Trabajo de Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Diseño de una estación virtual e implantación en sistema real de un robot IRB120

Autor: Álvaro Conejero Sánchez
Tutor: David Muñoz de la Peña Sequedo

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Diseño de una estación virtual e implantación en sistema real de un robot IRB120

Autor:
Álvaro Conejero Sánchez

Tutor:
David Muñoz de la Peña Sequedo

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

A mis abuelos a los que veo menos de lo que me gustaría

Agradecimientos

Un año que para mi comenzó siendo una huracán de acontecimientos ha pasado a llenarme de valores y de igual forma valorar más a los que me rodean y son parte de mis pilares fundamentales. Es por ello que en primer lugar sólo puedo acordarme de mis familiares y amigos más cercanos.

Doy gracias a David Muñoz, mi tutor, ya que al llegar a su despacho no tenía ni idea de que hacer mi trabajo, me propuso uno según mis preferencias hacia asignaturas que había dado en la escuela. Sinceramente, me ha venido "como anillo al dedo" y estoy muy contento de la decisión de haber aceptado su propuesta.

Debo por último incluir a Ignacio Alvarado, que ha sido el encagardo de las impresiones 3D y a veces de abrirme la puerta del laboratorio, pero siempre con un trato inigualable.

Álvaro Conejero Sánchez

Sevilla, 2018

Resumen

Este trabajo tuvo como objetivo la creación de una práctica para el alumnado en la que pudiera utilizar el *IRB120* ubicado en los laboratorios de la *ETSI* en asignaturas de introducción a la programación de robots manipulables. La compañía fabricante de estos robots, *ABB*, posee un software llamado *RobotStudio* que permite hacer simulaciones previas de nuestros programas en un entorno virtual. Esto último es ideal para que los alumnos puedan practicar previamente antes de realizar las prácticas con el robot real.

Como ejemplo de práctica a resolver por el alumno se propuso el juego tradicional de las Torres de Hanoi mediante recursividad. Para dicha resolución sería necesario el uso de algoritmos de cierto nivel de complejidad. Además se llevó a cabo la fabricación del juego mediante la impresora 3D, haciendo uso de las destrezas de diseño gráfico. A su vez, se debería tener en cuenta que el modelo de juego iba ser resuelto por un *IRB120* con una garra, por lo que sería necesario la creación de unos dedos adaptadores que pudieran coger las piezas para moverlas de una torre a otra.

El proyecto consistió en la creación de dicho entorno virtual y dejarlo todo preparado para que inmediatamente después de programar el robot virtual se pueda llevar esta programación al real sin problema. Dicha programación está basada en un lenguaje particular del *software* llamado *RAPID*.

Cabe decir que este trabajo tiene mucha relación con trabajos pasados también tutelados por David Muñoz. Estos trabajos pasados, ver *Bibliografía*, dejaron preparado el *IRB120* con la pinza neumática *MHQ2-D20* como terminación. Dicha pinza tiene unos adaptadores donde podemos colocar nuevas terminaciones que nos permitan realizar una acción específica. En nuestro caso se colocaron unas terminaciones que nos permiten mover las fichas del juego.

Un reto añadido al proyecto surgió al comprobar que *RobotStudio* no tenía la potencia suficiente para resolver el juego por recursividad. Es por ello que fue necesaria la creación de una comunicación por socket entre *Matlab* y *RobotStudio*.

Abstract

This dissertation had as a goal the creation of a practice for students in which they could use the *IRB120* located in ETSI's laboratories in subjects dealing with the introduction to programming manipulable robots. The company which manufactures these robots, *ABB*, owns a software called *RobotStudio*, which allows us to make previous simulations of our programs in a virtual setting. This is ideal for students to practise previously before doing the practice with the real robot.

As an example of practice to be solved by students, the traditional game "*The Towers of Hanoi*" through recursion was proposed. In order to solve it, the use of certain complex algorithms would be needed. Moreover, the making of the game was carried out through a 3D printer, using the graphic design skills. At the same time, it should be considered that the game model was going to be solved by an *IRB120* with a gripper, so the creation of adaptor fingers in order to take the pieces and move it from one tower to another would be needed.

The project dealt with the creation of the previously mentioned virtual setting and leave everything ready so that immediately after programming the virtual robot, students could be able to do it in a real environment without any problem. This programming is based on a particular software language called *RAPID.*

It should be noted that this project has a lot in common with previous projects also supervised by David Muñoz. These previous projects, see *Bibliography,* left the *IRB120* ready with the pneumatic clamp as the termination. This clamp has some adaptors where we can add new terminations which allow us to perform a specific action. In our case, we placed terminations which allowed us to move the pieces of the game.

A challenge added to the project emerged when we checked that *RobotStudio* did not have enough power to solve the game through recursion. Because of that, the creation of a communication by socket between *Matlab* and *RobotStudio.*

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN Y OBJETIVOS

En la actualidad, existen un gran número de industrias que utilizan robots en su cadena de producción. Es por ello que el Departamento de Ingeniería de Sistemas y Automática adquirió un par de *IRB120* cuyo objetivo principal es que el alumno pueda tener sus primeros contactos con un robot de carácter industrial.

El fabricante de estos robots, *ABB*, posee a su vez el software *RobotStudio*. En el que se pueden crear estaciones virtuales sobre las que probar diseños de programación. Esta herramienta es un gran valor añadido para el aprendizaje.

Inicialmente, los robots no poseen un elemento actuador, es por ello que en proyectos pasados se instaló en uno de los dos robots una garra neumática *MHZ2-20D* y se dejó lista para el funcionamiento. La garra abre y cierra un par de terminaciones, pero, lo más relevante de estas es que disponen de un sistema de adaptadores que permiten la colocación de una nueva terminación y así poder ser utilizada con diferentes fines. Esto último es de gran interés ya que el departamento posee una impresora *3D, XYZprinting da Vinci 2.0 Duo*, con la cual podemos construir nuevas terminaciones a bajo coste.

Teniendo este escenario en mente, el siguiente paso natural era el desarrollo de una práctica para el alumnado, este fue el objetivo principal de este proyecto.

Para su realización es necesaria la creación de creación de un entorno virtual basado en el *IRB120* con la pinza neumática, sobre el cual el alumno pueda hacer pruebas con sus diseños de programación, y, una vez que se haya comprobado que el código del alumno es correcto, su posterior implantación en el robot real.

Como enunciado de la práctica se pensó en que el alumno tuviera que resolver el juego de las Torres de Hanoi, concretamente por el método de recursividad.

Para hacer todo esto fue necesario hacer el modelo 3D de las diferentes piezas que forman el juego y los ejecutores finales que se adaptan a la pinza eléctrica para coger dichas piezas. Además, *RobotStudio* tiene en sus bibliotecas los *IRB120*., pero su terminación hay que diseñarla y hacer que sea "inteligente" mediante el diseño de un *SmartComponent* y de la lógica de la estación.

Para la definición de la herramienta y bases de coordenadas, es necesario el uso de la *FlexPendant*. Este utensilio está directamente conectado a la controladora del robot y nos permite mover las diferentes partes del robot, definir herramientas y bases de coordenadas entre otros. La *FlexPendant* también puede ser emulada desde *RobotStudio*, por lo que inicialmente se hicieron pruebas con ella.

Un valor añadido al proyecto fue la necesidad de comunicación entre *Matlab* y *RobotStudio*. Esto fue necesario porque *RobotStudio* no tiene la potencia suficiente para realizar los cálculos de recursividad necesarios para resolver el juego.

Por último, se resolvió mediante *RAPID* los movimientos ordenados por la matriz que llega desde *Matlab* y se realizaron los ajustes pertinentes para que el programa de la simulación y el real funcionen igual.

1.1 Robot *ARB120*

Robot industrial multiusos de 25 kg de peso. Este robot puede soportar cargas de hasta 3 kg (4 kg en posición vertical de la muñeca) con un alcance de 580 mm. Se trata del robot más pequeño de *ABB*, de gran capacidad de producción frente a una baja inversión.

Este robot posee una estructura abierta especialmente adaptada para un uso flexible y presenta la posibilidad de comunicación con sistemas externos.

En concreto el *IRB120 Clean room* (de acabado en blanco), está únicamente diseñado para su empleo a nivel académico.

Figura 1: *ABB IRB120 Clean room*

1.2 Pinza neumática *MHQ2-20D*

Pinza agregada a la muñeca del *IRB120*. Existe una comunicación entre nuestro robot y la pinza neumática gracias al TFG nombrado anteriormente. Además la pinza posee unos dedos los cuales pueden funcionar como adaptadores para darle distintas funcionalidades a la pinza. En nuestro caso se crearon unas garras capaces de coger piezas de 50 mm de grosor. Dicho proceso de creación se explicará en el apartado 3.1 de este documento.

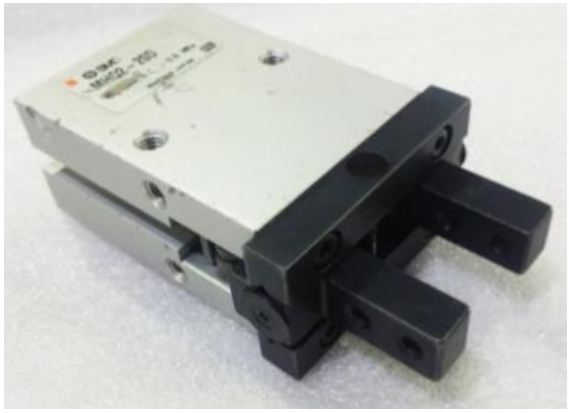


Figura 2: Pinza neumática *MHQ2-20D*

1.3 Instalación neumática

El laboratorio dispone de una instalación neumática que es utilizada para la apertura y cierre de la garra. Para disponer de aire comprimido en la instalación basta con girar la válvula roja.

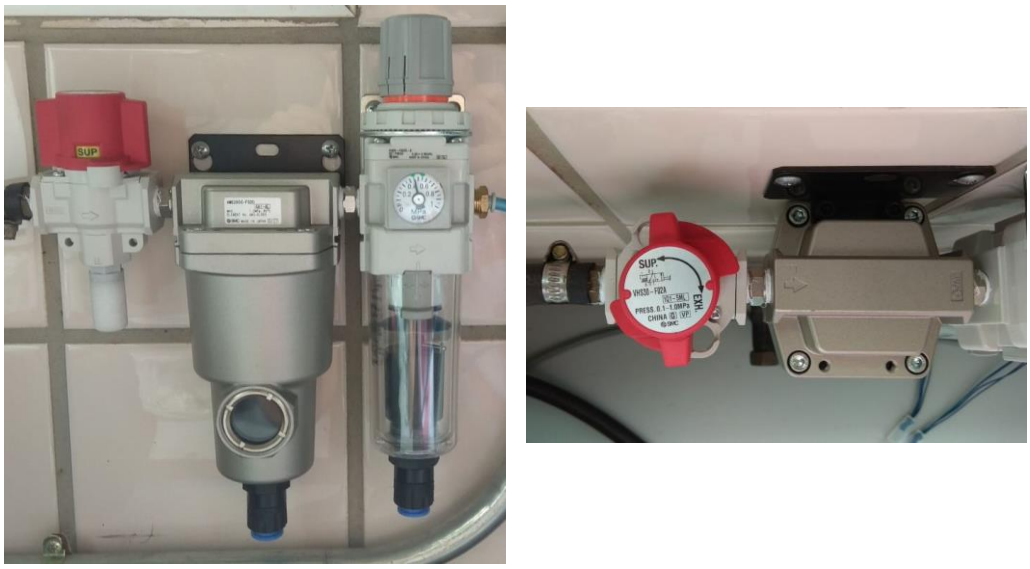


Figura 3: Instalación neumática laboratorio

1.4 Reglas de las Torres de Hanoi

El rompecabezas consiste en pasar una torre de un punto A a otro punto B pasando por un punto C auxiliar. Inicialmente se comienza con la torre apilada de mayor a menor tamaño (la de mayor tamaño en la parte baja de la torre) en el punto A.

Para mover las fichas hay que seguir las siguientes normas:

- Sólo se mueve una ficha a la vez.
- Una ficha de mayor tamaño no puede estar sobre otra de menor tamaño.
- Sólo podremos mover la ficha que esté encima de nuestras to

Figura 4: Juego Torres de Hanoi

2 DISEÑO DEL JUEGO Y DE LOS DEDOS DE LAS GARRA

EL diseño de las fichas del tablero y las garras que se agregaron al adaptador fueron todas diseñadas mediante la aplicación *Inventor* de *Autodesk*, *software* de *CAD 3D* para el desarrollo de productos, concretamenete su versión de 2018. Esta herramienta es de un gran uso en el diseño ingenieril junto a otros programas como *CATIA* o *Solid Edge*. El *software* se puede descargar accediendo mediante *UVUS* a la zona de descargas de la Universidad de Sevilla y descargándose el paquete de *Autodesk*.

2.1 Modelos preliminares

Se pensaron diversas posibilidades a la hora del diseño del juego. Por costumbre, el juego normalmente se ha diseñado con discos de mayor a menor radio, pero eso supondría variar los tamaños de agarre de la pinza. Al ser la pinza un mecanismo de todo o nada se decidió diseñar diferentes modelos de juego para las garras anteriormente diseñadas en el TFG *"Diseño y fabricación de garras para un IRB120"*, de *MªLuisa Fernández*.

Inicialmente se pensó en hacer la base del juego también con la impresora 3D, pero la idea se terminó desechando por la incomodidad que suponía imprimir la base debido a que el volumen máximo e impresión es de 15 x 20 x 20 cm. Por lo que finalmente se hizo mediante un trabajo en madera.

2.1.1 Proceso de creación de piezas en *AutoDesk Inventor*

En este apartado se hará un resumen de las herramientas utilizadas en *Inventor* durante la creación del proyecto. Para ello se irán comentando como se han realizado las diferentes piezas de este proyecto.

Todo proyecto creado en Inventor parte inicialmente de un boceto sobre uno de los tres planos en el espacio. Para ello utilizamos la herramienta Iniciar boceto 2D localizada en los distintos paneles de la cinta de opciones.

Figura 5: Iniciar boceto 2D *Inventor*

Para la realización del boceto podemos se hace uso de las opciones línea, círculo, arco o rectángulo. Como ejemplo vamos a hacer el diseño de un juego basado en cilindros de diferentes alturas de 50 mm de diámetro (para su utilización con la garra que puede coger piezas de 50 mm) por lo que se hará uso de la opción círculo.

Figura 6: Círculo *Inventor*

Una vez clicado podremos dimensionar un círculo de diámetro 50 mm.

