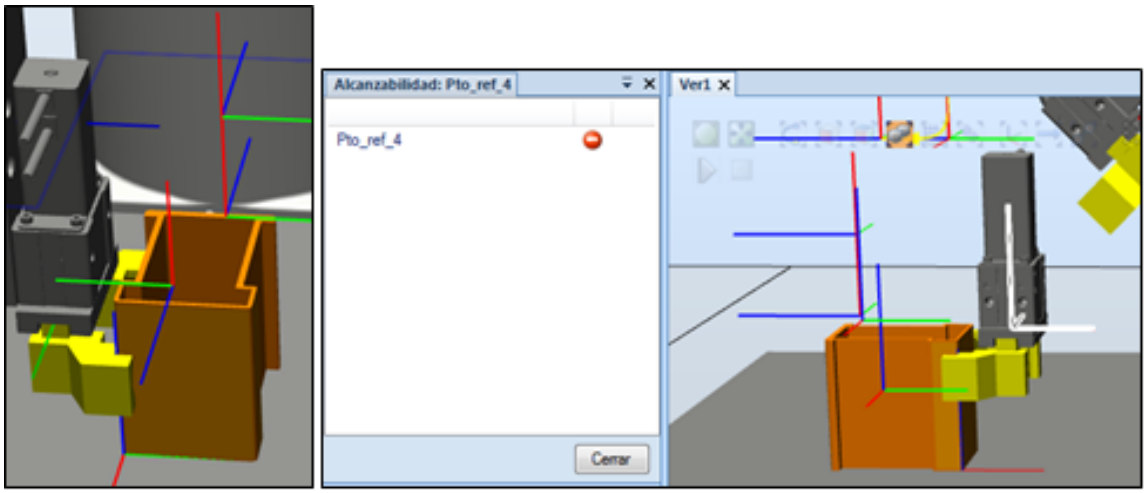


Figura 6.107 a) Agarre de viga en posición B. b) Ventana de alcanzabilidad de puntos.

Se tomó otro punto de agarre, el cual se muestra en la Figura 6.108 a, para tomar la pieza en la posición B para pasarla al C y se tuvo en cuenta esa posición para definir el punto en el que se agarra en la posición C para pasar a B (Figura 6.108 b).

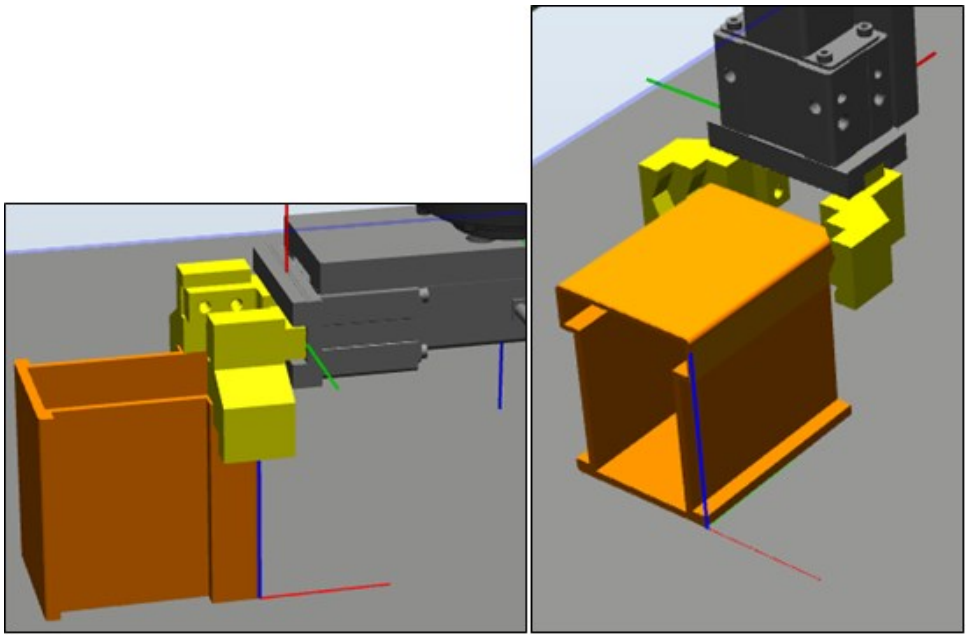

Figura 6.108 a) Punto de agarre para pasar a posición C. b) Viga en posición C.

Estos dos puntos se crearon de manera similar a los dos anteriores con la diferencia de que el centro al que se dirigía la primera aproximación con la orden offset no es el centro de una de las caras de paralelepípedo imaginario, sino que es el centro del rectángulo

formado por el ancho de la viga de 60mm y la altura del saliente de mayor con envergadura de 18mm, se comprobó más adelante en la simulación que la geometría del agarre dispone de unas secciones triangulares en las que no hay sólido,  y el área triangular tiene una base con una longitud mayor que 18mm, lo que obligó desplazar el punto en 5mm para que existiera una parte del elemento sólido del accesorio agarre que cortase el plano sensor.

También se ha de tener en cuenta que hasta ahora la posición de la herramienta respecto a la muñeca del brazo se ha mantenido constante, lo que ha permitido puntos de aproximación lineales en los que no existía ningún giro, sin embargo, en esta práctica eso no es posible, si el punto mostrado anteriormente se intentase tomar sin girar la herramienta el brazo entraría en una zona no alcanzable, entre la posición B y la C se han creado unos punto que giran a la herramienta para darle la posición de agarre de la posición C.

En la Figura 6.109 pueden observarse los tres puntos que forman este movimiento, pudiendo observarse el cambio del eje de la muñeca.

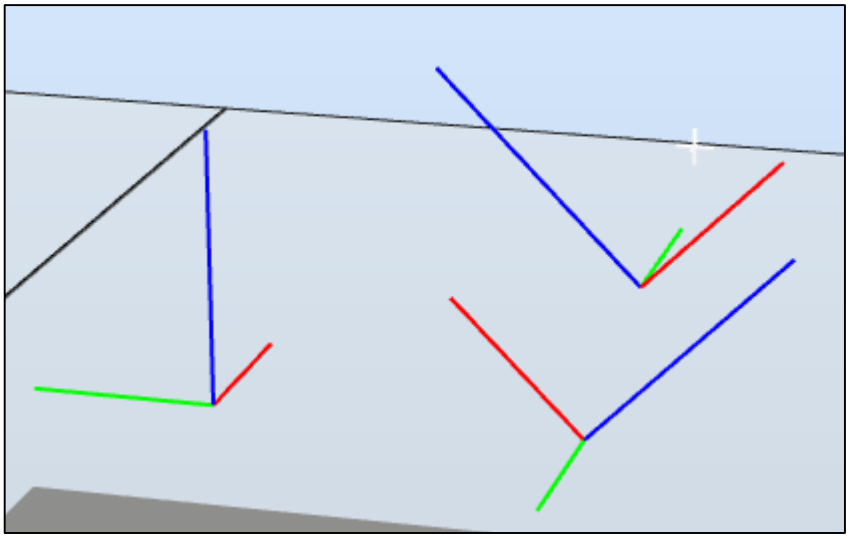

Figura 6.109 Puntos de cambio de orientación P2.

Los puntos de aproximación son necesarios para tener un mayor control del movimiento del brazo, y el número de estos puntos no está limitado a ninguna cantidad en particular, los puntos creados para realizar la práctica nº2 en este trabajo son los visibles en la Figura 6.110 a, pudiendo observarse su representación en la estación la Figura 6.110 b.

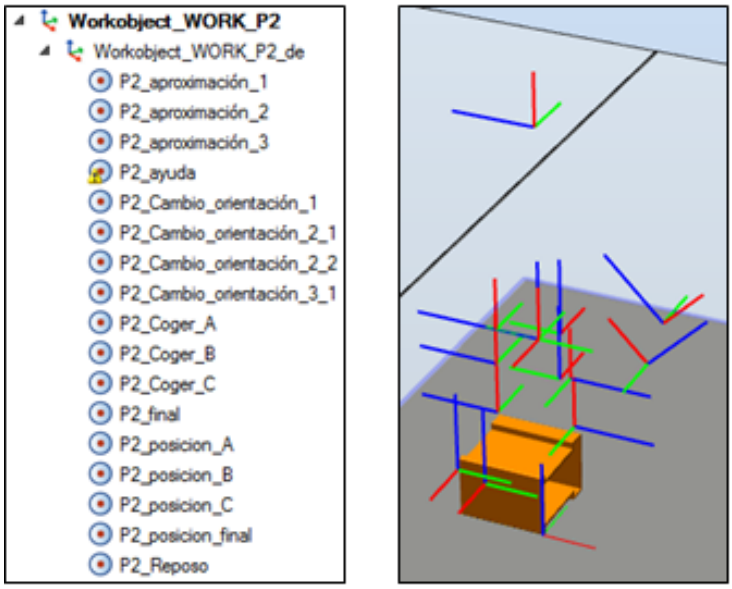


Figura 6.110 a) Puntos creados para P2, vista1. b) Puntos creados P2, vista 2.

Antes de proseguir con las entradas y salidas del controlador o la lógica de estación se comprueba la alcanzabilidad de los puntos creados y de configuraran las posiciones de las articulaciones del brazo en el mismo, paso explicado previamente en la práctica nº1.

A continuación, se prosigue con la creación de las entradas y salidas del controlador, en este trabajo todas las prácticas han sido creadas en la misma estación creando distintos planos de trabajo y módulos de programación para cada una de las prácticas, esto permite solo tener que configurar las entradas y salidas del controlador una vez.

Si se desease tener una estación por práctica se debería de volver a configurar las entradas y salidas del controlador como se hizo en la práctica nº1, si se crea una estación para todas las prácticas se deberá de añadir las entradas o salidas que sean nuevas.

Creadas ya estas señales si es que son necesarias se pasará a crear la lógica de la estación como se explicó en la práctica nº1, quedando como se muestra en la Figura 6.111.

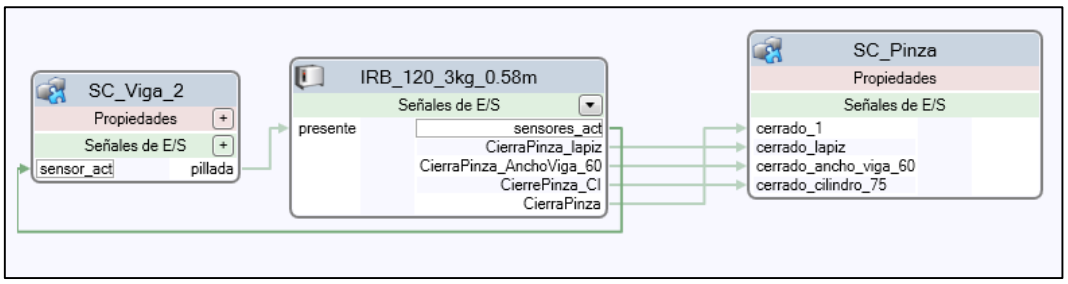


Figura 6.111 Lógica de estación P2.

Se pasará ahora a la creación de una trayectoria vacía con, los puntos creados y se sincronizará con los módulos de programación, en este trabajo, varias prácticas comparten un mismo módulo, pero nada impide la creación de tantos módulos como prácticas se realicen.

Creado ya el módulo de programación con la trayectoria y los puntos de la misma configuraremos la estación para que tomo como programa a simular el de la practica creada, esto ha sido explicado con anterioridad, teniendo que pulsar "Configuración de simulación" dentro del menú inicial "simulación", como se muestra en la Figura 6.112.

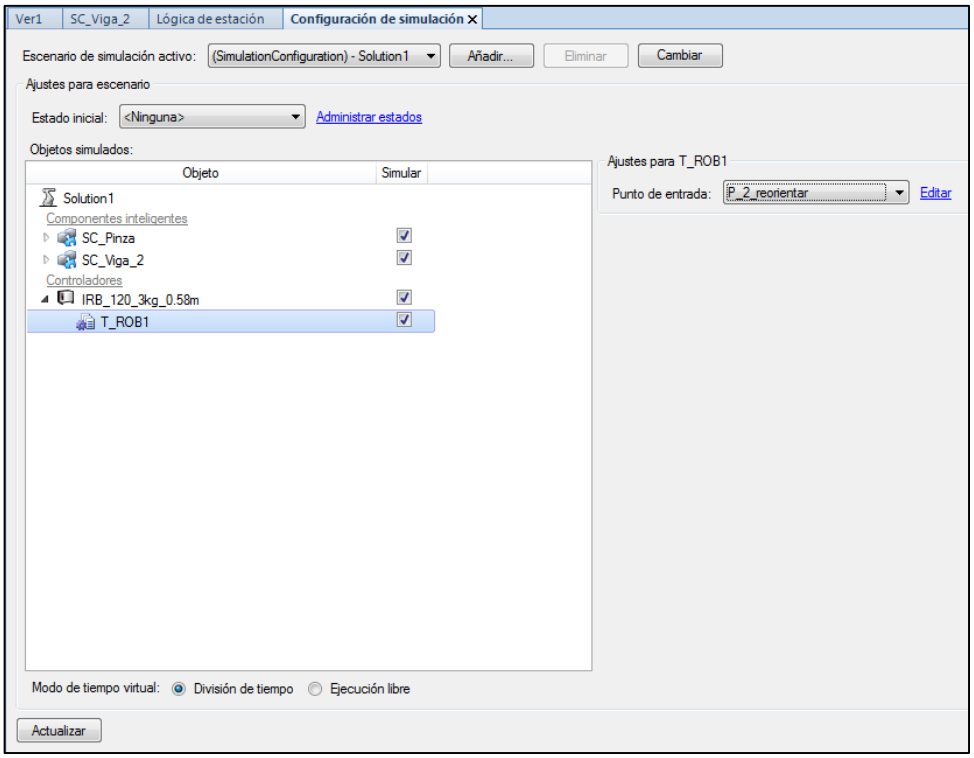


Figura 6.112 Ventana de configuración de simulación.