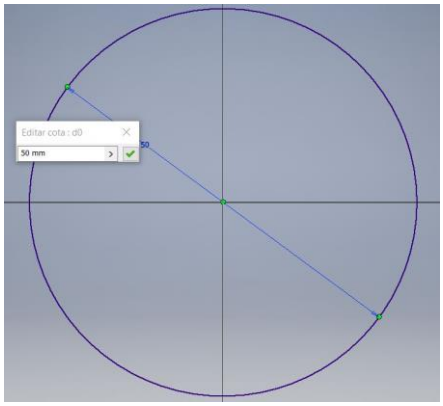


Figura 7: Dimensionado círculo 50 mm *Inventor*

Para terminar el boceto seleccionamos terminar boceto.

Figura 8: Terminar boceto *Inventor*

Para dar una altura a nuestro cilindro es necesario estruir la pieza. De nuevo en la misma barra de opciones podemos encontrar el botón extruir.

Figura 9: Extruir *Inventor*

Una vez estemos en la opción extruir, y hayamos clicado sobre nuestro boceto, nos aparecerá una ventana que nos permite seleccionar la distancia de extrusión. En este ejemplo la altura del cilindro tendrá 20 mm.

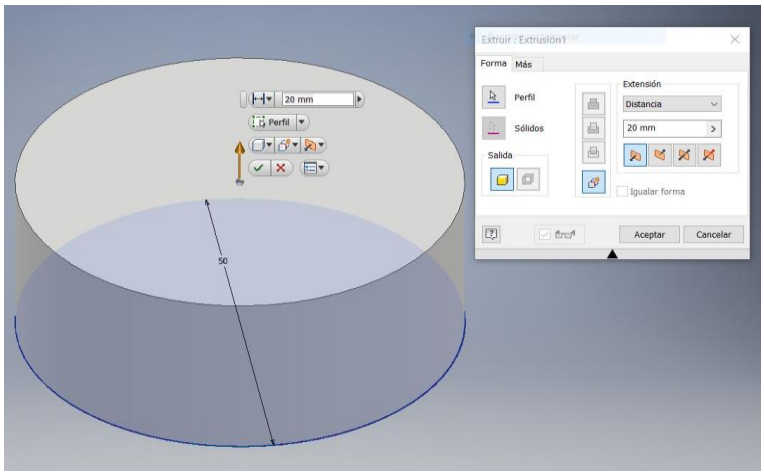


Figura 10: Selección de distancia de extrusión *Inventor*

Por último para hacerle un agujero pasante utilizamos el botón Agujero. Haciendo click sobre la superficie donde deseemos crear el agujero podremos crear un agujero de cierto diámetro según un centro que hayamos seleccionado.

Figura 11: Agujero *Inventor*

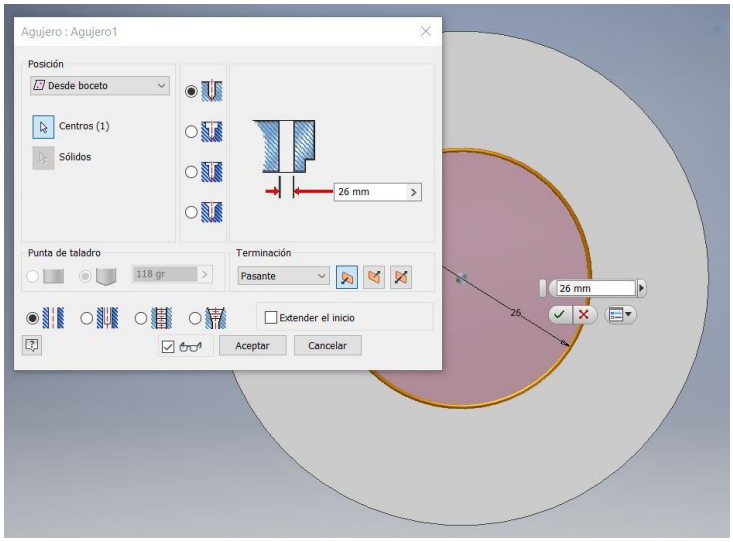


Figura 12: Creación agujero 26 mm *Inventor*

Por último, para poder tener una visión mejor del conjunto global, se modifica el aspecto del cilindro agujereado. Primero hemos de seleccionar el botón aspecto.

Figura 13: Aspecto *Inventor*

Se abre una ventana desde la cual podemos agregar distintos aspectos al documento y aspectos de la biblioteca de *Inventor* los cuales podemos usar. En este caso agregamos el aspecto naranja claro.

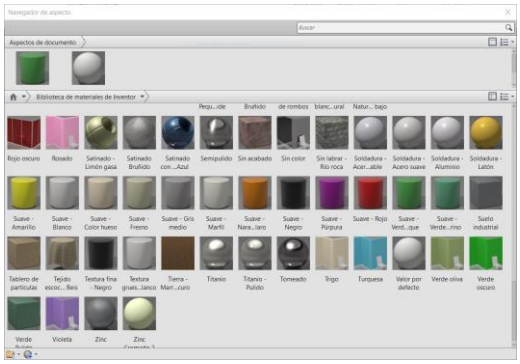


Figura 14: Agregar aspecto *Inventor*

Una vez agregado el aspecto al documento, ya podemos repetir el aspecto sobre nuestro cilindro.

Figura 15: Repetir aspecto *Inventor*

Quedando finalmente la ficha de nuestro juego basado en cilindros.

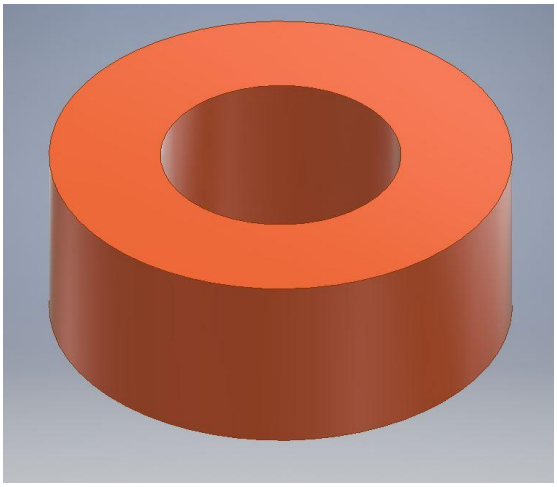


Figura 16: Diseño cilindro finalizado *Inventor*

De forma muy similar se crea la base del juego utilizando las herramientas boceto 2D, rectángulo, círculo y extrusión podremos diseñar la base desea. Cabe destacar que podemos extruir las tres columnas de la base del juego de forma simultánea.

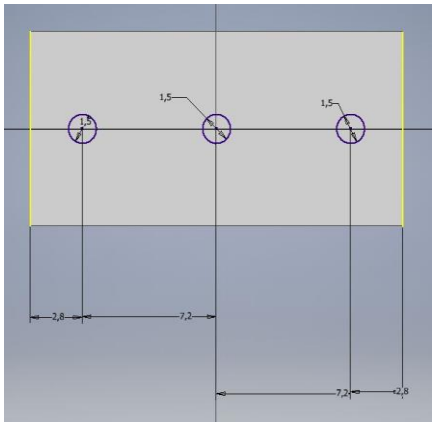


Figura 18: Boceto 2D tres columnas para base *Inventor*

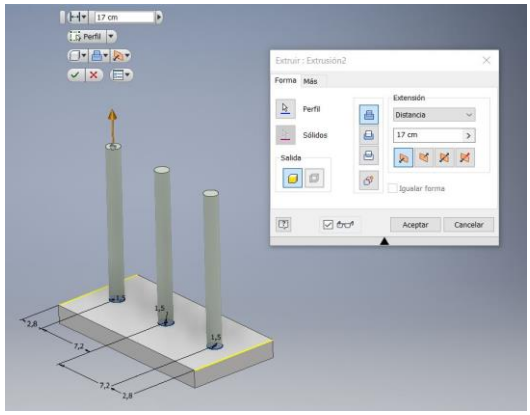


Figura 18: Extrusión tres columnas base *Inventor*

Un proceso parecido se utilizó para la creación de un modelo de juego con ortoedros.

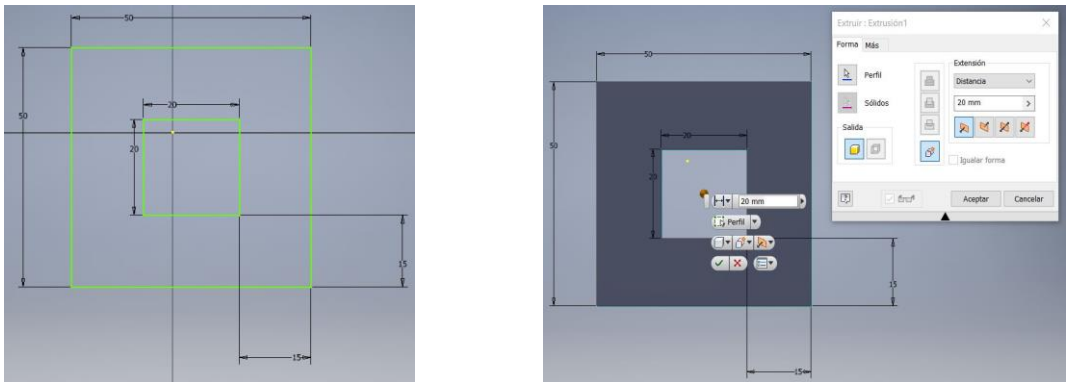


Figura 19: Creación ortoedro *Inventor*

Una herramienta también utilizada en el diseño de las piezas es la de empalme. Con ella podemos crear empalmes en las aristas de los ortoedros y bases para darles un acabado mas fino.

Figura 20: Emplame *Inventor*

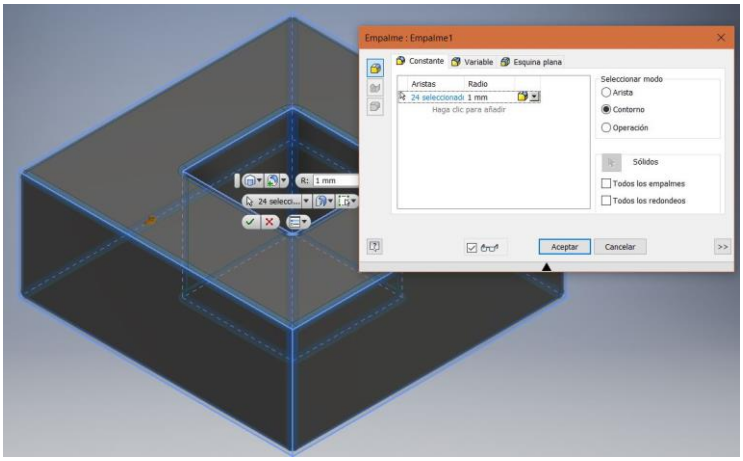


Figura 21: Creación empalmes 1 mm *Inventor*

Es necesario seleccionar las aristas donde queramos los empalmes y seleccionar en la ventana auxiliar el radio deseado.

Una vez se terminaron los ortoedros y la base se creó un conjunto donde tenemos una previsuacización de los ortoedros y la base antes de llevarlos a *RobotStudio*.

Otra de las alternativas propuestas es un juego basado en “gorritos invertidos”, esta variante se creó de forma muy similar a las anterioes piezas por lo que no se hace incapié de nuevo en su formación.

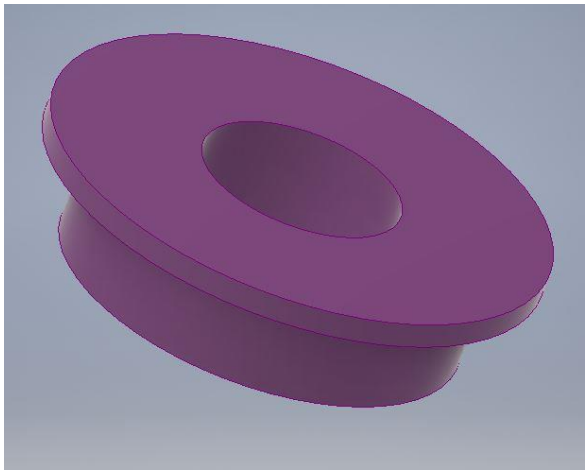
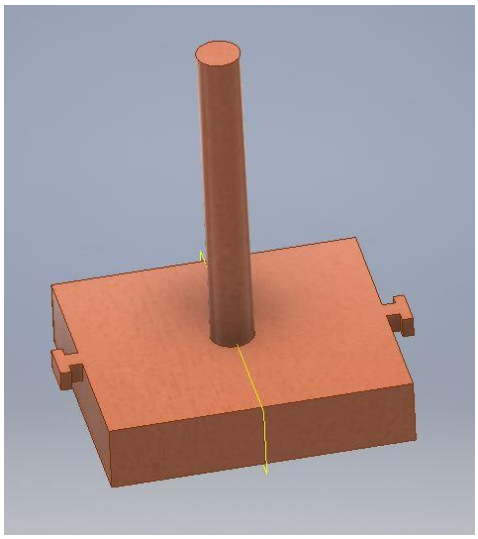
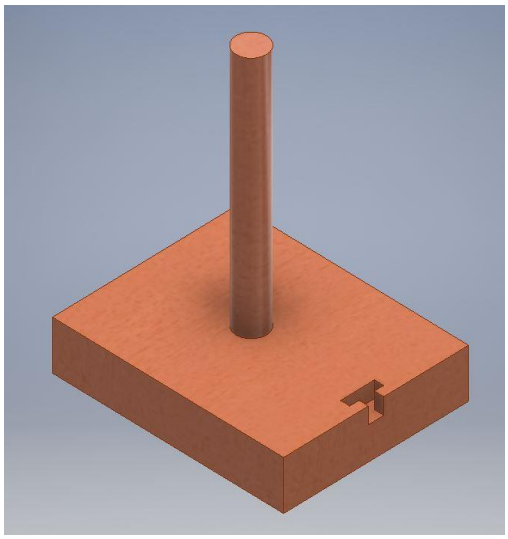


Figura 22: “Gorrito invertido” *Inventor*

Inicialmente, cuando aun se pensaba que la base sería realizada por impresión 3D, se pensó un modelo de base diferente tipo puzzle que permite impresión de una base de tres componentes que encajarían tras su fabricación.





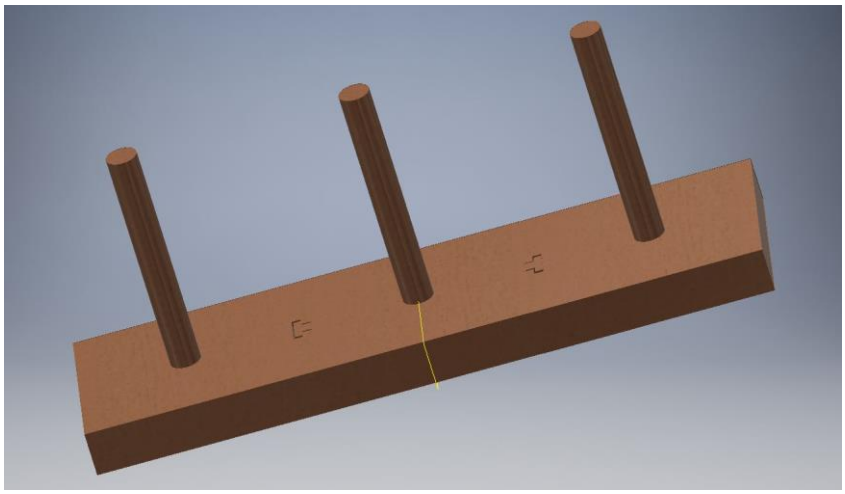


Figura 23: Base tipo puzzle *Inventor*

Para la creación de la base central, se hizo uso de simetría, esta herramienta permitió la construcción del segundo macho que hace que encajen las bases con mayor rapidez.

Figura 24: Simetría *Inventor*

Antes del proceso de creación de una simetría es necesario la creación de un plano que nos servirá como plano de simetría. Una vez creado el plano y le demos al botón de simetría nos aparece la ventana auxiliar donde deberemos seleccionar el plano respecto al cual hacer la simetría y la operación que precisa una simetría, en nuestro caso las relacionadas con la construcción del elemento conector de la base central.

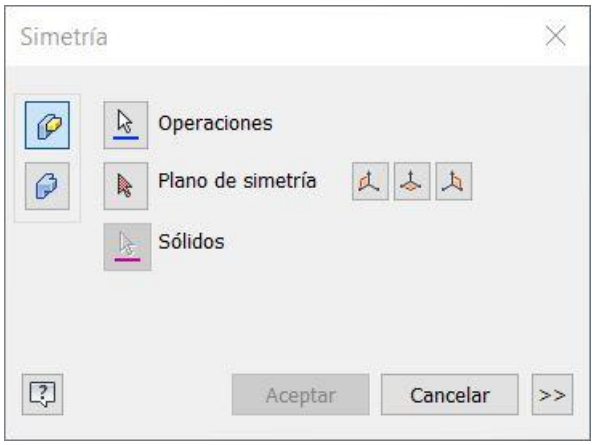


Figura 25: Operación simetría *Inventor*

Por último para ver los modelos completos se ensamblan creando un nuevo archivo accediendo a *archivo>nuevo>ensamblaje.*

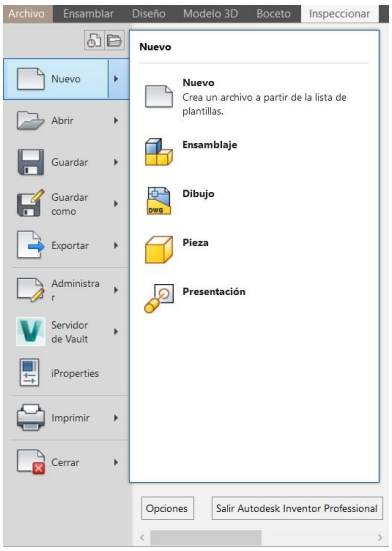


Figura 26: Nuevo ensamble *Inventor*

Se insertan las diferentes piezas que vayamos a usar. Por ejemplo vamos a insertar la base del juego de fichas cilíndricas y la ficha de 20 mm de altura que hemos creado anteriormente. Para ello pinchamos en insertar y abrimos los ficheros correspondientes.

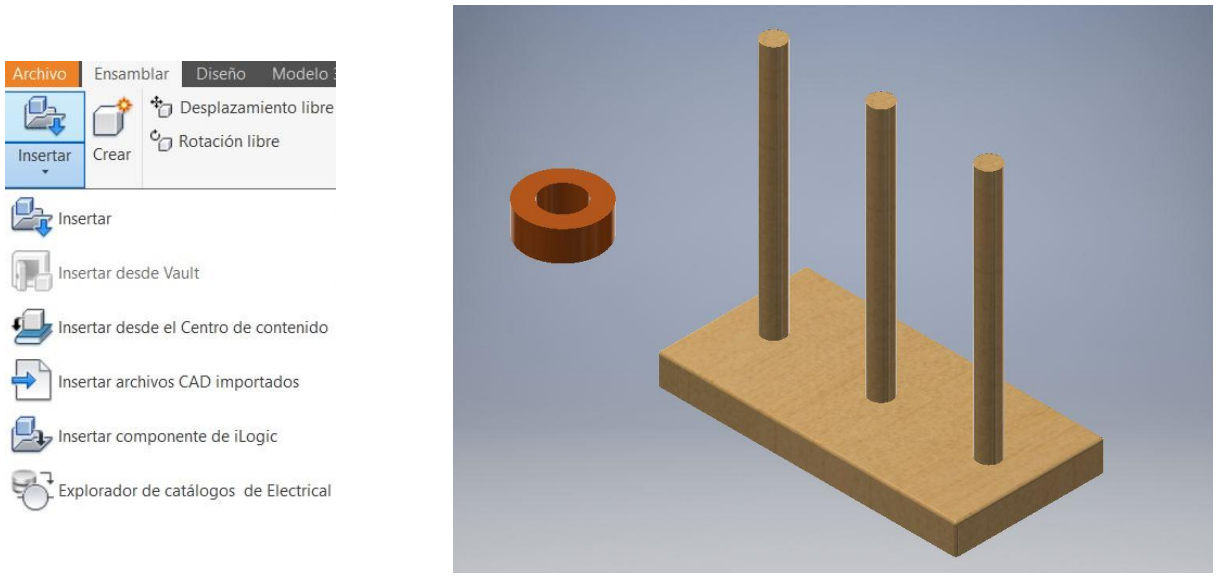


Figura 27: Insertar piezas *Inventor*

A continuación vamos a insertar la ficha cilíndrica en uno de los ejes del juego. Seleccionamos el icono unión desde ensamblar que nos da acceso a una ventana emergente donde podremos elegir el tipo de unión, en nuestro caso seleccionamos el tipo cilíndrica. Tras esto es necesario indicar el origen en el primer componente y en el segundo.

Figura 28: Unión *Inventor*

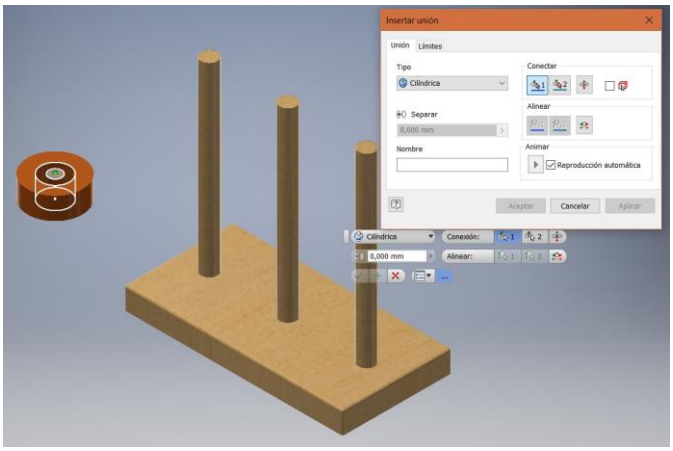


Figura 29: Crear unión primer componente *Inventor*

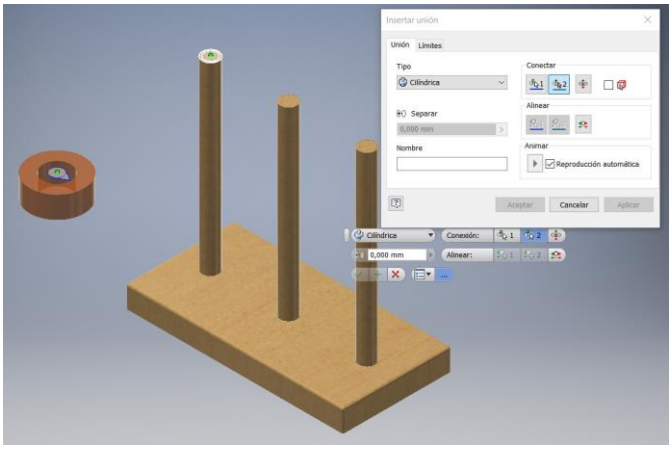


Figura 30: Crear unión segundo componente *Inventor*

Hecho esto el cilindro no podrá abandonar el eje. Pero notamos que si puede atravesar base. Para resolver este problema es necesario indicar que la pieza será tipo conjunto de contacto haciendo click derecho sobre ella y seleccionando conjunto de contactos.

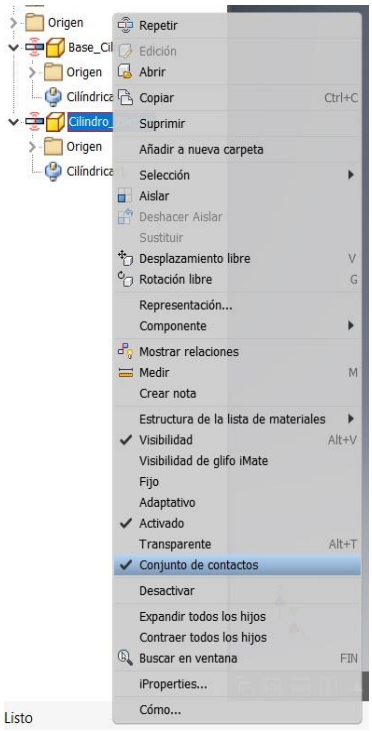


Figura 31: Selección de conjunto de contactos *Inventor*

Por último para ver que efectivamente todo marcha bien se debe acceder a inspeccionar y activar solucionador de contactos. Esto hace que el cilindro no pueda atravesar la base.

Figura 32: Activar conjunto de contactos *Inventor*

2.1.2 Elección del modelo

Tras el punto anterior tenemos tres diseños que parecen bastante útiles para el trabajo que nos ocupa. El modelo de ortoedros y cilindros están orientados a la utilización de la garra paralela de 50 mm. Pese a ello, el diseño de herramienta que se tenía pese a ser para piezas de 50 mm de grosor, no iba a permitir coger los cilindros de 50 mm ya que la pinza no tiene una longitud de dedo suficiente. Por lo que se decidió no modificar el diseño de los dedos de la garra (aunque si sufrió alguna modificación, se verá más adelante) y utilizar el juego de ortoedros.

Figura 34: Juego final cilindros *Inventor*

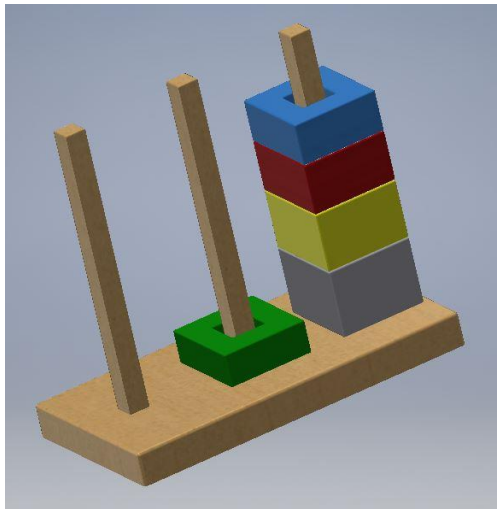


Figura 34: Juego final ortoedros *Inventor*

Por otro lado, el modelo de los gorritos invertidos, fué pensado para la pinza *Cleveland Guest Engineering LTD* la cual posee el segundo robot del laboratorio. Al tener estos un ala que permite la cogida de las piezas de forma satisfactoria. Por lo que se deja el modelo para proyectos futuros de índole semejante.

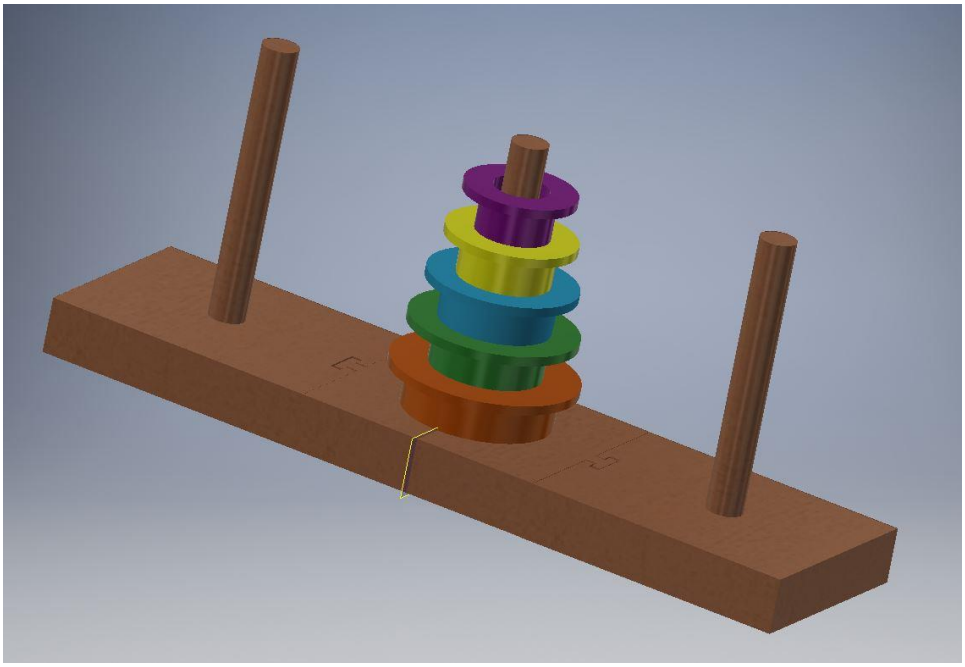


Figura 35: Juego final "gorritos" *Inventor*

2.2 Modelo final tras la realización de la base de madera

Obtenidos los modelos del juego se procedió a la creación de la base de madera en un taller de carpintería, pese a que su fabricación intentó ser lo más fiel posible a los planos facilitados. Tras tener la base construida pudimos ver que las columnas eran algo estrechas por lo que se decidió modificar el diseño y que fuera de 1,5 x 1,75 el cuadrado base. Es por ello que al finalizar la construcción de la base real se procedió a modelar una nueva base virtual con las medidas reales. Esto último se realizó tomando medidas de la base final y haciendo su modelo de forma similar a los creados en el *apartado 2.1 Modelos preliminares*.