Creado ya un módulo de programación y configurada la simulación se puede programar los movimientos de la práctica para inmediatamente después de haber aplicado los cambios simular los movimientos. (el código usado en este programa estará disponible en el apartado Anexos).

Para crear el programa de la práctica no ha sido necesaria la inclusión de ningún comando de RAPID desconocido, si bien hay que tener en cuenta que el brazo se encuentra en una posición límite dentro de su área de accesibilidad, lo que impide el uso del comando MoveL en muchos de los puntos por la necesidad del robot de efectuar movimientos curvos.

Cuando se escribe un código en el módulo de programación y se guarda la estación los datos de este módulo se guardan como archivos internos, si se desea exportar un módulo de programación para poder implementarlo en otra estación distinta se puede guardar los módulos como archivos externos, esto se hace desde el menú visible dentro de la pestaña de "RAPID" como se observa en la Figura 6.113.

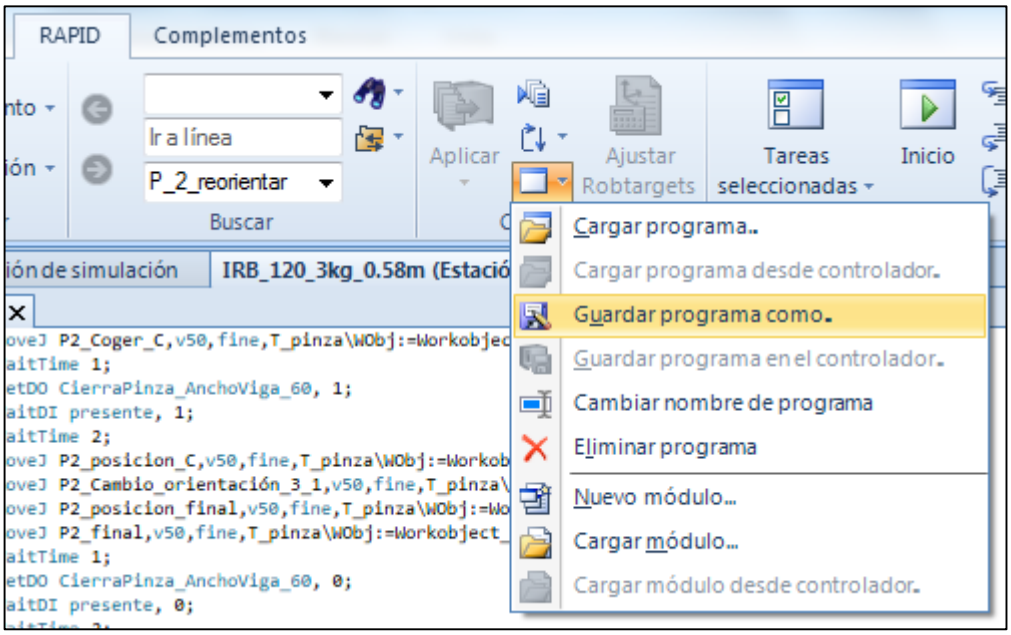


Figura 6.113 Ventana guardado de módulos de programación.

Como se comentó en la práctica nº1 se aconseja dividir la pestaña en dos partes mientras se simula el programa, esto permite visualizar la simulación y ver el código del programa al mismo tiempo, lo que permite detectar con mayor facilidad errores en el código.

### 6.2.4 Práctica nº3: Dispensador.

En esta práctica se simulará el movimiento de varios cilindros dentro de un distribuidor con un plano inclinado como se observa en la Figura 6.114.

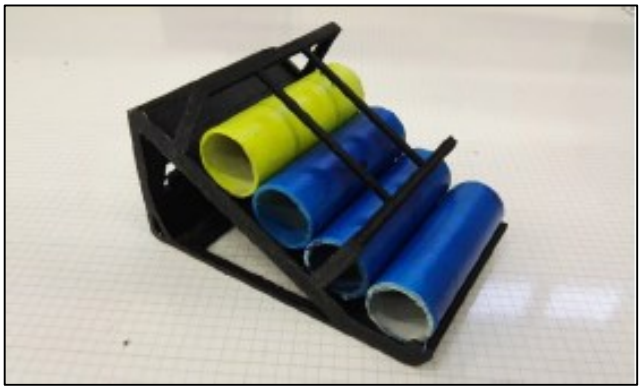

Figura 6.114 Dispensador con cilindros disponible en laboratorio.

En el laboratorio se dispone de un cilindro de distinto color, el cual se usar para diferenciarlo del resto, permitiendo crear un programa en el que se tomarán cilindros de la posición más baja de distribuidor a la más alta hasta que la posición más alta la ocupe el cilindro de distinto color, en caso de que al iniciar el programa el cilindro de distinto color se encuentre en esa posición se dará una vuelta completa hasta que el cilindro vuelva a la posición inicial.

La práctica del dispensador fue usada en el trabajo anterior para indagar en mayor profundidad en el uso del FlexPendant en programación, para ello se creó un programa que pedía una información al usuario, para después aplicar un programa que se adaptaría a la información aportada, el programa de FlexPendant comentado es visible en la Figura 6.115.

```
PROC Programa_4()!Selección dispensador

  !Definición de variables para pregunta
   VAR num Pr4_TPAnswer:=0;
         CONST string Pr4_TPText:="Posición del cilindro";
         CONST string Pr4_TPFK1:="Pos. 1";
         CONST string Pr4_TPFK2:="Pos. 2";
         CONST string Pr4_TPFK3:="Pos. 3";
         CONST string Pr4_TPFK4:="Pos. 4";

TPReadFK Pr4_TPAnswer, Pr4_TPText, Pr4_TPFK1, Pr4_TPFK2, Pr4_TPFK3, Pr4_TPFK4, stEmpty; !Código de pregunta
 IF  Pr4_TPAnswer=1 THEN  !Si el dispensador se encuentra en primera posición, mover hasta dejar igual
     Pr4_TPAnswer:=5;
 ENDIF
WHILE Pr4_TPAnswer>1 DO  !Mientras la posición sea mayor que 1 realizara el programa 3
    Programa_3;
    Pr4_TPAnswer:=Pr4_TPAnswer-1;
ENDWHILE

ENDPROC !Selección dispensador
```

Figura 6.115 Código RAPID para uso de FlexPendant.

Sin embargo, no fue posible duplicar esta práctica en RobotStudio, si bien el programa puede simular la herramienta FlexPendant y el código mostrado funciona, mientras se simula la herramienta FlexPendant los sensores no se simulan a tiempo real.

Como resultado se tiene que al empezar la simulación el brazo se mueve correctamente hasta la primera posición de agarre para quedarse en la misma sin poder avanzar.

Que los sensores no sean simulados en tiempo real implica que hay que activarlos y desactivarlos manualmente para que cambien de un valor a otro, lo que hace que pierda sentido la simulación al tener que ser un proceso automatizado.

Se decidió crear un programa en el cual la primera línea de código indica cuantas veces se debe de repetir el movimiento de toma de cilindro, (La información que se pedía al usuario con el FlexPendant era la posición del cilindro amarillo).

El procedimiento para empezar la creación de esta práctica en RobotStudio es similar a las prácticas anteriores, diferenciándose en que hay varios componentes inteligentes en el plano de trabajo.

Se debe de tener precaución, los componentes inteligentes no deben entrar en contacto entre sí, el programa informático es consciente de cuando dos geometrías están en contacto, lo que permite visualizar mejor los choques, sin embargo, se ha comprobado que el deslizamiento de un cilindro dentro del distribuidor lo considera un choque.

Primero se creará un punto de referencia como se hizo en la práctica nº2, este punto se encuentra en las coordenadas (320, 340, 0) en esta coordenada se ha colocado el extremo inferior izquierdo del componente inteligente distribuidor, tal y como se muestra en la Figura 6.116.

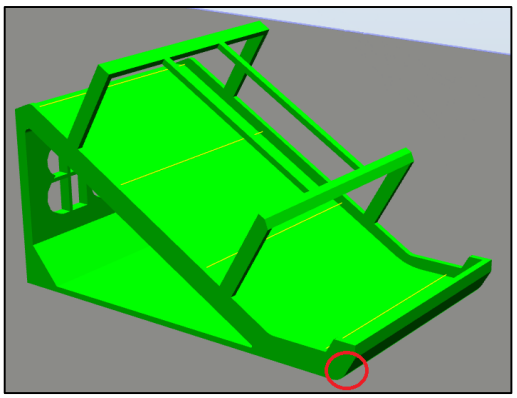

Figura 6.116 Colocación de dispensador en posición.

Como se puede observar en la imagen, la esquina, es curva, por tanto, se selecciona la esquina que se encuentra en más arriba y luego se realizará un movimiento con la orden offset, hacia abajo, siendo la medida de 12,7 milímetros.

Para colocar los componentes inteligentes cilindros primero hay que decidir cuáles van a ser las posiciones que va a ocupar, hecho esto empezaremos colocando el que ocupará la posición número 1 siendo esta la que se encuentra en la parte más alta del distribuidor.

Los archivos creados se les han dado el nombre de una posición para facilitar la configuración de la práctica mientras se realizaba el trabajo, este nombre no afecta al programa en ningún aspecto, pudiendo estos modificarse.

El primer cilindro que debe colocarse en la posición número 1 porque es la que resulta más sencilla, para ello se situará el centro de una de las caras del cilindro en el punto medio de la arista que da inicio a la pendiente, luego se usarán las herramientas de offset para colocar el cilindro en la posición mostrada en la Figura 6.117 a.

Como se ha comentado anteriormente se debe de colocar el cilindro en una posición en la que no entre en contacto con el distribuidor, esto se ha de tener en cuenta al hacer los desplazamientos con Offset, resultado de los cuales puede observarse en la Figura 6.117 b, el cilindro 'levita' sobre el distribuidor.

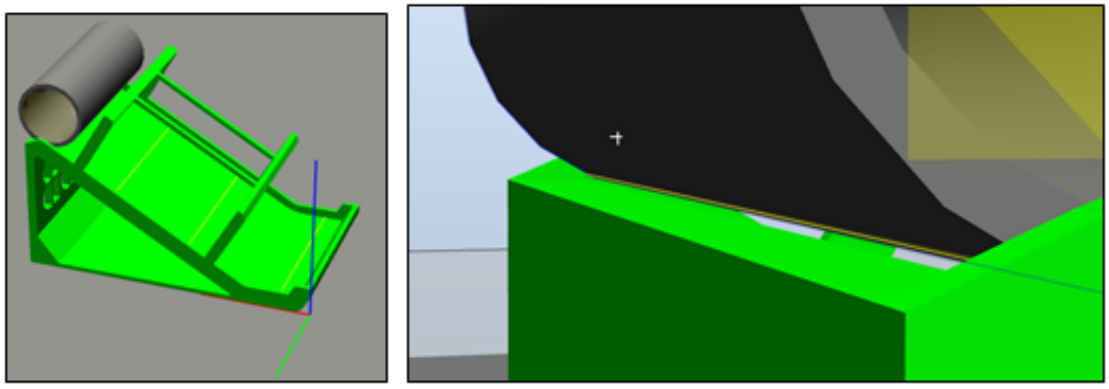

Figura 6.117 a) Colocación de cilindro en dispensador vista 1. b) Colocación cilindro en dispensador vista 2.

Este cilindro ya colocado se usará para colocar los demás, y de esta manera todos tendrán sus centros en la misma recta.

Ahora se colocará el cilindro de la posición número 2, para ello primero se situará en la misma posición del de la posición 1, usando herramientas como situar en un punto.

Realizado este paso deberemos acceder a la opción de editar componente inteligente en el distribuidor como se observa en la Figura 6.118.

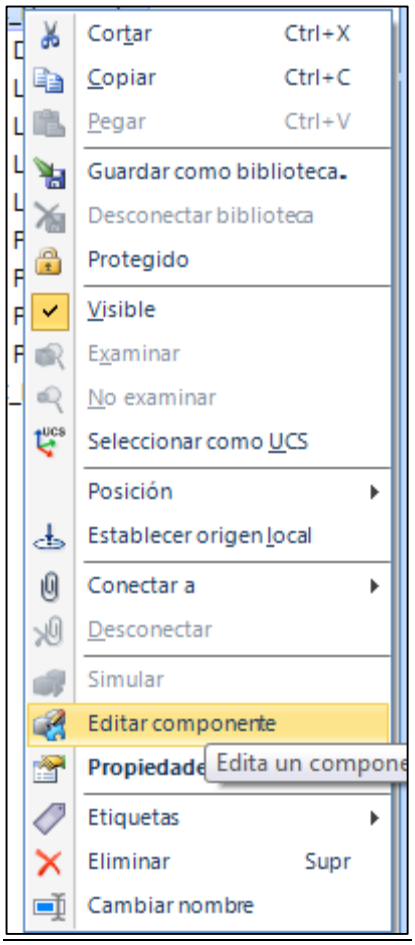


Figura 6.118 Ventana editar componente inteligente.