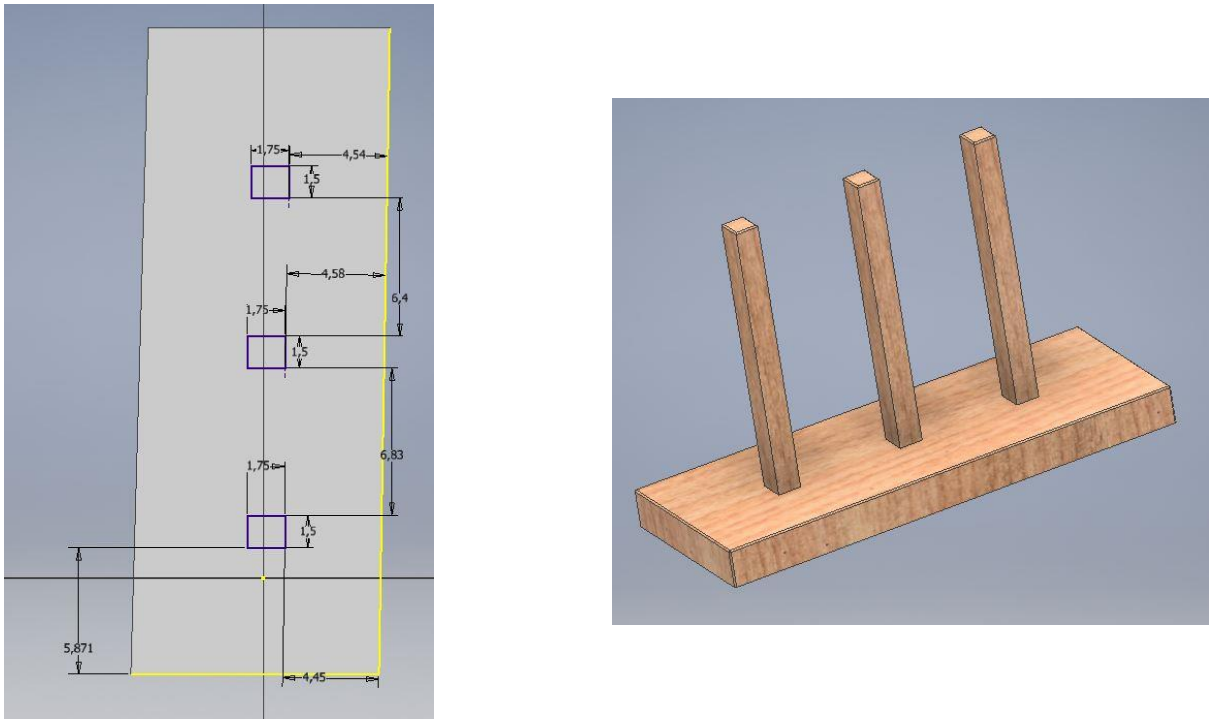


Figura 36: Creación de base real *Inventor*

2.3 Diseño de los dedos de la garra

El diseño del cuerpo de la pinza de *SMC* es del trabajo anterior *"Diseño y fabricación de Garras para un IRB120"* .Se hicieron un diseño de dedos para agarrar piezas de 50 mm. Además, para la visualización correcta en *RobotStudio* se decidió cambiar los conectores que tenían originalmente la pinza virtual ya que parecía que colisionaban cuando se cierra la pinza.

2.3.1 Diseño dedos

Las terminaciones de la pinza de *SMC* abren 5 mm y además la distancia desde el centro de la pinza hasta el frente de cada dedo es de unos 8 mm al estar cerrados y queriendo conseguir un apriete de unos 3 mm en total, se obtiene que la distancia desde el centro hasta el frente de la garra debe ser de 11 mm con la pinza cerrada. Por consiguiente, la pinza entra a coger la pieza con una holgura de 3.5 mm por garra.

Para su construcción se realizó una extrusión parecida a las de apartados anteriores. Se dibujó el boceto de una especie de zeta y se dimensionó. Tras ello se realizó la operación de extrusión.

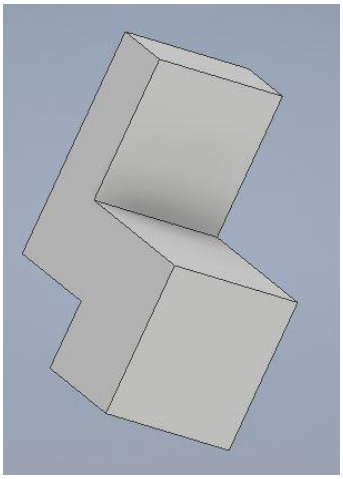


Figura 37: Extrusión para dedo pinza *Inventor*

Además, el conector de la pinza posee unos taladros roscados, por lo que se hace coincidir estos con unos agujeros en nuestros dedos. Para esto, en Inventor, hay que colocar el punto donde queramos ponerlo (en la foto hay dos puntos ya que hacemos dos agujeros).

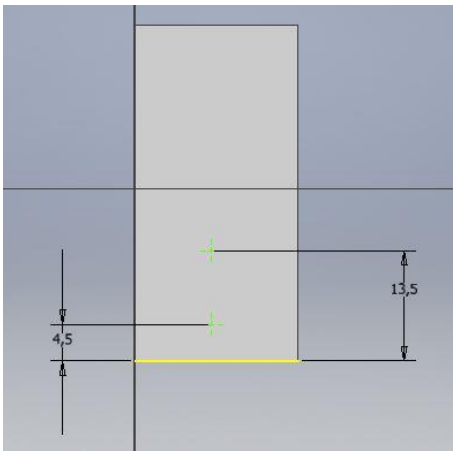


Figura 38: Creación de puntos para agujeros
Inventor

Tras esto, pulsando el botón agujero podemos seleccionar ambos puntos y hacer nuestros agujeros pasantes.

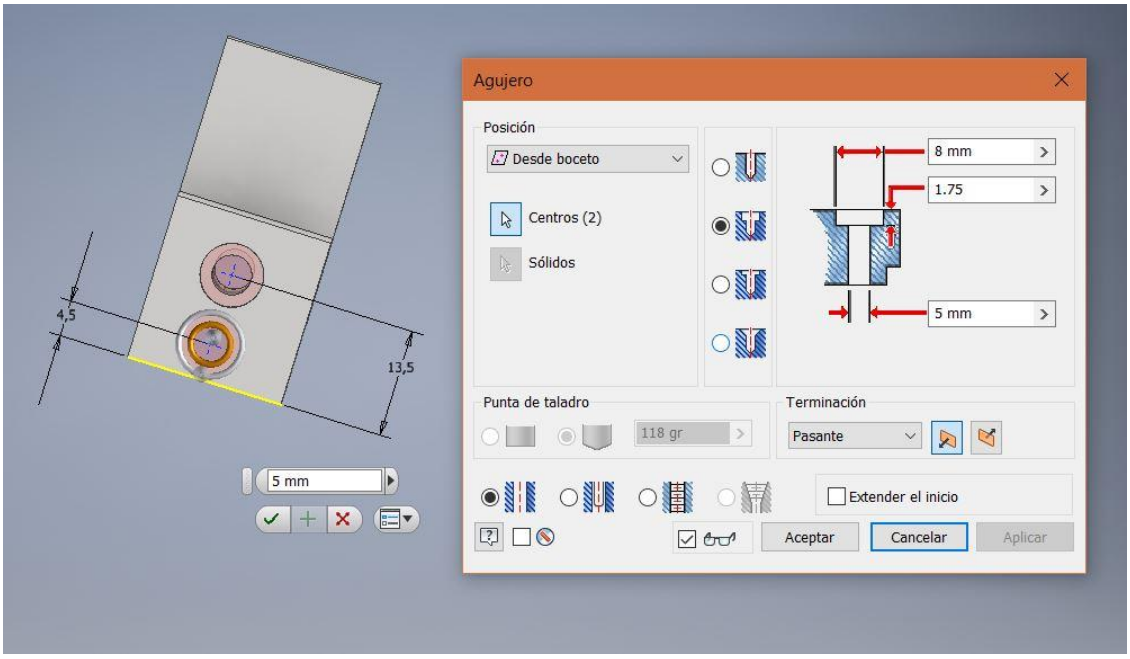


Figura 39: Creación de agujeros pasantes *Inventor*

Además, de nuevo hubo que realizar una extrusión de forma rectangular para hacer el hueco por donde vamos a ponertar las terminaciones de la pinza de *SMC* con nuestros dedos.

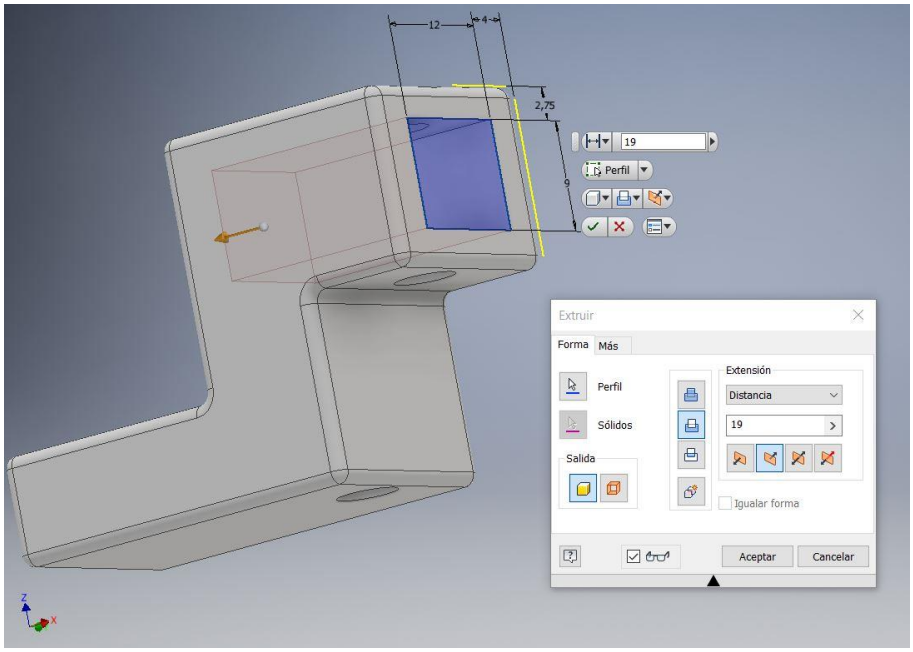


Figura 40: Creación de extrusión para agujero rectangular *Inventor*

Se realizaron también empalmes, de 1 mm. En esta ocasión todas las aristas de la pieza llevaban, por lo que era más fácil hacerlos por contorno en vez de aristas marcando la opción contorno.

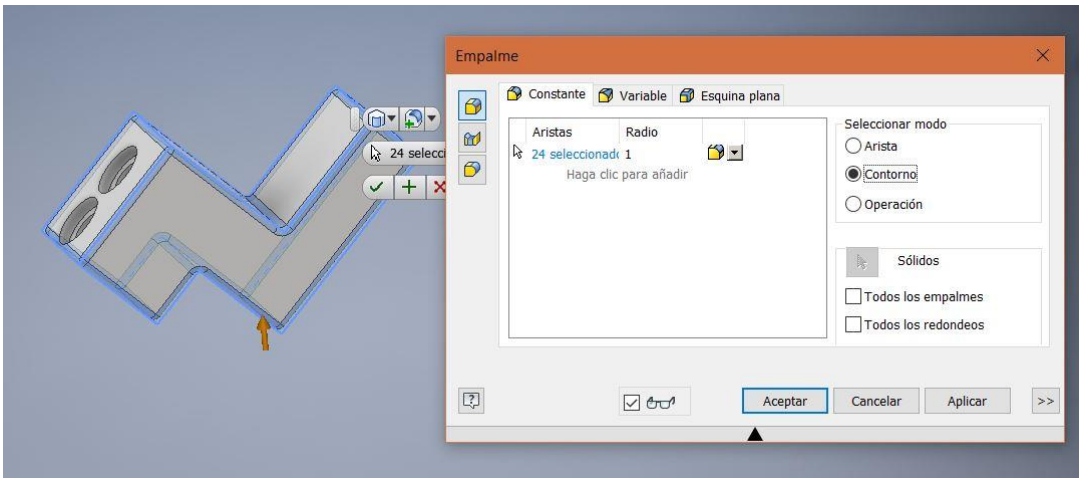


Figura 41: Creación de empalmes por contorno *Inventor*

Ambos dedos son iguales pero localizados de forma simétrica en las terminaciones de la pinza, por lo que se imprimeron dos.

2.3.2 Diseño conectores para herramienta

Los conectores son útiles en la visualización de la herramienta de robot estudio, en el diseño hemos realizado operaciones ya conocidas. Por lo que no se hace incapié de nuevo en ellas. Las medidas son las tomadas de los adaptadores del TFG de *Mª Luisa Fernandez* pero en esta ocasión la base del adaptador tiene la misma medida que la de nuestros dedos para que no parezca que colisionan los conectores en la visualización.

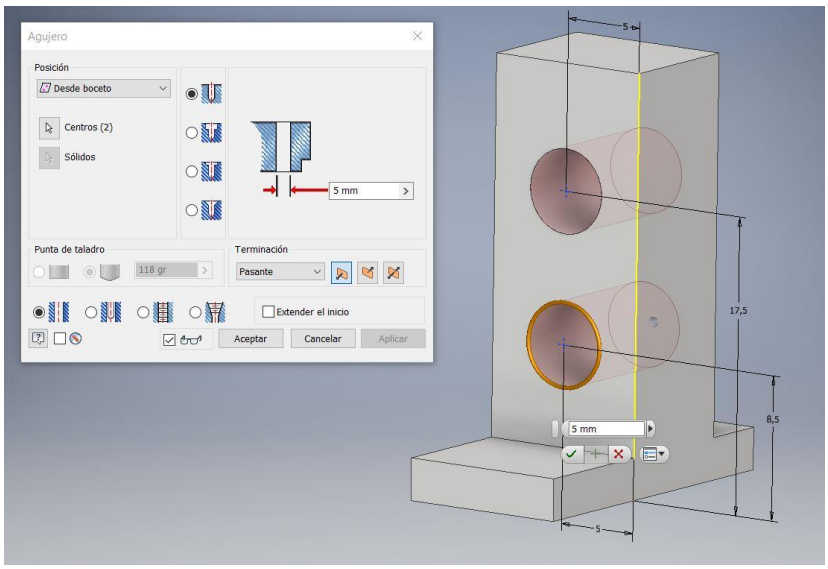


Figura 42: Creación agujeros adaptador *Inventor*

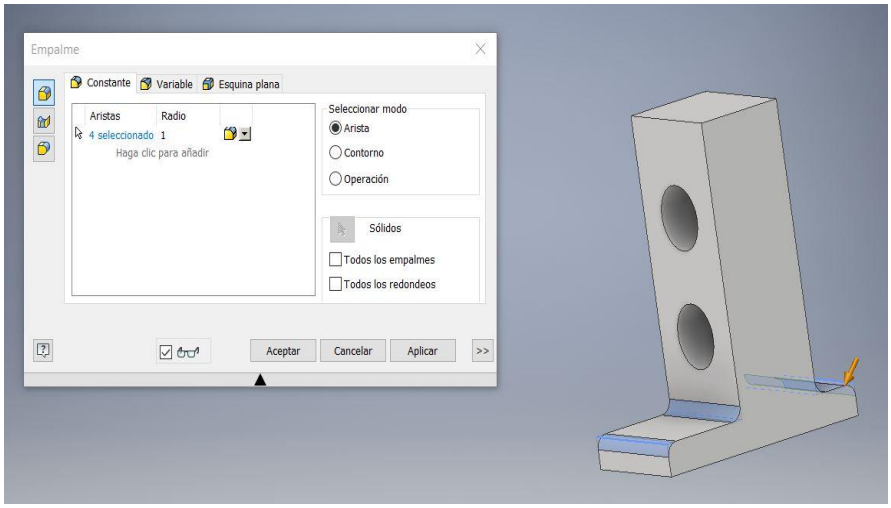


Figura 43: Creación empalmes adaptador *Inventor*

3 ESTACIÓN DE ROBOTSTUDIO

En este capítulo vamos a describir el proceso de creación de la estación virtual de *RobotStudio*, software de *ABB* el cual se puede obtener de su página web principal, aunque la versión descargable está bastante recortada en funciones, para acceder por completo hay que tener una licencia, como la que posee la *US*. No basta con la colocación espacial de las piezas creadas con Inventor. Además hay que dotar a la estación de cierta inteligencia para poder coger los objetos con la pinza. Este apartado es muy similar al del proyecto del año pasado, pero hay ciertos detalles en la creación de la herramienta que son necesarios para que pueda ser trasladado desde un programa, creado y probado con la estación, al robot real.

Concretamente la posición del efector fue corregida, ya que sus ejes no estaban bien colocados, y se creó un PlaneSensor asociado a uno de los dedos de la garra para una mayor precisión en la visualización de los movimientos. De igual forma, al tener que sustituir los dedos de la garra era necesario volver a crear la herramienta por completo por lo que también se incluye en este apartado su edición.

3.1 Creación de la herramienta

Para este apartado es necesario la importación de las geometrías de la herramienta y unirlas adecuadamente. Una vez hecho esto, se necesita saber exactamente cual es su posición y orientacíón al unirlo a *IRB120* ya que de ello depente que lo programado funcione correctamente. Para ello es necesario el cálculo del ángulo que forma la herramienta con el *IRB120* en su terminación.

3.1.1 Montaje de la herramienta en *RobotStudio*

Lo primero que debemos hacer es importar los diferentes components de nuestra herramienta por medio de importar geometría. Abierto buscaremos nuestras geometrías y las importaremos, para ello deben estar en format *.sat*.

Figura 44: Importar geometría
RobotStudio

En el caso de que queramos modificar el color de las piezas basta con *click derecho pieza>modificar>seleccionar color>seleccionar color>aceptar.*

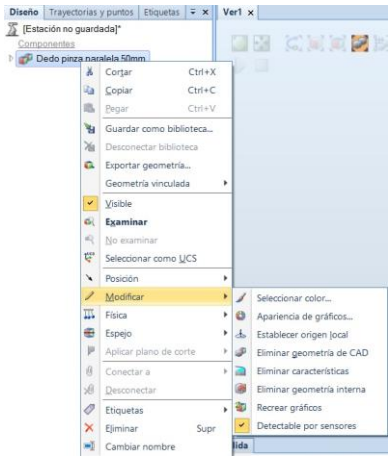


Figura 45: Selección color *RobotStudio*

De igual forma en el caso en el que queramos ver o no uno de los objetos representados en *RobotStudio* basta con click derecho *objeto>visible*, esto lo hace visible o visible según esté marcado o no.

Para posicionar el origen local de una pieza seleccionamos establecer *click derecho pieza>modificar>origen local*. Por ejemplo lo haremos en el adaptador y uno de los dedos que parece el caso más complicado de reubicación y selección de orígenes locales.

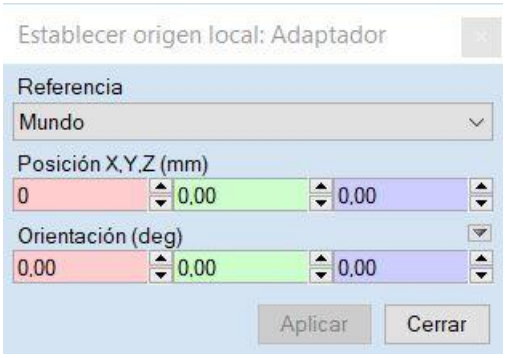


Figura 46: Establecer origen local *RobotStudio*

Para marcar la posición en la que queremos el origen local podemos hacerlo direcatmente introduciendo las coordenadas del punto que queremos. Además *RobotStudio* dispone de opciones de selección de puntos según la geometría. Para su uso debemos primero marcar el tipo de ajuste del punto después volver a la ventana de establecer origen local y seleccionar un apartado cualquiera de posición. Una vez hecho esto podemos indicarle con el ratón donde se encuentra nuestro punto. Por ejemplo, para seleccionar el origen local del adaptador, primero hemos hecho selección en establecer origen local del adaptador, tras esto ajustar a centro, volvemos a la ventana de origen local y marcamos una de las casillas de posición y ahora con el ratón seleccionamos el punto de la geometría que queremos marcar como origen local.

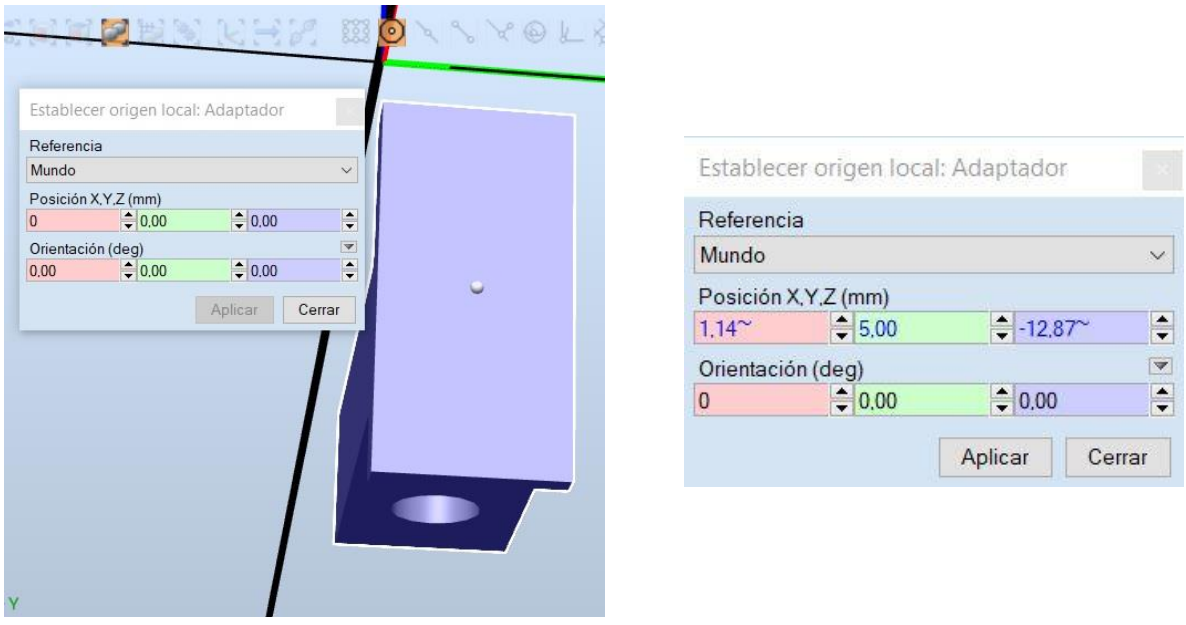


Figura 47: Creación origen local adaptador *RobotStudio*

De esta forma ya tenemos el origen local localizado, además podemos situar sus ejes de coordenadas ya que no son los que queremos, para ello debemos cambiar los grados la orientación según nos convenga. Para comprobar que hemos localizado correctamente el origen local basta con posicionarlo en el origen mundo hacienda uso de la opción fijar posición respecto a referencia mundo.

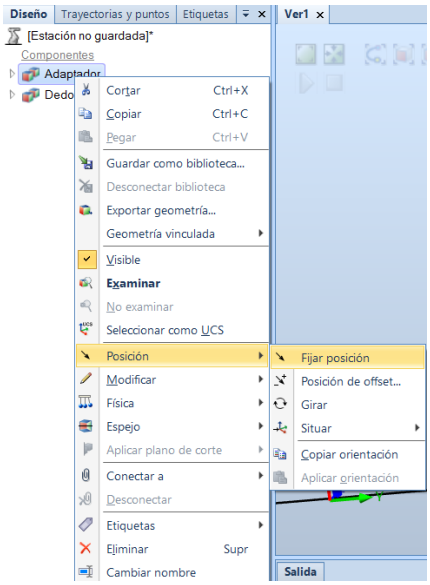


Figura 48: Seleccionar fijar posición *RobotStudio*

De la misma forma podemos localizar el orgen local en el dedo, en este caso lo mejor es ponerlo donde se va a posicionar con nuestro adaptador, un buen lugar es la cara del interior del agujero rectangular. Tras esto y con la opcion de definir posición y hacienda uso de ajustar a centro de nuevo, podemos situar nuestro dedo en el adaptador de la forma adecuada.

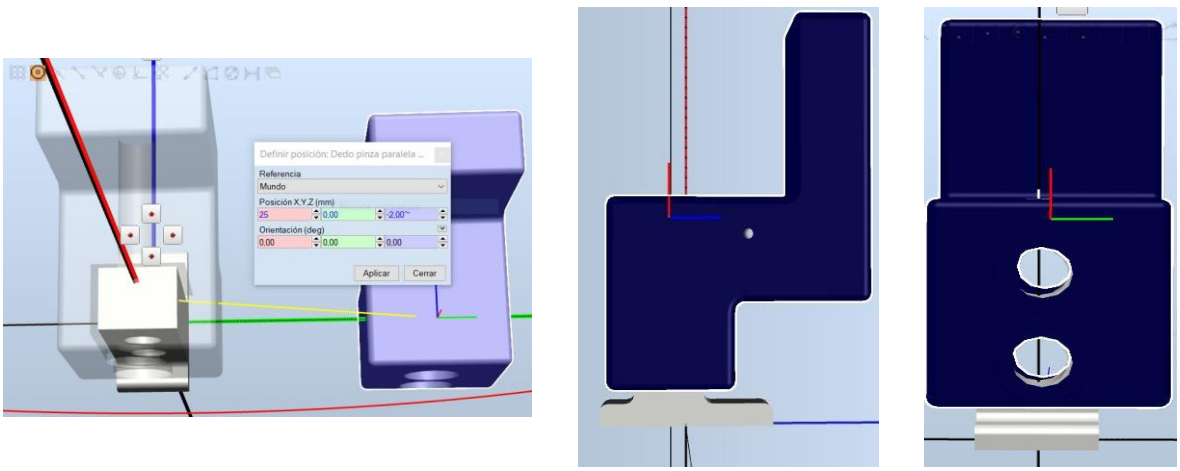


Figura 49: Posicionamiento de dedo en adaptador *RobotStudio*

Por ultimo vamos a hacer que estas dos piezas formen una única geometría llevando el cuerpo de adaptador a Dedo pinza paralela. Esto sirve para definir los eslabones de una vez en la creacion de la herramienta *3.1.4 Creación de mecanismo*. En concreto la unión del dedo y el adaptador se podría haber realizado también en *Inventor* mediante un nuevo un ensamble.

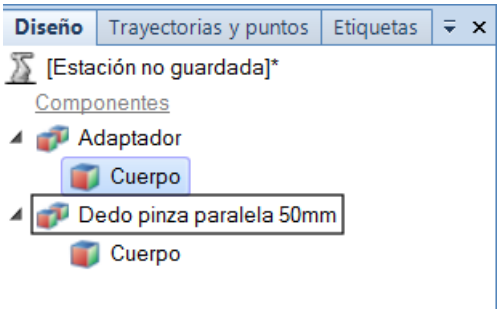


Figura 50: Unir dedo y adaptador en un mismo sólido *RobotStudio*

El resto de la herramienta se monta realizando los mismos pasos según era necesario.

3.1.2 Obtención del eje en el laboratorio

Existe un desfase angular entre el sistema de ejes de la terminación del robot sin herramienta y el eje de cogida de la herramienta. Es por ello necesario hacer un cálculo precio en el laboratorio para saber de cuanto es ese desfase.

Para esto se hizo uso de la *Flexpendant* poniendo todos los ejes a 0º. Tras esto, se colocó en la horizontal la herramienta girando solo el eje de la terminación del robot. Observando la pantalla de la *Flexpendant* podemos ver que existe un desfase de 45º entre ellos.

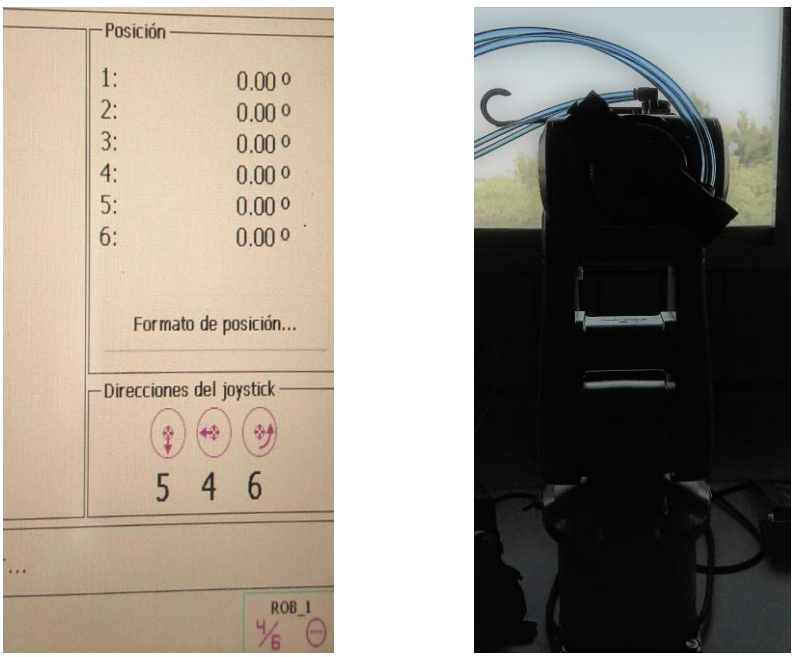


Figura 51: Posición con todos los ángulos a 0º *IRB120*

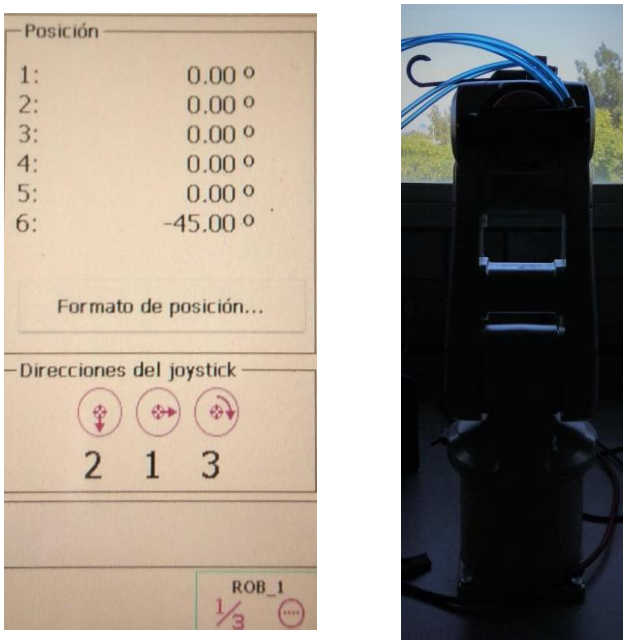


Figura 52: Posición del efector en horizontal *IRB120*

3.1.3 Posicionar origen local de la herramienta

Este apartado es necesario antes de la creación del mecanismo ya que tras ello no se puede editar.