Desde aquí se accederá al elemento de LinearMove2 que produzca el movimiento que debe realizar el cilindro para que pase a la posición número 2 (Figura 6.119 a), se abrirá este elemento abriendo una ventana en la que podremos manualmente seleccionar en la lista emergerte en “Object” al cilindro que se desea mover, tal y como se ve en la Figura 6.119 b.

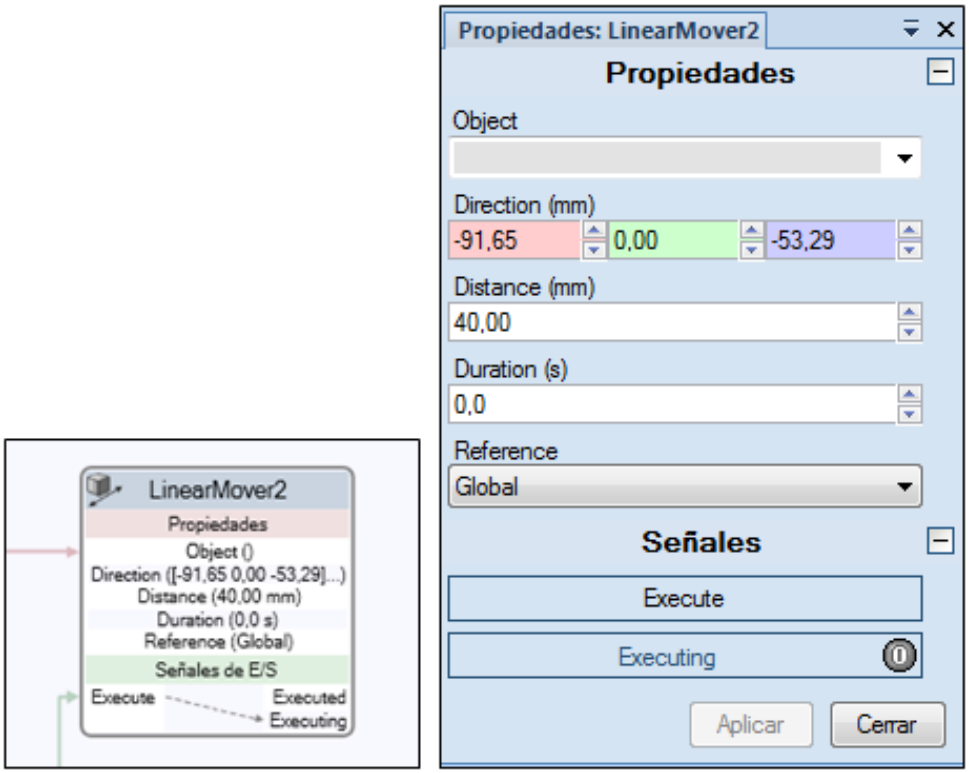


Figura 6.119 a) Elemento LinearMove2. b) Configuración LinearMove2.

Este proceso se repetirá para los dos cilindros restantes, teniendo en cuenta que para el último cilindro hay que realizar el movimiento compuesto por dos LineaMove2, el resultado será uno similar al visible en la Figura 6.120.

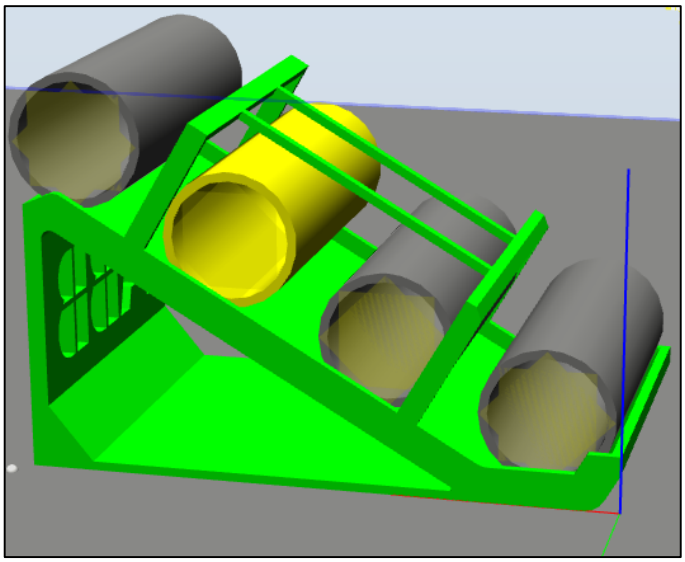

Figura 6.120 Componentes inteligentes de P3 en posición.

Colocados ya los cilindros los pasos restantes son similares a los ya explicados con anterioridad, se han creado dos puntos de agarre, uno para la posición más baja y otra para la más alta, deben de tener las mismas distancias respecto el cilindro, si no fuera así cuando se dejase el cilindro en la posición superior no estaría en la posición que se desea que esté.

Como los cilindros tiene una longitud de 75mm se debe de usar el área de agarre final, por tanto, primero se creará un punto en el centro del cilindro (ya que es un punto de fácil selección) y con la orden offset se moverá el punto al cuadrante exterior de la circunferencia, para luego desplazarlo a la mitad de la longitud del cilindro.

Hecho esto se añadirán las distancias necesarias para un agarre con el área deseada, (en este caso es la combinación de 21.5 y 28.1).

Este proceso se repetirá con el cilindro en la posición número 1. Los demás puntos son de aproximación siendo los cercanos a la posición más baja en los que se ha tenido que mostrar más cuidado, siendo necesarios varios puntos para tomar el cilindro sin provocar choques.

Los puntos creados para esta práctica con visibles en la Figura 6.121, el proceso de creación de la práctica a partir de la creación de los puntos es igual que en los anteriores, comprobar la accesibilidad, configurar las articulaciones para cada punto, crear las entradas y salidas del controlador, configurar la lógica de la estación (Figura 6.122), crear una trayectoria y sincronizarla con un módulo de programación y finalmente configurar la simulación para darle como  punto de inicio la práctica nº3.

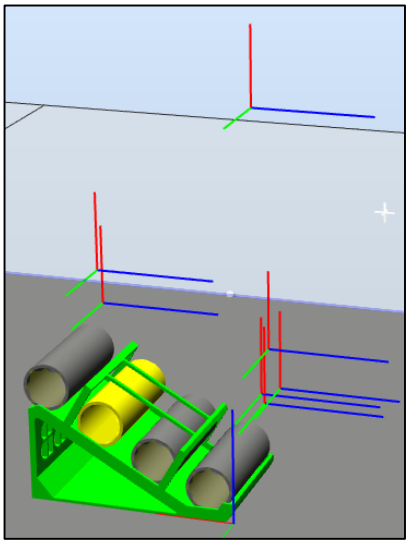

Figura 6.121 Puntos creados en P3.

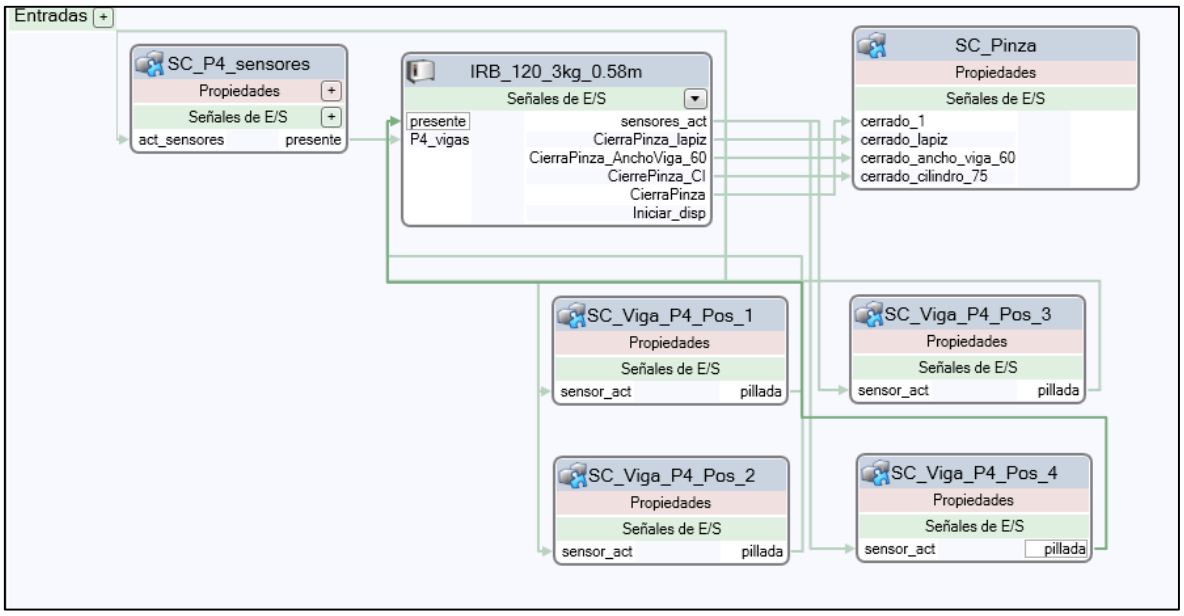


Figura 6.122 Lógica de estación P3.

En esta práctica de han usado una serie de elementos de programación RAPID que no han sido necesarios con anterioridad, estas funciones son: VAR y WHILE, ambas visible en la Figura 6.123, [11].

```
VAR num P3_Ans:=3; !Con esta variable determinas la posición del cilindro amarillo y cuantas veces hay que ejecutar el while.
                   !En este caso está en la posición dos.
WHILE P3_Ans>0 DO
```

Figura 6.123 Fragmento de código RAPID P3.

- VAR

VAR indica que la que el código que se va a escribir es para un dato variable, después de VAR hay que indicar que tipo de variable es, en el caso de esta práctica se ha usado una variable numérica, por tantos después de VAR se ha tenido que incluir “num”, después se indica el nombre de la variable y su valor inicial, esto se hace con “:=”.

En el caso de este programa se debe de escribir un número que indique cuantas veces se debe de coger el cilindro en la posición más baja para que el cilindro amarillo pase a la posición número 1, si se encuentra en la posición 1 todos los cilindros cambian de posición hasta que se vuelva a cumplir la condición de que el cilindro amarillo ocupa la posición nº1(Lo que equivale a un valor de P3_Ans=4).

La estructura es:

VAR num nombre variable:=valor;

- WHILE

WHILE es un elemento en RAPID que permite ejecutar cíclicamente una serie de comandos hasta que unas condiciones se cumplen, en nuestro caso la condición a cumplir es las de que P3_Ans, (la variable que indica el número de veces que hay que tomar el cilindro) debe de ser mayor que 0.

La estructura de WHILE es:

WHILE "Condición" DO

"Código"

ENDWHILE

Como es las anteriores prácticas el código se muestra en el apartado Anexos.

### 6.2.5 Práctica nº4: Apilado de piezas.

La práctica de apilado de piezas se creó en el trabajo anterior para que el alumno programase unos movimientos más complejos, en un principio se mostraba el código de apilamiento de las vigas en un orden determinado mediante FlexPendant y el alumno debía de crear un programa en el que se volviera a la posición inicial, la posición inicial y el resultado son visibles en la Figura 6.124.

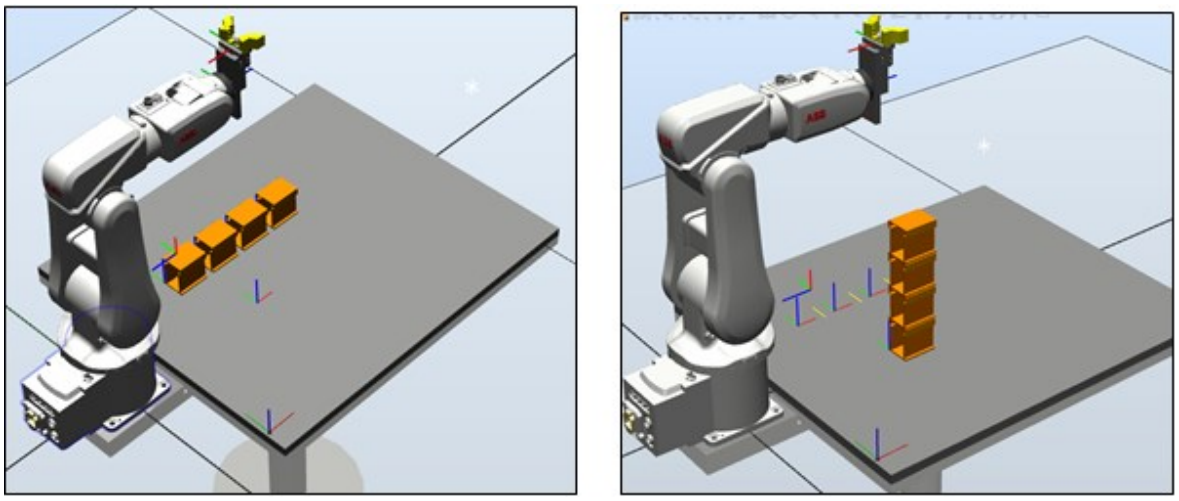

Figura 6.124 a) Posición inicial P4. b) Posición final P4.

Como ocurre con la práctica nº3 no ha sido posible simular la introducción de datos mediante FlexPendant por lo que se ha programado un código en el que se introduce esa información en las cuatro primeras líneas de código.

Además, para evitar tener un módulo de programación muy extenso se ha dividido en dos partes, tal y como hacía el anterior trabajo, creando un código para el apilamiento y otro para volver a situar las vigas en su posición.

Para esta práctica se han creado muy pocos puntos en el plano de trabajo, la razón de ello es que se usará esta práctica para introducir una serie de comandos y códigos RAPID típicos de FlexPendant en el módulo de programación.

Anteriormente se ha comentado que es más sencillo la creación de puntos en el programa RobotStudio que en la programación Gestual mediante el uso de FlexPendant y por ello hasta ahora se han creado todos los puntos necesarios para la simulación de los movimientos en vez de usar comandos en el código de RAPID, en esta práctica se mostrará cómo se haría en caso de usar estos códigos.

Antes de indagar en l programación de esta práctica se explicará los puntos creados en el plano de trabajo.

Solo se han creado dos puntos configurados en esta práctica, la de agarre de viga de la posición más cercana al brazo (Agarre realizado visible en la Figura 6.125, el agarre es con el área situada en medio del accesorio de agarre, esto se debe a que la anchura de la Viga es de 60mm) y la posición de reposo del brazo, que sirve de punto de origen para los dos programas creados.

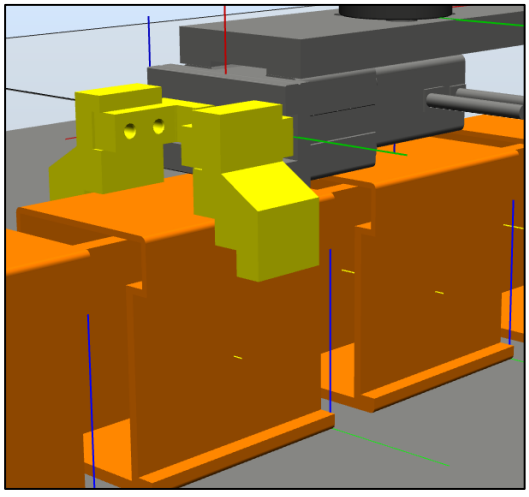

Figura 6.125 Agarre de Viga en P4.

Hay más puntos creados, pero su uso se ha limitado a ubicar las geometrías en el plano de trabajo, estos puntos son las esquinas inferiores izquierda de los cuadrados visibles en la Figura 6.126.

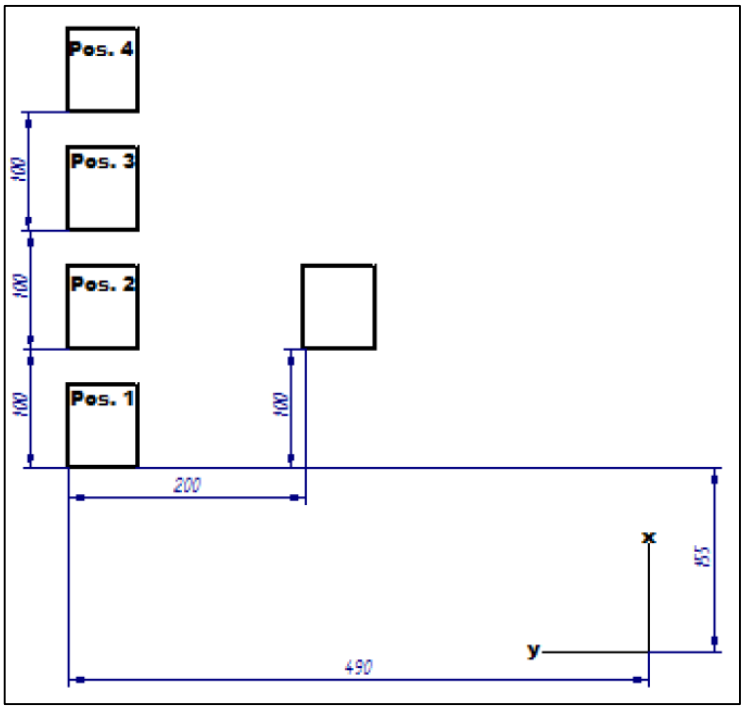


Figura 6.126 Esquema de posicionamiento de objetos en P4 [20].

El punto creado en la zona de apilamiento tiene como uso la comprobación de que se apilan los elementos en el punto adecuado.

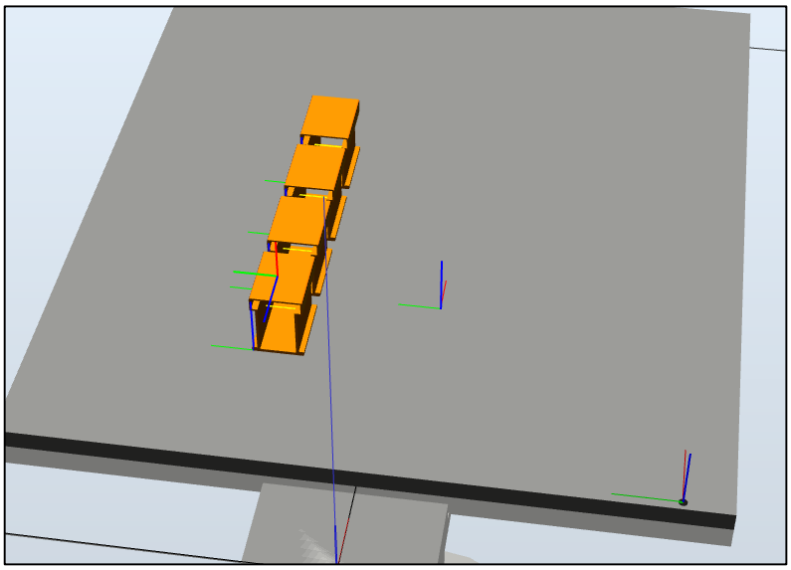

Figura 6.127 Puntos creados P4.

Creados estos puntos se debe colocar las Vigas en las posiciones indicadas de la forma que se observa en la Figura 6.127.

El posicionamiento de las vigas se ha realizado de la misma forma que se hizo con la viga individual de la práctica nº2, con la diferencia de que esta viga ha sido modificada sustrayendo un plano de trabajo, el de mayor área.

Se ha eliminado este sensor porque al realizar el apilamiento estos sensores detectarían a las demás vigas, generando una serie de falsos positivos en la señal pillada que no son deseable.

El procedimiento que sigue es el mismo que en las anteriores prácticas, comprobación de la accesibilidad a los puntos, configuración de los mismo, creación de las entradas y salidas de señal del controlador si son necesarias, configurar la lógica de estación, sincronizar con un módulo de programación, etc.

La lógica de estación de esta práctica tiene la forma indicada en la Figura 6.128.

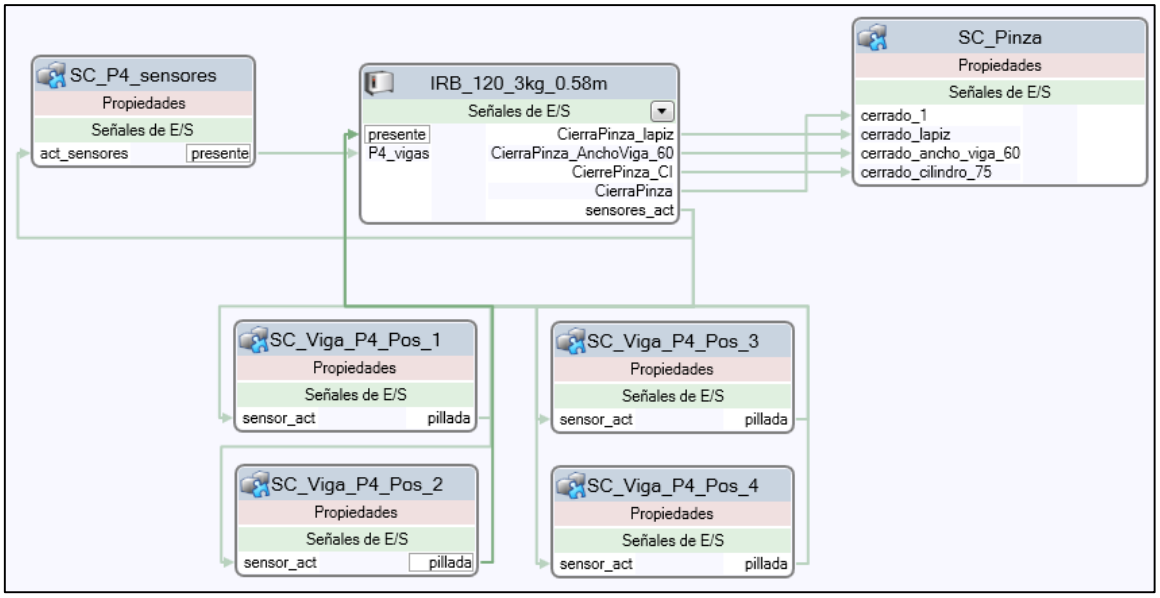


Figura 6.128 Lógica de estación P4.

Todos esos pasos han sido explicados con anterioridad, la única diferencial con las anteriores prácticas reside en su código de programa en RAPID.

Los dos códigos de estas prácticas serán visibles en su totalidad en el apartado anexo, sin embargo, ahora se van a explicar los elementos de programación que son nuevos, siendo estos Offs, IF y VAR robtarget:

- IF

  El comando IF es un elemento que como WHILE ejecuta un código si se cumplen unas condiciones, pero a diferencia de WHILE, IF solo las ejecuta una vez, si no se cumple la condición el programa ignora el código dentro de IF y pasa a las siguientes líneas de código.

  La estructura de IF es:

  IF condición THEN

  Código

  ENDIF

- VAR robtarget

  Como ocurre con VAR num este código indica que se va a incluir una variable en el programa que va a cambiar de valor dentro del mismo.

  Cuando VAR tiene a continuación robtarget se indica que el dato que se va a incluir es el dato de un punto dentro del plano de trabajo (Siendo rob una contracción de robot y target se traduce por destino).

  La estructura de VAR robtarget es la siguiente:

  VAR robtarget nombre del punto;

- Offs

  Estamos acostumbrados a usar herramientas tipo offset para situar puntos dentro del programa RobotStudio, pero también es posible hacerlo directamente dentro de un módulo de programación.

  Con la herramienta offset y VAR robtarget ha sido posible crear los puntos de esta práctica.

  Para crear los puntos se añaden distancias en x, y, z a un punto ya existente, de la forma:

  #####1:=Offs (####2,x,y,z)

  Siendo #####1 el nombre del punto que se desea crear y #####2 el nombre del punto que se le van a añadir las distancias, dentro de x,y,z es posible tanto indicar un valor fijo como añadir un cálculo que requiera de una variable numérica.

  Ejem:

  P4_Origen:=Offs(P4_Coger,0,0,100);

```
P4_coger_posicion:=Offs(P4_coger,(P4_posicion-1)*100,0,0);
```

Usando estos elementos y los ya conocidos es posible realizar la primera parte de la práctica, en la cual se realiza el apilado de las vigas en el orden indicado al inicio del programa mientras el componente inteligente de P4_sensores no indica que sigue habiendo vigas en la mesa.

El segundo programa usado es el que parte del apilado y debe devolver las vigas a su posición, para ello en las primera cuatro líneas de código se indica cual es la posición a la que se debe de dejar a las vigas, empezando por la superior y terminando con la que está en contacto con la mesa.

Para simplificar el código se ha hecho uso de contadores para saber cuándo se han colocado todas las vigas, nada impediría el uso de otro componente inteligente como P4_sensores o la modificación de este para que de otra señal solo cuando los cuatro sensores que lo componen detecten un elemento.

### 6.2.6 Práctica nº5: Apilamiento de pirámide.

Originalmente esta práctica iba a hacer uso del elemento “Random” dentro de los componentes inteligentes, “Random” genera de manera aleatoria un número dentro de un intervalo dado, gracias a esta función se planeaba crear una práctica que generara tres elementos aleatoriamente con posibles tamaños distintos, para permitir usar la capacidad de los sensores de determinar que pieza se está detectando y así usar un tipo de agarre u otro, además de luego apilar las piezas según su tamaño en una estructura piramidal.

Sin embargo, no se pudo realizar esta práctica de la forma comentada por una razón en particular.

Esta razón es la disposición de los sensores usada, los sensores han sido colocados en los componentes movidos, no en la pinza, esto implica que los elementos movidos con componentes inteligentes que se deben de conectar en la lógica de la estación.

La lógica de la estación debe de ser preparada antes de la simulación y no se puede modificar durante la misma, lo que impide que sea posible agregar componentes inteligentes en la estación durante la simulación y que sean funcionales.

Por tanto, se ha decidido actuar como en las prácticas nº3 y 4, en las cuales las posiciones de los elementos se decidieron cuando se colocaban los elementos en la estación.

La práctica consistirá en colocar tres cajas de distintos tamaños (80,75 y 60) paralelos y con los ejes en una misma recta, cuando se inicie la simulación el robot determinará que piezas ocupan cada lugar y las colocará en un orden de menor a mayor en una posición 100mm más alejada del brazo, para finalmente apilar las piezas en forma de pirámide en un punto concreto.

Las posiciones de interés son visibles en la Figura 6.129.