Para esto en primer lugar es necesario saber como están los ejes de la terminación del *ABB*.

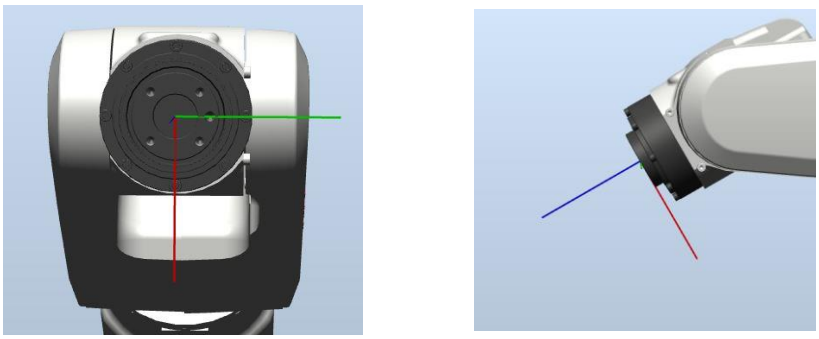


Figura 53: Posición de los ejes de la terminación del *IRB120 RobotStudio*

Al crear el mecanismo, el eje de dicho mecanismo será el asociado al origen mundo, es por ello que hay que posicionar la herramienta en el origen de forma que coincidan los 45º calculados en el punto anterior y los ejes del *TCP*.

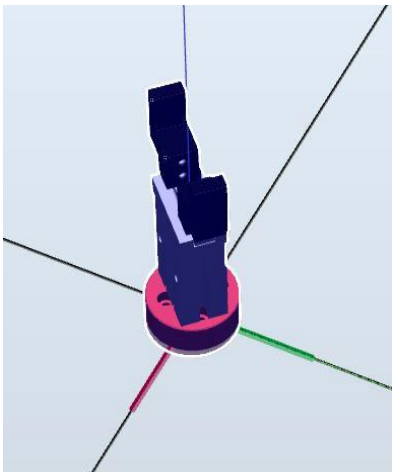


Figura 54: Posición herramienta para coincidencia del *TCP RobotStudio*

3.1.4 Creación de mecanismo

Para la creación de mecanismos *RobotStudio* tiene la herramienta crear mecanismos.

Figura 55: Crear mecanismo *RobotStudio*

Nos aparece la ventana de crear mecanimos, donde en tipo de mecanismo debemos seleccionar herramienta.

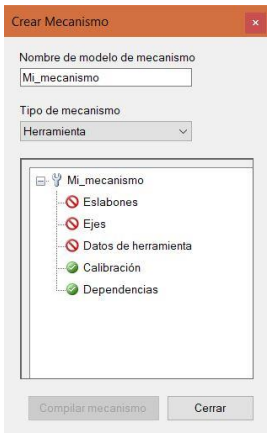


Figura 56: Ventana auxiliar crear mecanismo RobotStudio

Ahora debemos entrar en el apartado eslabones donde indicaremos los diferentes eslabones que componen nuestra herramienta. En nuestro caso tenemos un eslabón base, el cual debemos marcar como tal, y otros dos eslabones más, nuestros dedos de la garra con sus adaptadores.

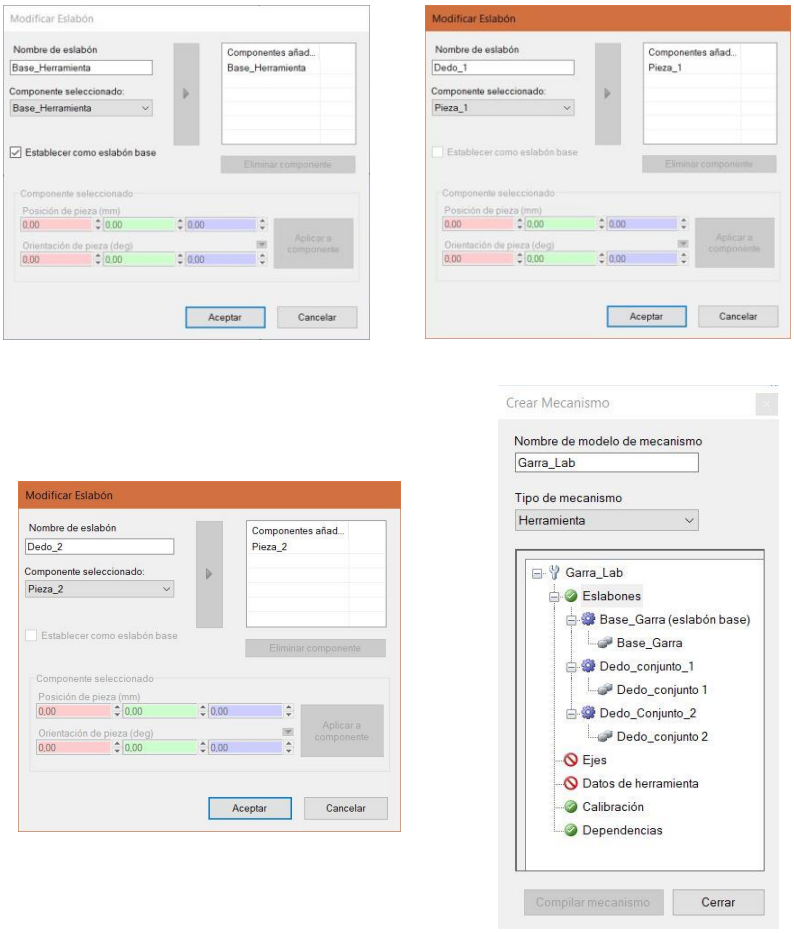


Figura 57: Creación de eslabones *RobotStudio*

A continuación debemos definir los ejes de la herramienta entrando en la siguiente característica de la ventana crear mecanismo.

En el caso que nos ocupa se trata de un eje prismático y además está a 45º respecto el eje *Z*, es por ello que primera posición y segunda posición están calculadas para cumplir con este ángulo. Además el sentido de cada movimiento según el eje es opuesta por lo que están pensadas para cumplir con la misma dirección pero sentidos opuestos (igualmente si se hubiese puesto con el mismo sentido basta con modificar la dependencia de ejes a un factor de -1 en vez de 1).

Es necesario indicar uno de los dos como eje activo.

El límite nos indica cuantos milímetros recorre el dedo según el eje. En nuestro caso 10 mm.

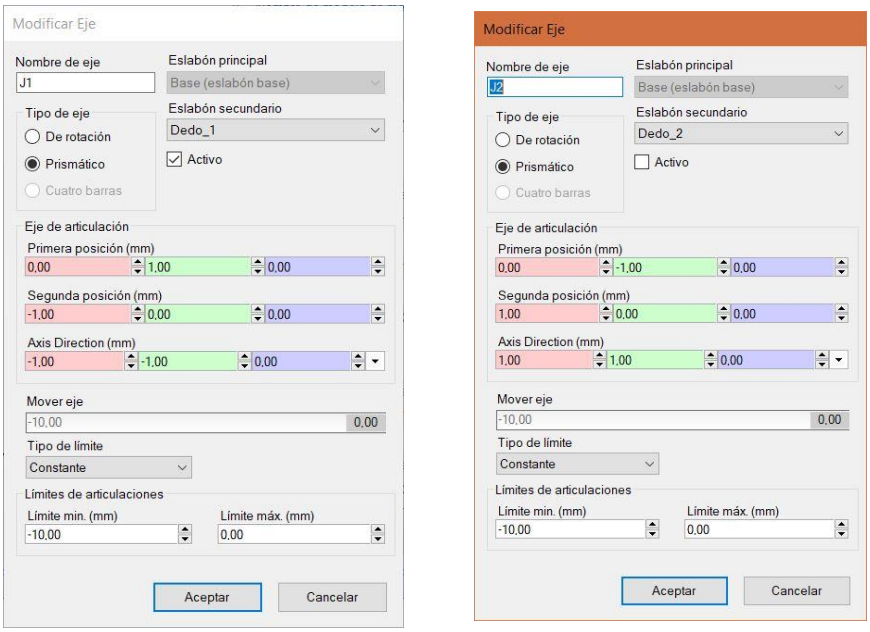


Figura 58: Modificar ejes *RobotStudio*

Las siguientes características de nuestra herramienta están en el apartado *Crear Datos de Herramienta*.

Debemos indicar una posición de *TCP* (*Tool Central Point*) o *PCH* (Punto central de la herramienta), que indica el punto de cogida de nuestra pinta y unos ejes de coordenadas con los que vamos a entrar a cogerlas, en nuestro caso de 107 mm además de un ángulo de 45º ya que esto nos permitirá poder coger las piezas con un ángulo deseado y sin tener que hacer ningún tipo de transformación. El centro de gravedad puede ser calculado con el uso de la *FlexPendant* pero en nuestro caso no es relevante por lo que se supone uno de 65 mm en Z.

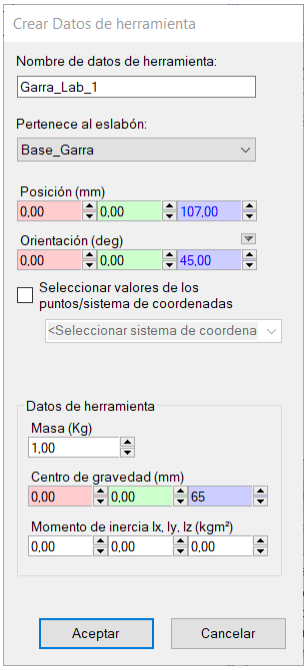


Figura 59: Crear Datos de herramiento *RobotStudio*

Tras estas operaciones nos queda la herramienta de la siguiente forma.

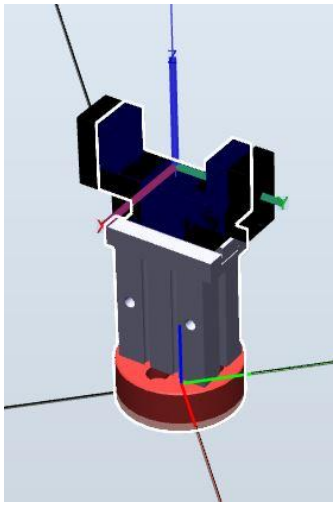


Figura 60: Heramienta terminada *RobtoStudio*

La cual podemos ver que una vez montada en el robot queda con sus ejes y *TCP* perfectamente ubicados.

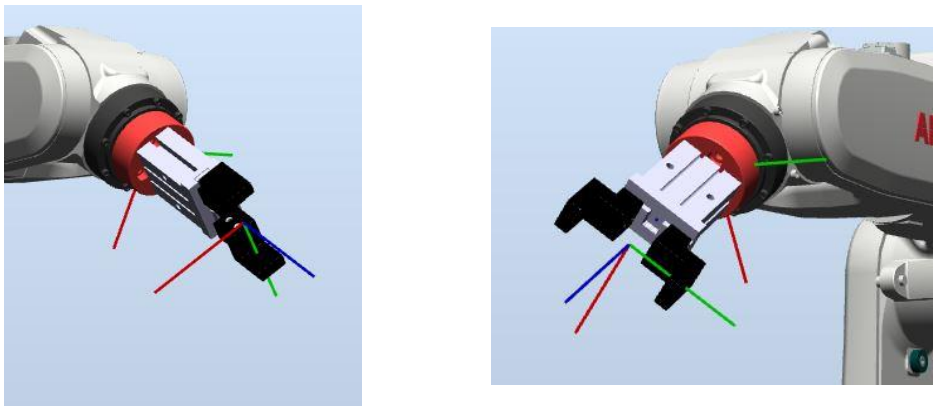


Figura 61: Herramienta montada sobre *IRB120 RobotStudio*

Como siguiente paso tenemos que definir la dependecia. Esta tiene que ser de 1 para que los dos dedos se muevan a la par.

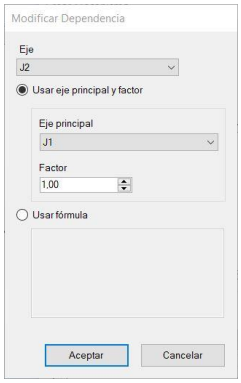


Figura 62:Definir dependencia
RobotStudio

Finalmente en la ventana de crear mecanismo le debemos dar a compilar y ya tendremos nuestra herramienta.

Para comprobar que todo funciona correctamente podemos verlo haciendo click derecho sobre la herramienta y seleccionando movimiento de ejes de mecanismo.

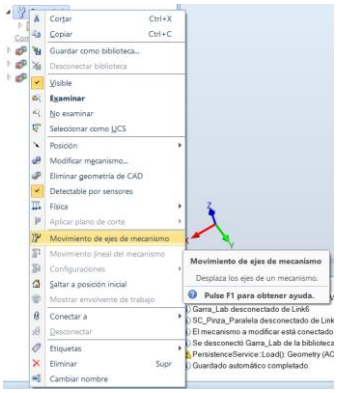


Figura 63: Selección movimiento de ejes de mecanismo *RobotStudio*

Se abrirá una ventana con un *scroll* donde podremos ver si nuestra herramienta abre y cierra correctamente.

Figura 64: Comprobación funcionamiento correcto de cierra y abre *RobotStudio*

3.2 Creación de *SmartComponent* y lógica de la estación

La herramienta creada en el punto anterior "*3.1 Creación de la herramienta"*, no es más que un mero objeto que no puede coger nada, ahora mismo solo es capaz de mostrarnos como movería sus ejes si lo hacemos manualmente. El software de *ABB* nos permite crear un tipo de componente denominado *SmartComponent*, *SC*, el cuál nos permitirá simular como la pinza es capaz de coge los objetos. A su vez la lógica de la estación permitirá crear la simulaciones de una señal con la cual podemos simular las señales reales que hacen que la pinza real se abra o cierre.

3.2.1 *SmartComponent*

Seleccionamos *Componente inteligente* en nuestro menú. Para crear uno nuevo.