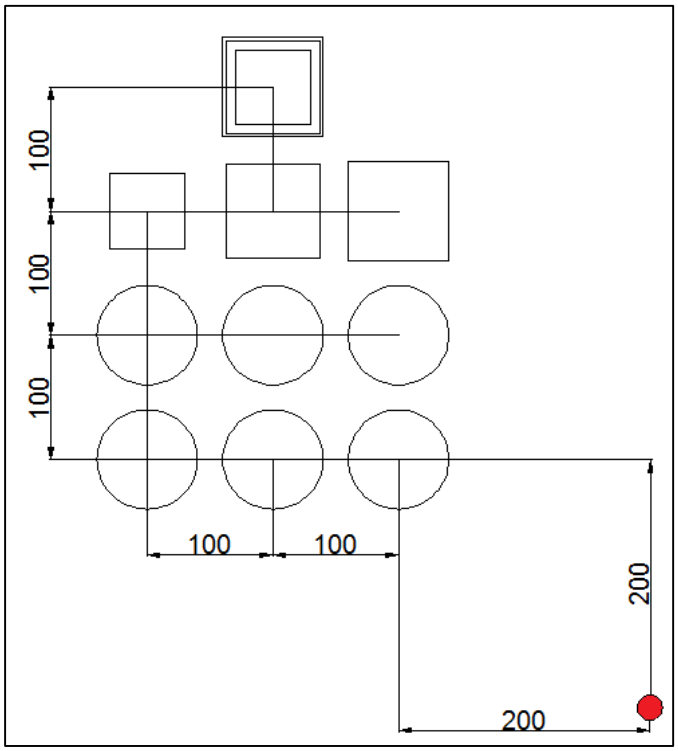


Figura 6.129 Esquema de posición de objetos P5.

El punto rojo representa el origen del plano de trabajo y los círculos y planos representan a las piezas, se puede observar que la línea inferior todos son del tamaño máximo (80mm), esto se hace para mostrar que todas las combinaciones de posiciones iniciales son posibles.

El componente inteligente denominado Sc_P5_componentes, es el componente inteligente que rige el movimiento de las cajas, ya que estas como la viga solo dan una señal de salida al ser cogidas.

El SC_P5_componentes tiene la tarea de primero mover todos los elementos detectados 100mm hacia delante, esto se hace para poder accionar los sensores de la segundada línea y que no detecten nada, evitando así que algún sensor quede atascado con el valor de una simulación previa.

La segunda línea determinará las posiciones que ocupan las cajas y permitirá al robot conocer qué apertura debe de usar para el agarre de las piezas y la posición destino de las mismas.

Los puntos creados son los visibles en la Figura 6.130, cada caja tiene dos puntos, el de agarre y el de aproximación que se encuentra elevado en el eje z, como la pirámide siempre tiene las mismas posiciones se ha podido crear unos puntos fijos para colocar las piezas.

Durante la prueba de la simulación se encontró un problema con los puntos creados, estos al realizar los movimientos generaban choque entre las piezas, entre las opciones disponibles se decidió que como el error era homogéneo (Se debían elevar los puntos de agarre escasos milímetros) se decidió usar los comandos Offs en el programa RAPID para añadir esta distancia.

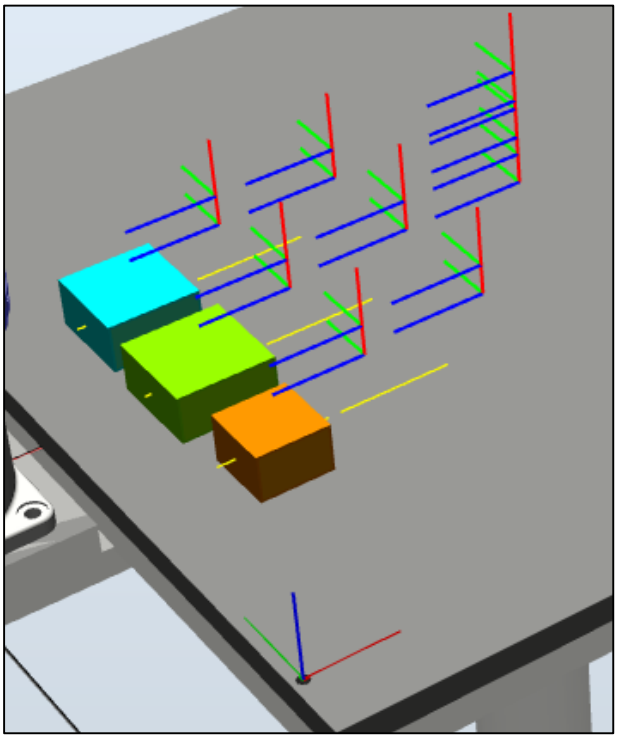

Figura 6.130 Puntos creados en plano de trabajo P5.

El procedimiento a seguir es el mismo que en las demás prácticas, aunque en esta hay que tener en cuenta que, si se deben de crear una gran cantidad de entradas en el controlador, quedando luego una lógica de estación como en la Figura 6.131, la señal de reseteo reset_sist tenía como uso inicial dar un valor 0 a todas las salidas que indican posicionamientos, pero se comprobó que no era necesario.

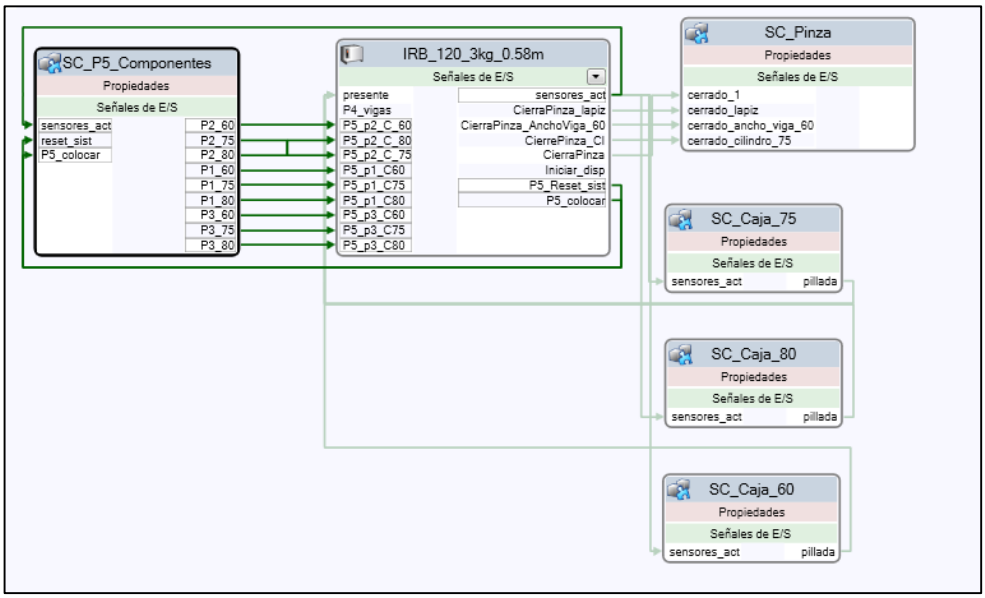


Figura 6.131 Lógica de estación Práctica 5.

El código RAPID generado para esta estación se encuentra en el apartado anexo y no hay ningún elemento nuevo que no se encuentre en ninguna de las anteriores prácticas.

En la Figura 6.132 a, b, c, d, se pueden observar los distintos pasos en la simulación de la práctica.

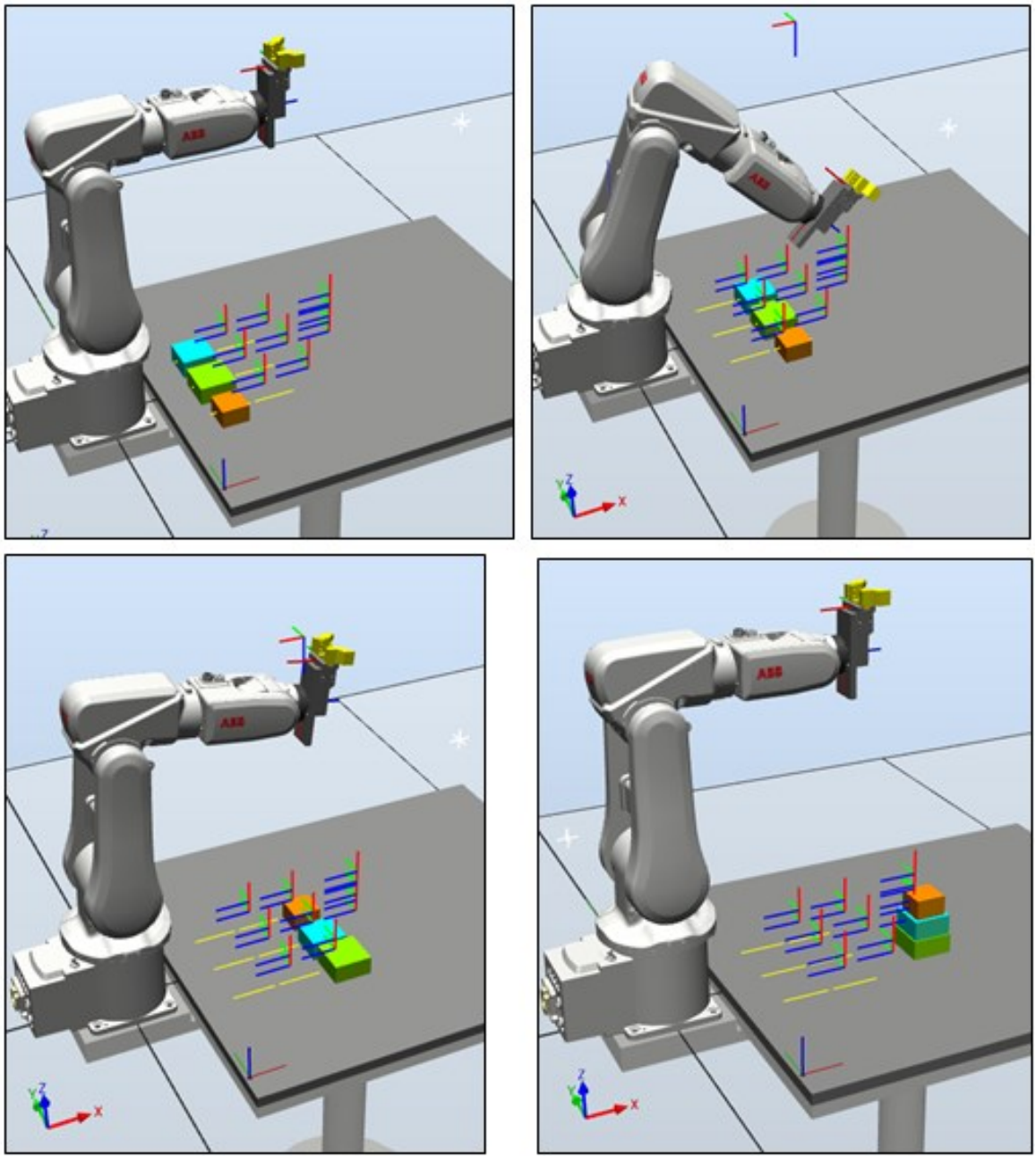

Figura 6.132 a) Posición inicial P5. b) Segundo paso P5. c) Tercer paso P5. d) Posición final P5.

Antes de iniciar la simulación es aconsejable comprobar que los elementos GetParent de los componentes inteligentes Cajas y componente inteligente SC_P5_componentes, tienen definido el objeto, en el caso de las cajas el accesorio pinza-brazo y en SC_P5_componentes SC_Caja_60_75_80 respectivamente.

# 7. Discusión y Conclusiones.

Este trabajo ha sido el primero en usar las herramientas de creación de ambientes tridimensionales virtuales disponibles en el campus (RobotStudio), abriendo las puertas a un uso didáctico más intensivo del Robot disponible.

El empleo de este trabajo permitiría un uso individual de robots virtuales por parte del alumnado, además el uso del programa RobotStudio en el ámbito didáctico proporciona una mayor versatilidad para planificar prácticas en la asignatura de ROBOTS.

Como resultados de este TFG cabe destacar:

- Se ha llevado a cabo el modelado de los elementos tridimensionales elementos usados, incluyendo los que conforman la estación en formatos aceptados por el programa informático RobotStudio.
- Se ha creado y configurado el elemento Pinza, el cual no se encuentra dentro de las opciones del programa RobotStudio.
- Se ha implementado una guía que indica los pasos para la configuración de una herramienta.
- Creación de los componentes inteligentes necesarios para la realización de las prácticas realizadas
- Se ha realizado una guía que indica los pasos a seguir para crear un componente inteligente y como modificarlo si es necesario.
- Se ha elaborado una guía para la creación de las estaciones virtuales necesarias para la creación de las prácticas realizadas en años anteriores.

# 8. Trabajos Futuros.

Los posibles trabajos futuros que amplíen lo realizado en este trabajo son a múltiples y variados:

- Implementación de sensores.

Como se ha comentado en este trabajo la decisión de incluir los sensores de la estación en las geometrías movidas y no en la pinza se realizó por la inexistencia de sensores en la pinza existente en el laboratorio.

En este trabajo se ha comprobado que esta decisión no es una idea acertada, debería estudiarse la posibilidad de incluir los sensores en la herramienta pinza disponible en el laboratorio, permitiendo crear estaciones virtuales no solo realistas respecto al laboratorio sino prácticas.

- Análisis de sensores.

Existe una gran variedad de sensores de los que se pueden hacer uso en una célula robotizada. En el programa RobotStudio no se indica distintos tipos de sensores, solo indica un área que detecta o no distintos elementos.

Un estudio de los sensores aplicables en una célula como el laboratorio resultaría de interés a la hora de diseñar células virtuales, permitiría conocer las soluciones viables en un espacio real.

- Volcado de las simulaciones.

Debido a limitaciones de tiempo en este trabajo no se ha profundizado en el volcado de una simulación en el robot disponible.

Un posible trabajo futuro sería el estudio de las limitaciones del programa a la hora de crear células con elementos ajenos a la compañía ABB (Creadora del software RobotStudio) y estudiar la viabilidad de un volcado total o parcial de una simulación y como solventar estas limitaciones.