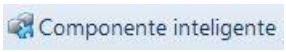


Figura 65: Componente inteligente *RobotStudio*

En la ventana que se abre podemos accede a añadir componentes y agregar los distintos componentes que nos serán útilies en nuestra creación. En nuestro caso se han usado un plane sensor, una puerta lógica *Not*, un *Attacher*, un *Detacher* y por ultimo dos *JointMover*. Además, hay que arrastrar el icono de nuestra herramienta al componente inteligente para que forme parte de él. Por ultimo en el apartado de señales y conexiones hay que crear las señales "*Cierra*" de tipo *DigitalInput* y como valor predeterminado 0. Como opción está la de crear una señal más en este caso *DigitalOutput* que nos indique si la garra coge algo.

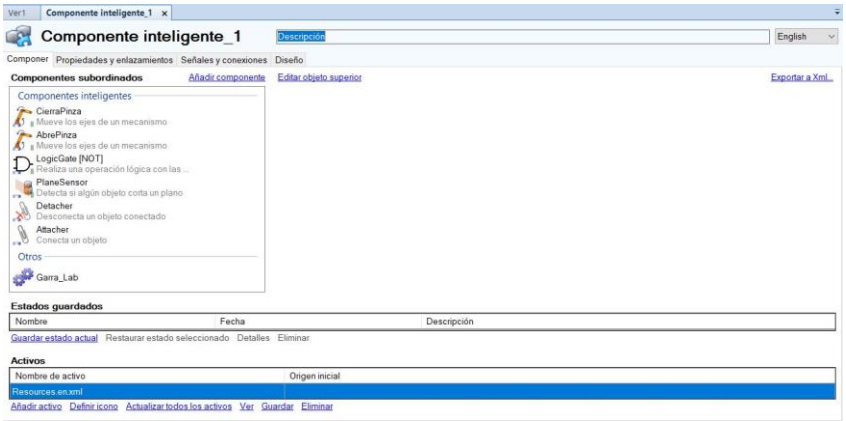


Figura 66: Ventana de *SmartComponent RobotStudio*

A los *JointMover* se les cambiaron sus nombres, a uno se le llamó "*CierraPinza*" y al otro "*AbrePinza*". Para ambos se seleccionó como mecanismo el componente "*Garra_Lab*"."*CierraPinza*" es editado para que lleve al eslabón principal hasta la posición de 9 mm y con un tiempo de cierre de 0.2 segundos, este ultimo valor es estimado y no es relevante. Los mismo se hizo con "*AbrePinza*" pero en esta ocasión con duración de 0 segundos y moviendo el eslabón principal a la posición 0 mm de su eje de movimiento.

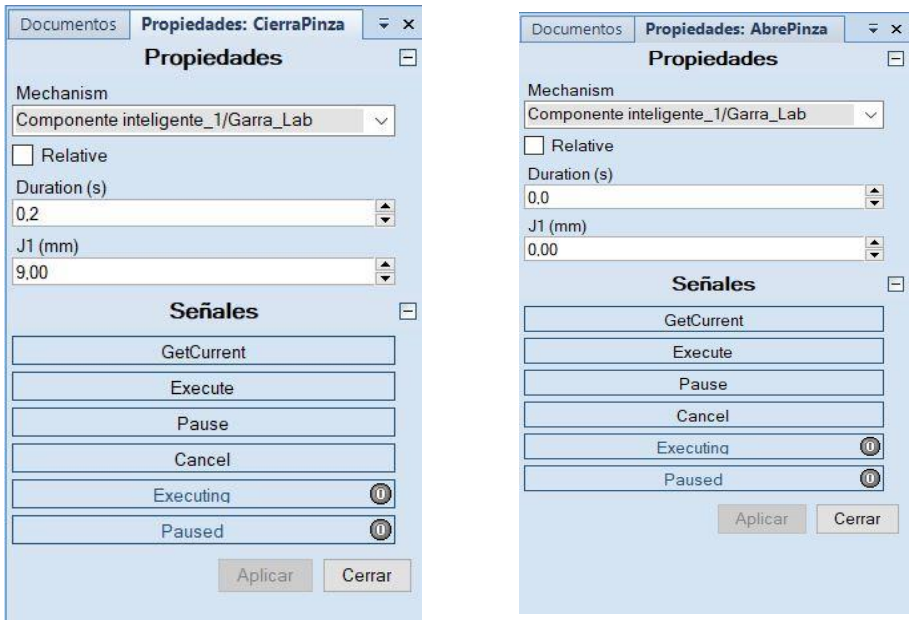


Figura 67: Propiedades CierraPinza y AbrePinza *RobotStudio*

La puerta lógica se configure tipo *NOT*.

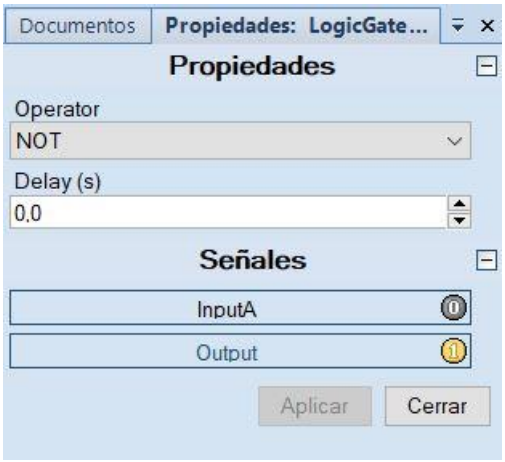


Figura 68: Propiedades puerta lógica NOT *RobotStudio*

Por ultimo se editó el *PlaneSensor*. Este componente cuando se encuentra con un objeto nos permitirá interactuar con él. En nuestro caso nos es de utilidad para que la pinza agarre un objeto cuando la cerramos. Para hacerlo tuvimos que crear un plano paralelo a una de las caras de la pinza por donde vamos a coger las piezas y además vincular el movimiento del movimiento del *SmarComponent* a ese dedo de la pinza para que se mueva con él. De esta forma conseguimos que en simulación el robot agarre piezas.

En primer lugar debemos posicionar el plano paralelo a uno de las cara de agarre de uno de los dedos, debe estar lo suficientemente pegado a la cara pero sin llegar a tocar, ya que si tocase siempre detectaría que hay algo y por lo tanto no funcionaría adecuadamente.

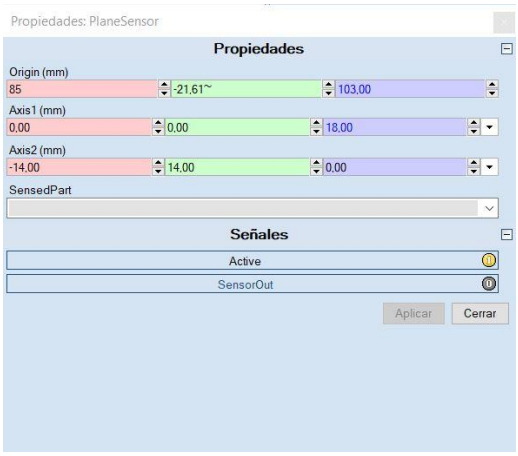


Figura 69: Propiedades PlaneSensor
RobotStudio

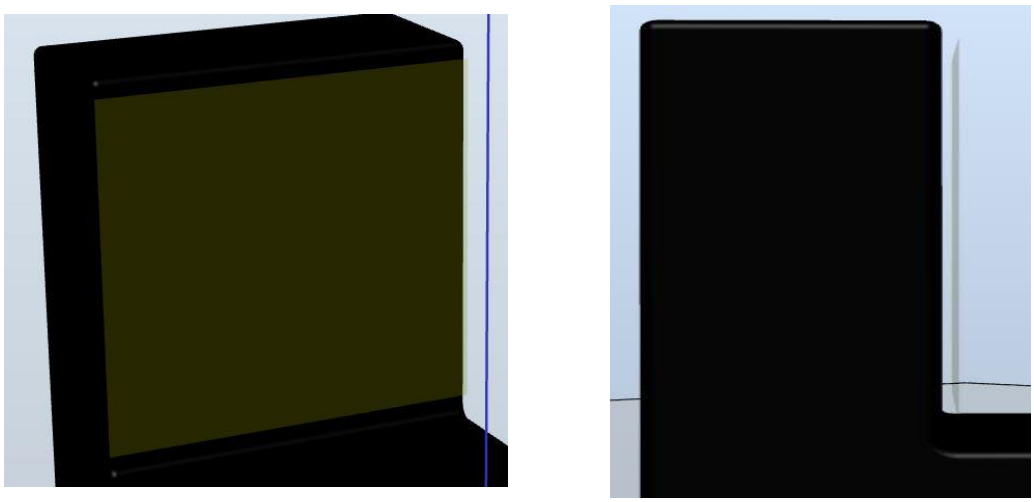


Figura 70: PlaneSensor sobre dedo *RobotStudio*

Por ultimo, para mover el plano con el dedo es necesario asociarlo a éste ultimo para ello hay que localizar el dedo siguiendo la secuencia *Garra_lab>Eslabones>Dedo_1* y una vez localizado arrastrar el PlaneSensor al interior.

Figura 71: Llevar PlaneSensor a dedo
RobotStudio

Para la comprobación de que hemos situado bien el plano y que está asociado, se recomienda al lector mover los ejes del eslabón y comprobar que efectivamente el *PlaneSensor* se mueve con el dedo.

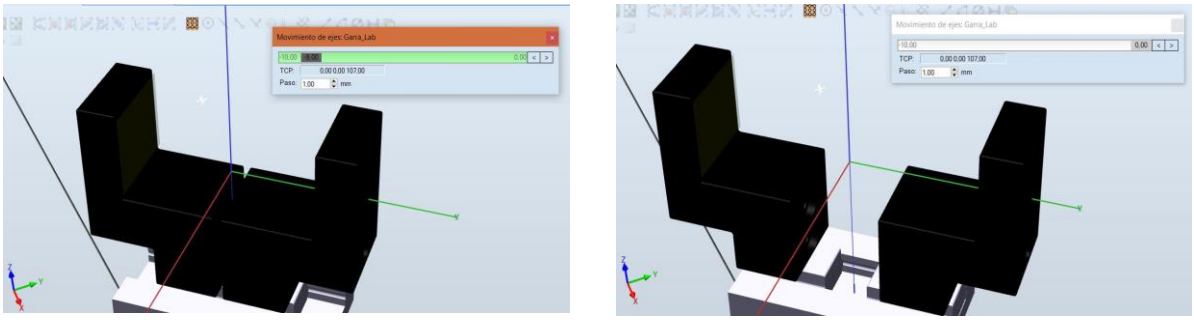


Figura 72: Comprobación de PlaneSensor *RobotStudio*

Entramos ahora en la ventana *Diseño*. En un principio nos aparecerán todos los componentes desordenados. Una vez ordenados el diseño del componente inteligente quedaría de la siguiente forma.

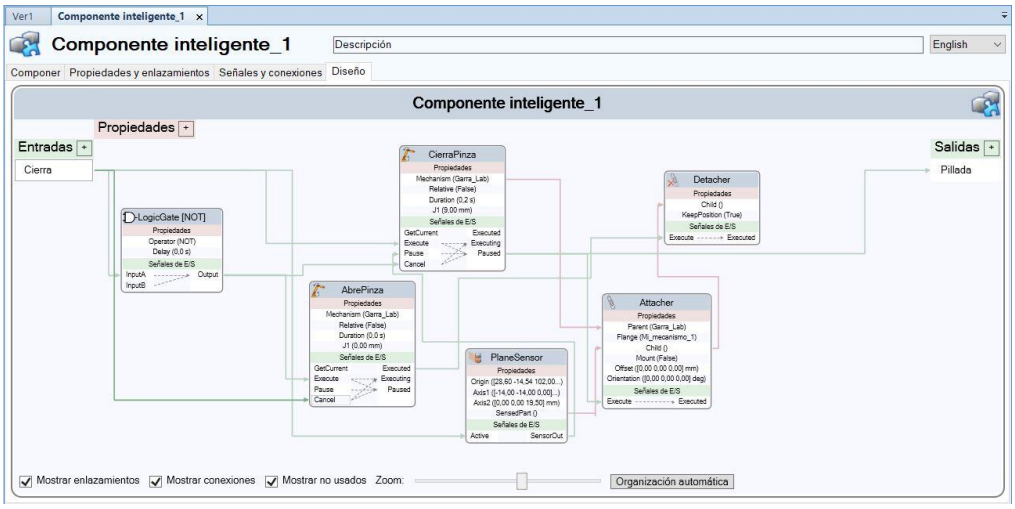


Figura 73: Ventana diseño de SmartComponent *RobotStudio*

El cierre o apertura de la pinza según la señal "*Cierra*" se realiza mediante los elementos "*CierraPinza*","*AbrePinza*", la puerta lógica *NOT* y la señal de entrada "*Cierra*". Conectamos la señal *"Cierra"* con *Execute* de "*CierraPinza*", con lo que se ejecutará el cierre de la pinza cuando se active *"Cierra"*. A su vez debe de interrumpirse "*AbrePinza*," es por ello que se conecta también la señal a *Cancel* de "*AbrePinza*".

El funcionamiento para abrir se basa en que la señal *"Cierra"* esté desactivada. En ese caso pasaremos por la puerta *NOT* que hará que haya señar por su *OutPut* y éste se conectará al *Cancel* de *"CierraPinza*", cancelando el cierre y a *Execute* de *"AbrePinza"*, ejecutando la apertura.