- Creación de nuevos accesorios.

A la herramienta pinza disponible en el laboratorio ya se le han creado ya accesorios para mejorar el agarre de elementos de distintas dimensiones.

Esto no impide que en un futuro se realice el diseño de otros accesorios o la creación de accesorios para herramientas que disponga el campus en un futuro.

## 9. Bibliografía.

[1] Asociación Española de Normalización y Certificación. Robots Manipuladores industriales. Vocabulario ISO 8373:2012.

[2] Robot Gargantua. http://cyberneticzoo.com/robots/1937-the-robot-gargantua-bill-griffith-p-taylor-australiancanadian/ [Consultado 10/5/17].

[3] Wikipedia, "Bill" Griffith P. Taylor https://en.wikipedia.org/wiki/Industrial_robot [Consultado 10/5/17].

[4] Wikipedia, Unimation https://es.wikipedia.org/wiki/Unimation [Consultado 10/5/17].

[5] Wikipedia, Unimate https://es.wikipedia.org/wiki/Unimate [Consultado 11/5/17].

[6] Web Robotics online, https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/The-Business-of-Automation-Betting-on-Robots/content_id/6076 [Consultado 13/5/17].

[7]Simulador robótica. https://sites.google.com/site/proyectosroboticos/Descargar-Simuladores [Consultado 7/5/17].

[8] Simulador robótica. V-rep https://robologs.net/2016/01/22/vrep-simulacion-de-robots-virtuales/ [Consultado 7/5/17].

[9] Web ABB España. http://new.abb.com/es [Consultado 7/5/17].

[10] ABB Group, cambios en software RobotStudio 6.04 https://library.e.abb.com/public/8e57c5eb5611448d9b6b0f9527e17c86/ReleaseNotesRobotStudio6-04-00-01.pdf [Consultado 1/6/17].

[11] Manual Usuario, RobotStudio http://www.infoplc.net/files/descargas/abb_robotica/infoplc_net_3hac032104-005_revd_es_.pdf [Consultado 5/5/17].

[12] CRAIG, John J. *Robotica.* 3º edición. Mexico: Pearson, 2006. ISBN 9786074426020

[13] http://fabryka-robotow.pl/2015/01/programming-languages-to-control-robot/ [Consultado 4/5/17]

[14] Lenguajes de programación en robots. https://revistadigital.inesem.es/gestion-integrada/mejor-lenguaje-estudiar-programacion-robot/ [Consultado 5/6/17].

[15] Medios de programación de robots. http://www.monografias.com/trabajos3/progrob/progrob.shtml [Consultado 5/6/17].

[16] Video tutoriales de uso RobotStudio
https://www.youtube.com/user/alonsoprofe/videos [Consultado 3/3/17].
[17] Web compañía RightHand Robotics. https://www.righthandrobotics.com/ [Consultado 4/6/17].
[18] ABB Group. Datos técnicos robot IRB 120.
https://library.e.abb.com/public/3bd625bab3c7cae1c1257a0800495fac/ROB0149EN_D_LR.pdf [Consultado 7/6/17]
[19] Corporación SMC, 2017 https://www.smc.eu [Consultado 15/3/17].
[20] Puesta en marcha de brazo robótico y desarrollo de aplicaciones, Trabajo fin de grado de Axel José Córdoba López, 2016.
[21] Manual de programación en RAPID.
http://egela.oteitzalp.org/pluginfile.php/5828/mod_resource/content/1/RAPID%20Manual%20operador.pdf [Consultado 10/6/17].

# 10. Anexos.

En este apartado se van a presentar los códigos RAPID usados para la simulación de las distintas prácticas creadas en este trabajo, además los códigos incluyen notas explicativas.

Los códigos mostrados son solo los programas, en el módulo en el que se encuentre un programa deben de estar los puntos definidos en el plano de trabajo como constantes al principio del módulo, tal y como se muestra en la Figura 10.1.

T_ROB1/Module1

```
MODULE Module1
    CONST robtarget reposo:=[[273.8,344.126,690.15],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget posicionar_1:=[[373.8,344.126,540.15],[0,0,1,0],[0,0,0,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget posicionar_2:=[[373.8,344.126,340.15],[0.707106781,0,-0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget posicionar_3:=[[369,344.536,209.1],[0.707106781,0,-0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget posicionar_4:=[[369,344.536,109.1],[0.707106781,0,-0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget coger_lapiz_0:=[[137.5,100,512],[0.707106781,0,-0.707106781,0],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget coger_lapiz_2:=[[-62.5,0,113],[0.707106781,0,-0.707106781,0],[-2,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget inicio:=[[-22.5,0,212],[0.707106781,0,-0.707106781,0],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget reposo_2:=[[273.8,344.126,690.15],[0,0,1,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget coger_lapiz_1:=[[-62.5,0,312],[0.707106781,0,-0.707106781,0],[-2,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget posicion_2:=[[255,490,140],[0.707106781,0,-0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget coger_lapiz_3:=[[-26.5,0,112.7],[0.707106781,0,-0.707106781,0],[-1,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget posicion_1:=[[455.548,344.536,200],[0.707106781,0,-0.707106781,0],[0,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Figura 10.1 Código de constantes, puntos definidos en plano de trabajo.

## 10.1 Práctica nº1: Dibujar en Plano.

```
75
76  PROC P_1_Lapiz()
77      SetDO CierraPinza_AnchoViga_60, 0;  !Todos los valores de entrada de la herramienta
78      SetDO CierraPinza, 0;               !pinza se inicializan a 0 para asegurar una posición
79      SetDO CierrePinza_CI, 0;            !inicial de apertura máxima, es en principio una medida
80      SetDO CierraPinza_lapiz, 1;         !de seguridad que no es necesaria para el funcionamiento
81      SetDO CierraPinza_lapiz, 0;         !del código , pero si aconsejable, al darle valor 1 a la
82      SetDO sensores_act, 0;              !señal que primero se va a usar podemos comprobar que
83      SetDO sensores_act, 1;              !funciona correctamente.
84      !--------
85      MoveJ reposo_2,v200,fine,T_pinza\WObj:=Workobject_WORK_P1;      !Este es el movimiento desde
86      MoveJ coger_lapiz_0,v100,z50,T_pinza\WObj:=Workobject_WORK_P1;  !el punto de reposo hasta
87      MoveJ coger_lapiz_1,v100,z50,T_pinza\WObj:=Workobject_WORK_P1;  !la toma del lápiz.
88      MoveL coger_lapiz_2,v100,fine,T_pinza\WObj:=Workobject_WORK_P1;
89      MoveL coger_lapiz_3,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
90      !--------
91      WaitTime 1;                         !Se añade esta pausa de un segundo para permitir
92      SetDO CierraPinza_lapiz, 1;         !al controlador y al robot  llegar al punto de
93      WaitDI presente, 1;                 !interes a la vez, esta sincronización permite que
94      WaitTime 2;                         !la cogida de piezas sea óptima, además de asegurar
95      !--------                           !que en caso de fallo se sepa que el error no
96      !--------                           !se encuentra en el código.
97      MoveL inicio,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;         !Este es el movimiento de escritura
98      MoveJ posicion_1,v50,z50,T_pinza\WObj:=Workobject_WORK_P1;      !de escritura, incluye puntos
99      MoveL posicion_2,v50,z50,T_pinza\WObj:=Workobject_WORK_P1;      !con z>0 para poder escribir iniciales
100     MoveL Target_P11,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;     !independientes.
101     MoveL Target_P12,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
102     MoveL Target_P22,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
103     MoveL Target_P32,v50,z15,T_pinza\WObj:=Workobject_WORK_P1;
104     MoveL Target_P31,v50,z15,T_pinza\WObj:=Workobject_WORK_P1;
105     MoveL Target_P21,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
106     MoveL posicion_2,v50,z10,T_pinza\WObj:=Workobject_WORK_P1;
107     MoveL Target_P33,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
108     MoveL posicion_2,v50,z10,T_pinza\WObj:=Workobject_WORK_P1;
109     MoveL Target_P34,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
110     MoveL Target_P14,v50,z20,T_pinza\WObj:=Workobject_WORK_P1;
```

Figura 10.2 Parte I de P_1_Lápiz.

```
110         MoveL Target_P14,v50,z20,T_pinza\WObj:=Workobject_WORK_P1;
111         MoveL Target_P15,v50,z20,T_pinza\WObj:=Workobject_WORK_P1;
112         MoveL Target_P35,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
113         MoveL posicion_1,v50,z10,T_pinza\WObj:=Workobject_WORK_P1;
114         MoveL Target_P24,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
115         MoveL Target_P25,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
116         MoveJ posicion_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
117         MoveJ coger_lapiz_3,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;
118         SetDO CierraPinza_lapiz, 0;         !Como en el anterior se añaden pausas para que el
119         WaitDI presente, 0;                 !controlador y el brazo estén sincronizados.
120         WaitTime 2;
121         MoveL coger_lapiz_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P1;  !Esta parte devuelve el brazo
122         MoveL coger_lapiz_1,v100,fine,T_pinza\WObj:=Workobject_WORK_P1; !al punto de reposo.
123         MoveJ reposo_2,v100,fine,T_pinza\WObj:=Workobject_WORK_P1;
124     ENDPROC
```

Figura 10.3 Parte II de P_1_Lápiz.

## 10.2 Práctica nº2: Reorientar pieza.

```
158 PROC P_2_reorientar()
159     SetDO CierraPinza_lapiz, 0;               !Se inicializan las salidas del controlador.
160     SetDO CierraPinza_AnchoViga_60, 0;
161     SetDO CierrePinza_CI, 0;
162     SetDO CierraPinza, 1;
163     SetDO CierraPinza, 0;
164     SetDO sensores_act, 0;
165     SetDO sensores_act, 1;
166     MoveJ P2_Reposo,v100,fine,T_pinza\WObj:=Workobject_WORK_P2;         !Este es el movimiento del reposo
167     MoveJ P2_posicion_A,v100,fine,T_pinza\WObj:=Workobject_WORK_P2;     !hasta la viga.
168     MoveJ P2_coger_A,v20,fine,T_pinza\WObj:=Workobject_WORK_P2;
169     WaitTime 1;
170     SetDO CierraPinza, 1;       !Cierre de la pinza sincronizando
171     WaitDI presente, 1;         !controlador y brazo con pausas de tiempo.
172     WaitTime 2;
173     MoveJ P2_posicion_A,v20,fine,T_pinza\WObj:=Workobject_WORK_P2;      !Cambio de posición A a B.
174     MoveJ P2_Cambio_orientación_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
175     MoveJ P2_posicion_B,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
176     MoveJ P2_Coger_B,v10,fine,T_pinza\WObj:=Workobject_WORK_P2;
177     WaitTime 1;
178     SetDO CierraPinza, 0;
179     WaitDI presente, 0;
180     WaitTime 2;
181     MoveJ P2_posicion_B,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;      !Cambio de la posición del agarre
182     MoveJ P2_Cambio_orientación_2_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P2; !y agarre para el cambio
183     MoveJ P2_posicion_C,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;              !a la posición C
184     MoveJ P2_Coger_C,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
185     WaitTime 1;
186     SetDO CierraPinza_AnchoViga_60, 1;
187     WaitDI presente, 1;
188     WaitTime 2;
189     MoveJ P2_posicion_C,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;              !Cambio a posición C
190     MoveJ P2_Cambio_orientación_3_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
191     MoveJ P2_posicion_final,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
192     MoveJ P2_final,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
```

Figura 10.4 Parte I P_2_Reorientar.

```
193 WaitTime 1;
194 SetDO CierraPinza_AnchoViga_60, 0;
195 WaitDI presente, 0;
196 WaitTime 2;
197 MoveL P2_posicion_final,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;     !Movimiento a pto de reposo
198 MoveJ P2_Reposo,v150,fine,T_pinza\WObj:=Workobject_WORK_P2;            !Y agarre en posición C
199 MoveJ P2_posicion_final,v150,fine,T_pinza\WObj:=Workobject_WORK_P2;
200 MoveJ P2_final,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
201 WaitTime 1;
202 SetDO CierraPinza_AnchoViga_60, 1;
203 WaitDI presente, 1;
204 WaitTime 2;
205 MoveL P2_posicion_final,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;          !Colocado en posición B
206 MoveJ P2_Cambio_orientación_3_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
207 MoveJ P2_posicion_C,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
208 MoveJ P2_Coger_C,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
209 WaitTime 1;
210 SetDO CierraPinza_AnchoViga_60, 0;
211 WaitDI presente, 0;
212 WaitTime 2;
213 MoveJ P2_posicion_C,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;              !Agarre de la viga para pasar
214 MoveJ P2_Cambio_orientación_2_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;  !a la posición A
215 MoveJ P2_posicion_B,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
216 MoveJ P2_Coger_B,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
217 WaitTime 1;
218 SetDO CierraPinza_AnchoViga_60, 1;
219 WaitDI presente, 1;
220 WaitTime 2;
221 MoveJ P2_posicion_B,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;              !Posicionamiento de la viga en
222 MoveJ P2_Cambio_orientación_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;    !la posición A.
223 MoveJ P2_posicion_A,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
224 MoveJ P2_coger_A,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;
```

Figura 10.5 Parte II P_2_reorientar.

```
225     WaitTime 1;
226     SetDO CierraPinza_AnchoViga_60, 0;
227     WaitDI presente, 0;
228     WaitTime 2;
229     MoveJ P2_posicion_A,v50,fine,T_pinza\WObj:=Workobject_WORK_P2;          !Movimiento al pto de reposo.
230     MoveJ P2_Reposo,v150,fine,T_pinza\WObj:=Workobject_WORK_P2;
231 ENDPROC
```

Figura 10.6 Parte III P_2_reorientar.

## 10.3 Práctica nº3: Dispensador.

El primer programa mostrado es el que se usó en el trabajo anterior, este código requiere de la herramienta FlexPendant, permitiendo al usuario introducir información que le pregunta el código, por razones de limitaciones de programa informático, no se pudo simular esta práctica de la manera planeado, ya que al iniciar un programa con FlexPendant los sensores no actualizan sus valores a tiempo real.

```
52      PROC P3_FLEX()  !este sería el programa en flexpendant
53          VAR num P3_Ans:=0;
54          CONST string P3_Text:="Posición del Cilindro";
55          CONST string P3_V1:="Pos_1";
56          CONST string P3_V2:="Pos_2";
57          CONST string P3_V3:="Pos_3";
58          CONST string P3_V4:="Pos_4";
59      TPReadFK P3_Ans,P3_Text,P3_V1,P3_V2,P3_V3,P3_V4, stEmpty; ! Este es el códico que
60                                                                !pide información al ususario.
61      IF P3_Ans=1 THEN
62          P3_Ans:=5;
63      ENDIF
64      WHILE P3_Ans>1 DO
65          P_3_MOV;
66          P3_Ans:=P3_Ans-1;
67      ENDWHILE
68      ENDPROC
```

Figura 10.7 Programa con uso de FlexPendant.

Como sustituto de este código se creó uno en el que la primera línea de código es un dato modificable según la situación que se quiera llevar a cabo.

```
PROC P_3_MOV()

    VAR num P3_Ans:=3; !Con esta variable determinas la posición del cilindro amarillo
                       !y cuantas veces hay que ejecutar el while, en este caso
                       !está en la posición dos.
    WHILE P3_Ans>0 DO

    SetDO CierraPinza_lapiz, 0;
    SetDO CierraPinza_AnchoViga_60, 0;
    SetDO CierraPinza, 0;
    SetDO CierrePinza_CI, 1;
    SetDO CierrePinza_CI, 0;
    SetDO Iniciar_disp, 0;      !La señal de iniciar dispositivo es crucial para el
    SetDO sensores_act, 0;      !funcionamiento de la simulación, si no existiera esta señal
    SetDO sensores_act, 1;      !al activar los sensores los cilinidros cambiarian de
    SetDO Iniciar_disp, 1;      !posición, esto se debe a que las señales cambian de valores
    !------                     !de manera lineal, la señal de activación permite decidir
    !------                     !el momento en el que quieres que el componente distribuidor
    !------                     !tiene permitido el movimiento de cilindros.
    MoveJ P3_Acercamiento,v100,z10,T_pinza\WObj:=Workobject_WORK_P3;
    MoveL P3_Acercamiento_Pos_INF,v50,fine,T_pinza\WObj:=Workobject_WORK_P3;
    MoveL P3_Coger_CI_INF,v50,fine,T_pinza\WObj:=Workobject_WORK_P3;
    WaitTime 1;
    SetDO CierrePinza_CI, 1;
    WaitDI presente, 1;
    WaitTime 1;
    MoveJ P3_Coger_CI_INF_3,v50,fine,T_pinza\WObj:=Workobject_WORK_P3;
    MoveJ P3_Coger_CI_INF_3_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P3;
    MoveL P3_Acercamiento_Pos_INF,v50,fine,T_pinza\WObj:=Workobject_WORK_P3;
    MoveJ P3_Acercamiento,v50,z10,T_pinza\WObj:=Workobject_WORK_P3;
    WaitTime 2;
    MoveJ P3_Acercamiento_Pos_SUP,v100,fine,T_pinza\WObj:=Workobject_WORK_P3;
    MoveL P3_Coger_CI_SUP,v50,fine,T_pinza\WObj:=Workobject_WORK_P3;
```

Figura 10.8 Parte I P_3_MOV.

```
    WaitTime 1;
    SetDO CierrePinza_CI, 0;
    WaitDI presente, 0;
    WaitTime 1;
    MoveL P3_Acercamiento_Pos_SUP,v100,fine,T_pinza\WObj:=Workobject_WORK_P3;
    MoveJ P3_Acercamiento,v100,z10,T_pinza\WObj:=Workobject_WORK_P3;

    P3_Ans:=P3_Ans-1;   !Sin esta línea de código el WHILE entraría en un bucle infinito
    !-----              !al darle un valor de X a la variable P3_Ans el ciclo WHILE se ejecutará
    !-----              !X veces, en cada ciclo le restamos una unidad y parará cuando X=0
    ENDWHILE
ENDPROC
```

Figura 10.9 Parte II P_3_MOV.

## 10.4 Práctica nº4: Apilado de piezas.

Como se ha comentado con anterioridad este programa sufrió un cambio similar al de la práctica del dispensador por la misma razón, asimismo se ha dividido la práctica en dos programas por la longitud del código, nada impide unirlos en un único programa.

Se empezará mostrando el programa de apilado para después mostrar el de vuelta a la situación inicial.

```
76  PROC P4_Amontonar()
77
78      VAR num coger_1:=3; !Determina que viga coges en primer lugar.
79      VAR num coger_2:=2; !Determina que viga coges en segundo lugar.
80      VAR num coger_3:=1; !Determina que viga coges en tercer lugar.
81      VAR num coger_4:=4; !Determina que viga coges en cuarto lugar.
82
83      VAR robtarget P4_Origen;
84      VAR robtarget P4_coger_posicion;        !Punto de agarre de pieza.
85      VAR robtarget P4_coger_aprox_posicion;  !Punto de aproximación a la pieza a la hora de coger.
86      VAR robtarget P4_coger_alejar_posicion; !Punto de separación de la pieza para evitar choques.
87
88      VAR robtarget P4_dejar;        !Punto de depositar pieza en superficie de mesa, se usa como referencia.
89      VAR robtarget P4_dejar_altura; !Punto de depositar pieza en altura especifica.
90      VAR robtarget P4_dejar_aprox;  !Punto de aproximación a depositar pieza a altura especifica.
91      VAR robtarget P4_dejar_alejar; !Punto de separación de la pieza para evitar choques según la altura.
92
93      VAR num P4_posicion;
94      VAR num P4_altura:=0;          !Altura de dejar pieza, inicial.
95      VAR num Cont:=0;               !Este contador nos permite hacer los 4 movimientos en el while.
96      MoveJ P4_reposo,v200,fine,T_pinza\WObj:=Workobject_WORK_P4;  !Pto de inicio del programa.
97      SetDO CierraPinza_lapiz, 0;
98      SetDO CierraPinza, 0;
99      SetDO CierrePinza_CI, 0;
100     SetDO CierraPinza_AnchoViga_60, 0;
101     SetDO sensores_act, 0;
102     SetDO sensores_act, 1;
103
104     WHILE P4_vigas=1 DO            !Mientras se detectec vigas en sin apilar se aplica el código dentro del WHILE
105         Cont:=Cont+1;
106         IF Cont=1 THEN                      !Los siguientes IF permiten saber que posición se debe de tomar.
107             P4_posicion:=coger_1;           !Los P4_altura no se deben de modificar, inican a que altura
108             P4_altura:=0;                   !de la columna de apilamiento deben de colocarse.
109         ENDIF
110         IF Cont=2 THEN
111             P4_posicion:=coger_2;
112             P4_altura:=80;
113         ENDIF
114         IF Cont=3 THEN
115             P4_posicion:=coger_3;
116             P4_altura:=160;
117         ENDIF
118         IF Cont=4 THEN
119             P4_posicion:=coger_4;
120             P4_altura:=240;
121         ENDIF
```

Figura 10.10 Parte I P4_Amontonar.

```
122         !Ahora se van a definir todos los puntos necesarios para hacer el movimiento dependiendo
123         !del orden que queramos coger las vigas.
124         P4_Origen:=Offs(P4_Coger,0,0,100);                          !Se define el punto como un desplazamiento del
125         P4_coger_posicion:=Offs(P4_coger,(P4_posicion-1)*100,0,0);  !punto de cogida de la primera pieza. Se define
126         P4_coger_aprox_posicion:=Offs(P4_coger_posicion,0,0,15);    !este punto dependiendo de la posición de la pieza
127         !-----                                                      !a coger, separadas entre sí 100 mm.Se define este
128         !-----                                                      !punto como una aproximación a 15 mm de la pieza
129         !-----                                                      !para hacer el movimiento de aproximación con delicadeza.
130         !-----
131         P4_coger_alejar_posicion:=Offs(P4_coger_posicion,0,0,P4_altura+50);   !Se define la altura a la que debe subir
132         !-----                                                                 !la pieza después de ser cogida para evitar
133         !-----                                                                 !choques, siendo mayor cuanto más piezas
134         !-----                                                                 !hay en la columna central.
135         !-----
136         P4_dejar:=Offs(P4_coger,100,-200,0);              !Se define el punto como un desplazamiento
137         !-----                                            !del punto de cogida de la primera pieza.
138         !-----
139         P4_dejar_altura:=Offs(P4_dejar,0,0,P4_altura);  !Se define la altura a la que se deposita el objeto
140         !-----                                          !a partir del punto de depositar pieza en superficie.
141         !-----
142         P4_dejar_aprox:=Offs(P4_dejar_altura,0,0,15);  !Se define este punto como una aproximación a 15 mm
143         !-----                                         !de la pieza para hacer el movimiento de aproximacion con delicadeza.
144         P4_dejar_alejar:=Offs(P4_dejar_altura,0,0,100);
145         !-----
146         MoveJ P4_Origen,v200,fine,T_pinza\WObj:=Workobject_WORK_P4;            !Este es el código de movimiento con los ptos
147         MoveJ P4_coger_aprox_posicion,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;!definidos anteriormente, cada ciclo del WHILE
148         MoveL P4_coger_posicion,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;      !moverá una viga.
149         WaitTime 1;
150         SetDO CierraPinza_AnchoViga_60, 1;
151         WaitDI presente, 1;
152         WaitTime 1;
153         MoveJ P4_coger_alejar_posicion,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
154         MoveJ P4_dejar_aprox,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
155         MoveL P4_dejar_altura,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
156         WaitTime 1;
157         SetDO CierraPinza_AnchoViga_60, 0;
158         WaitDI presente, 0;
159         WaitTime 1;
160         MoveJ P4_dejar_aprox,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
161         MoveJ P4_reposo,v200,z100,T_pinza\WObj:=Workobject_WORK_P4;
162     ENDWHILE
163
164 ENDPROC
```

Figura 10.11 Parte II P4_Amontonar.

Esto da fin al programa de apilado, a continuación, se muestra el programa que vuelve a poner las vigas en sus posiciones, originales, para ello hay que indicar como en el programa anterior las posiciones que ocupan las vigas en las primeras cuatro líneas de código.

```
PROC P4_Desmontar()

    VAR num dejar_1:=4; !Determina en que posición dejas la viga en primer lugar.
    VAR num dejar_2:=1; !Determina en que posición dejas la viga en segundo lugar.
    VAR num dejar_3:=2; !Determina en que posición dejas la viga en tercer lugar.
    VAR num dejar_4:=3; !Determina en que posición dejas la viga en cuarto lugar.

    VAR robtarget P4_Origen;
    VAR robtarget P4_coger_posicion;
    VAR robtarget P4_coger_aprox_posicion;
    VAR robtarget P4_coger_alejar_posicion;

    VAR robtarget P4_dejar;
    VAR robtarget P4_dejar_altura;
    VAR robtarget P4_dejar_aprox;
    VAR robtarget P4_dejar_alejar;

    VAR num P4_posicion;
    VAR num P4_altura:=0;
    VAR num Cont:=4;
    MoveJ P4_reposo,v500,z100,T_pinza\WObj:=Workobject_WORK_P4;
    SetDO CierraPinza_lapiz, 0;
    SetDO CierraPinza, 0;
    SetDO CierrePinza_CI, 0;
    SetDO CierraPinza_AnchoViga_60, 0;
    SetDO sensores_act, 0;
    SetDO sensores_act, 1;

    WHILE Cont>0 DO
        IF Cont=4 THEN                      !Los siguientes IF permiten saber
            P4_posicion:=dejar_1;           !que posición se debe de tomar,
            P4_altura:=240;                 !los P4_altura no se deben de modificar,
        ENDIF                               !inican a que altura de la columna
        IF Cont=3 THEN                      !deben de tomarse las vigas.
            P4_posicion:=dejar_2;
            P4_altura:=160;
        ENDIF
        IF Cont=2 THEN
            P4_posicion:=dejar_3;
            P4_altura:=80;
        ENDIF
        IF Cont=1 THEN
            P4_posicion:=dejar_4;
            P4_altura:=0;
        ENDIF
```

Figura 10.12 Parte III P4_Amontonar.

En este caso en el WHILE no se ha usado sensores para indicar cuantas veces debe de realizarse el programa, se podría crear un sensor con un componente inteligente para hacer una acción similar al usado en el primer programa o modificar el existente, no se hizo en este caso por lo que se podría clasificar como error humano, ya que solo hay que incluir otra señal de salida al componente inteligente y una puerta lógica tipo NOT con una señal de entrada proveniente de la última puerta lógica del componente, esto daría un sistema con dos señales de salida, una tendría valor 1 mientras cualquiera de los sensores detecte un objeto, y otra que tendría valor 1 si hay algún sensor detectando un objeto.

```
211         ! Ahora se van a definir todos los puntos necesarios para hacer el movimiento
212         !dependiendo del orden que queramos coger las vigas, se hace de la misma
213         !forma que el caso anterior.
214         P4_Origen:=Offs(P4_Coger,0,0,100);
215         P4_coger_posicion:=Offs(P4_coger,(P4_posicion-1)*100,0,0);
216         P4_coger_aprox_posicion:=Offs(P4_coger_posicion,0,0,15);
217
218         P4_coger_alejar_posicion:=Offs(P4_coger_posicion,0,0,P4_altura+50);
219
220         P4_dejar:=Offs(P4_coger,100,-200,0);
221
222         P4_dejar_altura:=Offs(P4_dejar,0,0,P4_altura);
223         P4_dejar_aprox:=Offs(P4_dejar_altura,0,0,15);
224         P4_dejar_alejar:=Offs(P4_dejar_altura,0,0,100);
225
226         MoveJ P4_dejar_aprox,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
227         MoveL P4_dejar_altura,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
228         WaitTime 1;
229         SetDO CierraPinza_AnchoViga_60, 1;
230         WaitDI presente, 1;
231         WaitTime 1;
232         MoveJ P4_dejar_aprox,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
233         MoveJ P4_coger_alejar_posicion,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
234         MoveL P4_coger_posicion,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
235         WaitTime 1;
236         SetDO CierraPinza_AnchoViga_60, 0;
237         WaitDI presente, 0;
238         WaitTime 1;
239         MoveJ P4_coger_aprox_posicion,v50,fine,T_pinza\WObj:=Workobject_WORK_P4;
240         MoveJ P4_Origen,v200,fine,T_pinza\WObj:=Workobject_WORK_P4;
241         MoveJ P4_reposo,v200,z100,T_pinza\WObj:=Workobject_WORK_P4;
242         Cont:=Cont-1;
243
244     ENDWHILE
```

Figura 10.13 Parte IV P4_Amontonar.

## 10.5 Práctica nº5: Apilamiento de Pirámide.

En este caso como los dos anteriores se han tenido que colocar las piezas en una configuración de posiciones que no es única, a diferencias de los anteriores esta práctica no necesita un aporte de información externa para llevarse a cabo.

```
PROC P_5_Pirámide()

    VAR robtarget P5_Dejar_pto_1_prev_75_80; !Estos puntos se pueden definir tanto en este código
    VAR robtarget P5_Dejar_pto_2_prev_75_80; !o como cualquier otro punto desde la pestaña de inicio
    VAR robtarget P5_Dejar_pto_3_prev_75_80; !En este caso su uso es corregir un error de altura.
    VAR robtarget P5_Dejar_pto_1_75_80;
    VAR robtarget P5_Dejar_pto_2_75_80;
    VAR robtarget P5_Dejar_pto_3_75_80;

    P5_Dejar_pto_1_prev_75_80:=Offs(P5_Dejar_pto_1_prev, 0, 0, 8);
    P5_Dejar_pto_2_prev_75_80:=Offs(P5_Dejar_pto_2_prev, 0, 0, 8);
    P5_Dejar_pto_3_prev_75_80:=Offs(P5_Dejar_pto_3_prev, 0, 0, 8);
    P5_Dejar_pto_1_75_80:=Offs(P5_Dejar_pto_1, 0, 0, 8);
    P5_Dejar_pto_2_75_80:=Offs(P5_Dejar_pto_2, 0, 0, 8);
    P5_Dejar_pto_3_75_80:=Offs(P5_Dejar_pto_3, 0, 0, 8);

    SetDO CierraPinza_lapiz, 0;
    SetDO CierraPinza_AnchoViga_60, 0;
    SetDO CierrePinza_CI, 0;
    SetDO CierraPinza, 0;
    WaitTime 1;
    SetDO sensores_act, 0;      !Al activar, desacrivar y volver a activar los sensores, estamos asegurando que
    SetDO sensores_act, 1;      !la línea de sensores que determina que elemento se está detectando no de falsos
    SetDO sensores_act, 0;      !positivos o problemas en general.
    WaitTime 1;
    SetDo P5_colocar, 0;        !Esto activa la primera linea de sensores de la mesa, lo que provoca el movimiento
    SetDo P5_colocar, 1;        !de las cajas a la segunda línea de sensores, esto se realiza porque queremos
    !-------                    !que los sensores que ocupan la segunda línea tenga un valor inicial de 0.
    WaitTime 1;
    SetDO sensores_act, 0;      !Esto vuelve a activar los sensores de la seguda línea de sensores en la mesa
    SetDO sensores_act, 1;      !y los planos en las cajas, en este caso al ya existir una geometría que detectar
    !------                     !los sensores de la segunda linea comparan y determinan que elementos hay en cada posición.
    MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;

    ! Pto 1 con esta serie de IF´s se está seleccionando que movimiento de pinza se debe de realizar
    !si en un punto determinado hay una caja u otra.
```

Figura 10.14 Parte I P_5_Pirámide.

```
291     IF P5_p1_C60=1 THEN
292         MoveJ P5_Dejar_pto_1_prev,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
293         MoveJ P5_Dejar_pto_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
294         WaitTime 0.5;
295         SetDO CierraPinza_AnchoViga_60, 1;
296         WaitDI presente, 1;
297         WaitTime 0.5;
298         MoveJ P5_Dejar_pto_1_prev,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
299         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
300         MoveJ P5_Prep_60_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
301         MoveL P5_Prep_60,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
302         WaitTime 0.5;
303         SetDO CierraPinza_AnchoViga_60, 0;
304         WaitDI presente, 0;
305         WaitTime 0.5;
306         MoveL P5_Prep_60_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
307         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
308     ENDIF
309
310     IF P5_p1_C75=1 THEN
311         MoveJ P5_Dejar_pto_1_prev_75_80,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
312         MoveJ P5_Dejar_pto_1_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
313         WaitTime 0.5;
314         SetDO CierrePinza_CI, 1;
315         WaitDI presente, 1;
316         WaitTime 0.5;
317         MoveJ P5_Dejar_pto_1_prev_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
318         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
319         MoveJ P5_Prep_75_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
320         MoveL P5_Prep_75,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
321         WaitTime 0.5;
322         SetDO CierrePinza_CI, 0;
323         WaitDI presente, 0;
324         WaitTime 0.5;
325         MoveL P5_Prep_75_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
326         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
327     ENDIF
```

Figura 10.15 Parte II P_5_Pirámide.

```
329     IF P5_p1_C80=1 THEN
330         MoveJ P5_Dejar_pto_1_prev_75_80,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
331         MoveJ P5_Dejar_pto_1_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
332         WaitTime 0.5;
333         SetDO CierraPinza, 1;
334         WaitDI presente, 1;
335         WaitTime 0.5;
336         MoveJ P5_Dejar_pto_1_prev_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
337         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
338         MoveJ P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
339         MoveL P5_Prep_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
340         WaitTime 0.5;
341         SetDO CierraPinza, 0;
342         WaitDI presente, 0;
343         WaitTime 0.5;
344         MoveL P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
345         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
346     ENDIF
347     ! Pto 2 igual que en el punto Pto 1 esta serie de if tienen el mismo propósito excepto para el punto intermedio
```

Figura 10.16 Parte III P_5_Pirámide.

```
348     IF P5_p2_C_60=1 THEN
349         MoveJ P5_Dejar_pto_2_prev,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
350         MoveJ P5_Dejar_pto_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
351         WaitTime 0.5;
352         SetDO CierraPinza_AnchoViga_60, 1;
353         WaitDI presente, 1;
354         WaitTime 0.5;
355         MoveJ P5_Dejar_pto_2_prev,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
356         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
357         MoveJ P5_Prep_60_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
358         MoveL P5_Prep_60,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
359         WaitTime 0.5;
360         SetDO CierraPinza_AnchoViga_60, 0;
361         WaitDI presente, 0;
362         WaitTime 0.5;
363         MoveL P5_Prep_60_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
364         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
365     ENDIF
366
367     IF P5_p2_C_75=1 THEN
368         MoveJ P5_Dejar_pto_2_prev_75_80,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
369         MoveJ P5_Dejar_pto_2_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
370         WaitTime 0.5;
371         SetDO CierrePinza_CI, 1;
372         WaitDI presente, 1;
373         WaitTime 0.5;
374         MoveJ P5_Dejar_pto_2_prev_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
375         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
376         MoveJ P5_Prep_75_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
377         MoveL P5_Prep_75,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
378         WaitTime 0.5;
379         SetDO CierrePinza_CI, 0;
380         WaitDI presente, 0;
381         WaitTime 0.5;
382         MoveL P5_Prep_75_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
383         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
384     ENDIF
385
386     IF P5_p2_C_80=1 THEN
387         MoveJ P5_Dejar_pto_2_prev_75_80,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
388         MoveJ P5_Dejar_pto_2_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
389         WaitTime 0.5;
390         SetDO CierraPinza, 1;
391         WaitDI presente, 1;
392         WaitTime 0.5;
393         MoveJ P5_Dejar_pto_2_prev_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
394         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
395         MoveJ P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
396         MoveL P5_Prep_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
397         WaitTime 0.5;
398         SetDO CierraPinza, 0;
399         WaitDI presente, 0;
400         WaitTime 0.5;
401         MoveL P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
402         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
403     ENDIF
```

Figura 10.17 Parte IV P_5_Pirámide.

```
404         ! Pto 3
405         IF P5_p3_C60=1 THEN
406             MoveJ P5_Dejar_pto_3_prev,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
407             MoveJ P5_Dejar_pto_3,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
408             WaitTime 0.5;
409             SetDO CierraPinza_AnchoViga_60, 1;
410             WaitDI presente, 1;
411             WaitTime 0.5;
412             MoveJ P5_Dejar_pto_3_prev,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
413             MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
414             MoveJ P5_Prep_60_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
415             MoveL P5_Prep_60,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
416             WaitTime 0.5;
417             SetDO CierraPinza_AnchoViga_60, 0;
418             WaitDI presente, 0;
419             WaitTime 0.5;
420             MoveL P5_Prep_60_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
421             MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
422         ENDIF
423
424         IF P5_p3_C75=1 THEN
425             MoveJ P5_Dejar_pto_3_prev_75_80,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
426             MoveJ P5_Dejar_pto_3_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
427             WaitTime 0.5;
428             SetDO CierrePinza_CI, 1;
429             WaitDI presente, 1;
430             WaitTime 0.5;
431             MoveJ P5_Dejar_pto_3_prev_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
432             MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
433             MoveJ P5_Prep_75_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
434             MoveL P5_Prep_75,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
435             WaitTime 0.5;
436             SetDO CierrePinza_CI, 0;
437             WaitDI presente, 0;
438             WaitTime 0.5;
439             MoveL P5_Prep_75_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
440             MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
441         ENDIF
442
443         IF P5_p3_C80=1 THEN
444             MoveJ P5_Dejar_pto_3_prev_75_80,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
445             MoveJ P5_Dejar_pto_3_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
446             WaitTime 0.5;
447             SetDO CierraPinza, 1;
448             WaitDI presente, 1;
449             WaitTime 0.5;
450             MoveJ P5_Dejar_pto_3_prev_75_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
451             MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
452             MoveJ P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
453             MoveL P5_Prep_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
454             WaitTime 0.5;
455             SetDO CierraPinza, 0;
456             WaitDI presente, 0;
457             WaitTime 0.5;
458             MoveL P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
459             MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
460         ENDIF
```

Figura 10.18 Parte V P_5_Pirámide.

```
462         !Ahora esta parte del código se ocupa de apilar las cajas ordenadas en una pirámide
463         WaitTime 1;
464         MoveJ P5_Prep_80_2,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
465         MoveJ P5_Prep_80,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
466         WaitTime 1;
467         SetDO CierraPinza, 1;
468         WaitDI presente, 1;
469         WaitTime 1;
470         MoveJ P5_Prep_80_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
471         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
472         MoveJ P5_Colocar_1_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
473         MoveJ P5_Colocar_1,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
474         WaitTime 1;
475         SetDO CierraPinza, 0;
476         WaitDI presente, 0;
477         WaitTime 1;
478         MoveJ P5_Colocar_1_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
479         MoveJ P5_reposo,v500,fine,T_pinza\WObj:=Workobject_WORK_P5;
480         WaitTime 1;
481         !--------
482         MoveJ P5_Prep_75_2,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
483         MoveL P5_Prep_75,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
484         WaitTime 1;
485         SetDO CierrePinza_CI, 1;
486         WaitDI presente, 1;
487         WaitTime 1;
488         MoveJ P5_Dejar_pto_2_prev,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
489         MoveJ P5_reposo,v50,z200,T_pinza\WObj:=Workobject_WORK_P5;
490         MoveJ P5_Colocar_2_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
491         MoveL P5_Colocar_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
492         WaitTime 1;
493         SetDO CierrePinza_CI, 0;
494         WaitDI presente, 0;
495         WaitTime 1;
496         MoveL P5_Colocar_2,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
497         MoveJ P5_reposo,v500,z200,T_pinza\WObj:=Workobject_WORK_P5;
498         !---------
499         MoveJ P5_Prep_60_2,v100,fine,T_pinza\WObj:=Workobject_WORK_P5;
500         MoveL P5_Prep_60,v50,fine,T_pinza\WObj:=Workobject_WORK_P5;
501         WaitTime 1;
502         SetDO CierraPinza_AnchoViga_60, 1;
503         WaitDI presente, 1;
504         WaitTime 1;
```

Figura 10.19 Parte VI P_5_Pirámide.