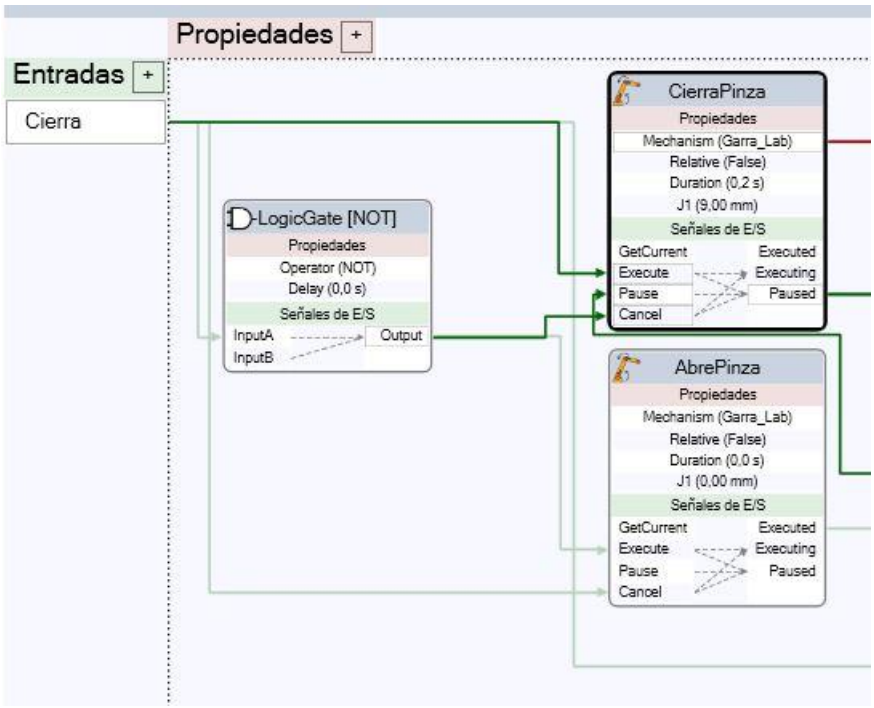


Figura 74: Detalle ventana diseño sobre apertura y cierre *RobotStudio*

Si *"Cierra"* está activo significa que podemos coger algo. Por esto se active el *PlaneSensor* tener señal en "*Cierra"*. El *PlaneSensor* si detecta algo mandará una señar por *SensorOut* que llegará a "*CierraPinza*" haciendo que cancele el cierre ya que hemos detectado algo. A su vez al ser cancelado debemos hacer un *Attacher* para coger el objeto. Como *Parent* se debe elegir el mecanismo que viene de "*CierraPinza*", la pinza, y como *Child* el objeto detectado por el *PlaneSensor*, conexión *SensedPart()* con *Child()* del attacher.

Si abrimos significará que estamos soltando si hemos cogido algo, para ello ejecutamos el detacher si "*Abrepinza*" está ejecutado y se desvinculará del objeto cogido por el attacher conectando el apartado *Child()* de este ultimo con el *Child()* de *Detacher*.

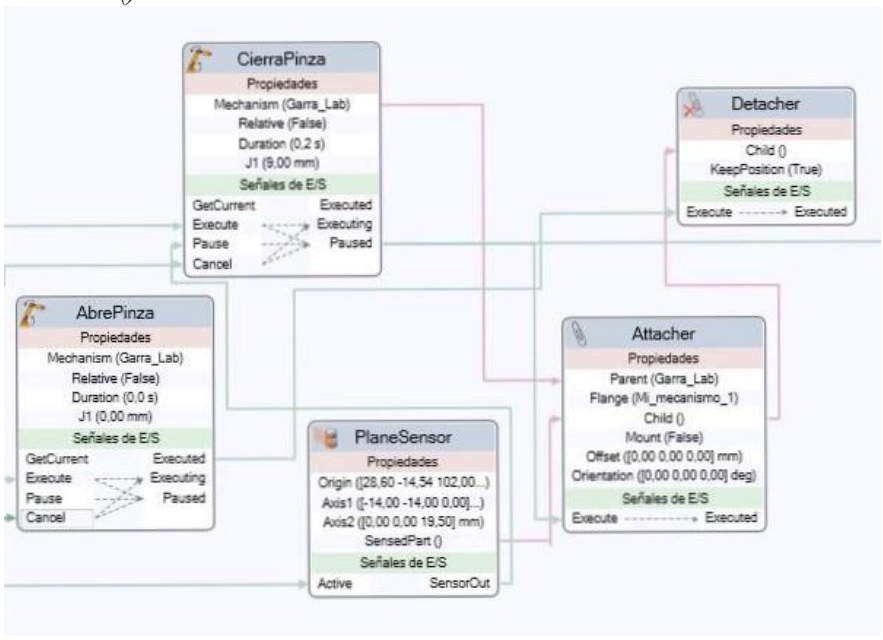


Figura 75: Detalle pestaña diseño *RobotStudio*

Como opción para pruebas se conectó el paused de "*CierraPinza*" a la señal "*Pillada*", utilizada para pruebas, que nos indicará si hay algo cogido, para ello debe haber sido pausado el cierre de la pinza por culpa del *PlaneSensor*.

Ahora debemos programar las señales de entradas y salidas simuladas. Accedemos a *Controlador>Sistema>Configure>I/O System.*

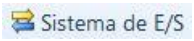


Figura 76: Sistema de E/S *RobotStudio*

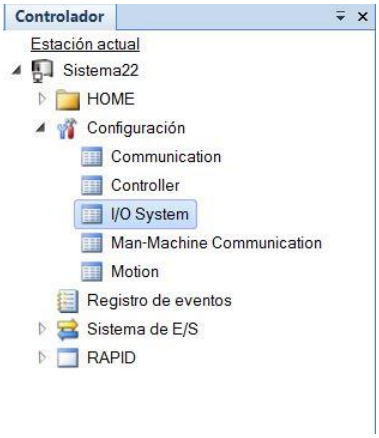


Figura 77: Seleccionar I/O System RobotStudio

Dentro de la *DeviceNet* creamos una nueva unidad hacienda click derecho sobre ella. Creamos una nueva unidad *combi* digital de entradas y salidas por la que irá asociada nuestras señales.

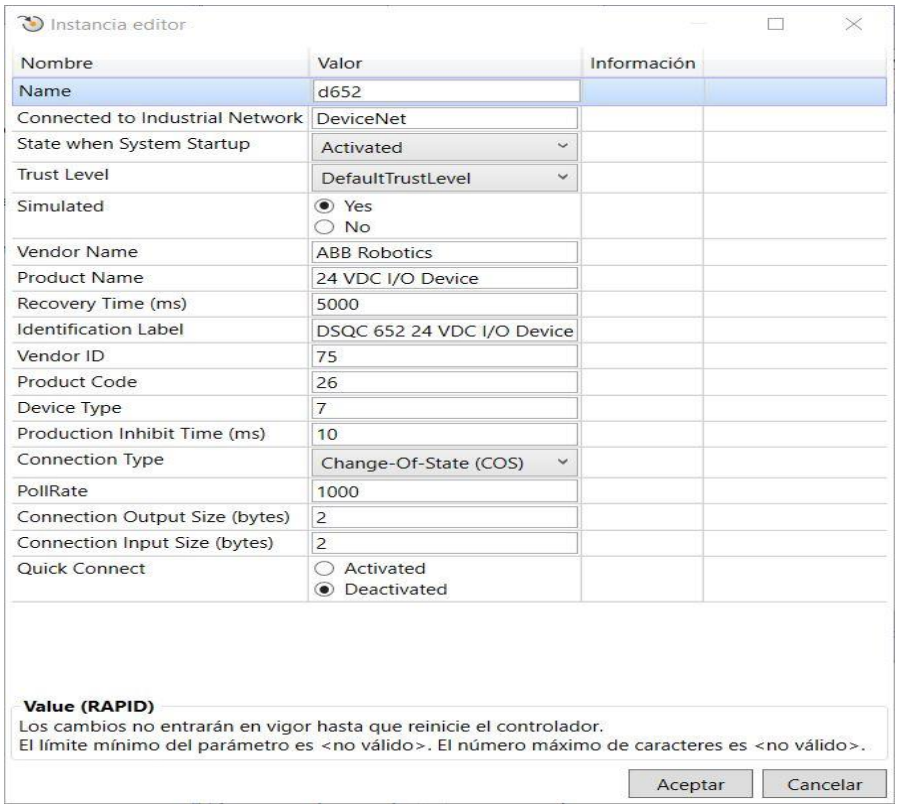


Figura 78: Creación de nueva unidad *combi RobotStudio*

Ahora desde la pestaña señales podemos crear nuestras señales "*CierraPinza*" y "*Pillada*" asignada al *Combi d652* y con *Device Mapping* 0. En el *pop-up* de nivel de acceso marcar *ALL* para poder utilizarlo en simulación.

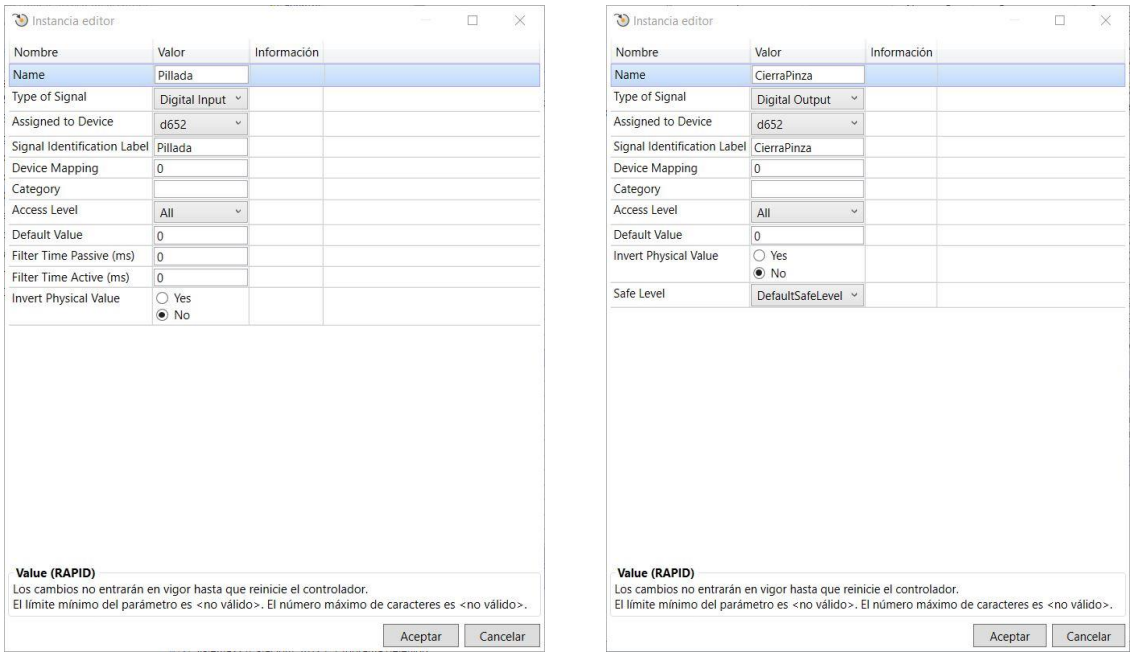


Figura 79: Creación de señales Pillada y CierraPinza *RobotStudio*

Ya creadas las señales es hora de darle lógica a la estación. Con este ultimo paso, al hacer uso en *RAPID* de la señal "*CierraPinza"* se ejecutará *"Cierra"* del *SmartComponent*. Entramos en *Lógica de estación* y vamos a la pestaña *Diseño*.

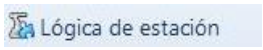


Figura 80: Lógica de estación
RobotStudio

Figura 81: Ventana lógica de la estación *RobotStudio*

Estableciendo las conexiones de la imagen hacemos que si *"CierraPinza"* está activo se ejecute *"Cierra"* del *SmartComponent*. Por otro lado el *SmartComponent* cuando ejecute "*Pillada*" mandará una señal para que se ejecute la señal "*Pillada*" simulada.

3.3 Posicionar robot y elementos de la estación

Situar los diferentes components de la estación en la posición adecuada puede llegar a ser una tarea engorrosa, es por ello que se trabajó utilizando *WorkObjects* y estableciendo su uso por el usuario, así podremos posicionar los objetos relaticos a otros con los que nos sea agradable trabajar.

De igual forma, la creación de *WorkObjects* nos sirve tambien para definir respecto que base de trabajo va a estar moviendose el robot. En nuestro caso coincidirá, o al menos en principio, con la creada con la *Flexpendant* en el sistema real, ver *"4.3 Definir WorkObject con Flexpendant"*. Cabe decir que la creación de un *WorkObject* donde trabajar hace que el robot sepa como ha de moverse según como esté este situado respect al propio robot.

Un último paso es la conexión de nuestro *SmartComponent* al robot.

3.3.1 Posicionar mesa y robot

En realidad, la mesa la podriamos haber situado en el sitio que nos plazca, a partir de la situación de la mesa en el *Workobject* mundo podremos definir un sistema de coordenadas más útil. En nuestro caso, se decidió situar en una esquina del tablero de la mesa un *WorkObject* que facilite el posicionamiento del resto de objetos.

Una vez colocada la mesa donde queramos, por ejemplo en el origen del *WorkObject* mundo y respecto a valores positivos de los ejes, podemos crear el *WorkObject* en una de las esquinas del tablero superior, de forma que el resto de objetos que se vayan a posicionar sobre la mesa lo hagan relative a un plano y sea mucho más cómoda su colocación. En primer lugar creamos una nueva posición con el botón *Posición* y seleccionando la esquina del tablero.

Figura 82:
Posición
RobotStudio

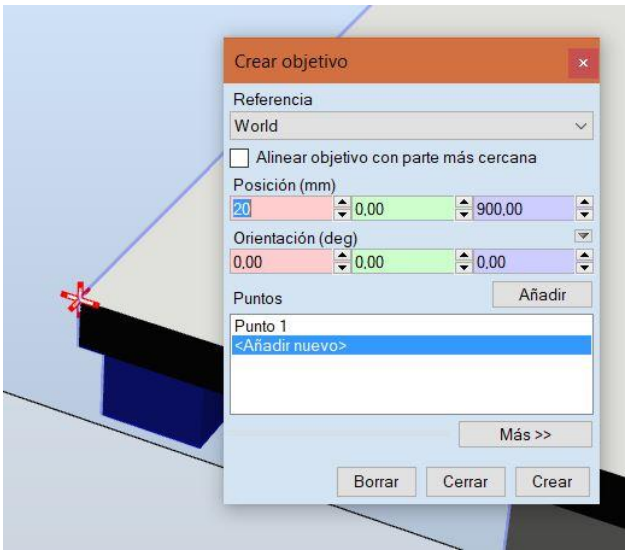


Figura 83: Crear objetivo para *WorkObject_Mesa*
RobotStudio

De esta forma se crea una nueva posición en el objeto de trabajo actual predefinido por *RobotStudio*, "*wobj0"*, aparece en la siguiente imagen con el nombre de *"Target_10"*.

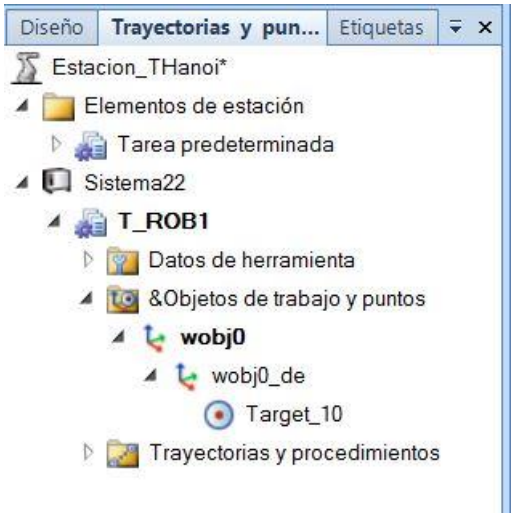


Figura 84: Localización de *Target_10 RobotStudio*

Un paso previo a continuar con la creación del *WorkObject* es orientar la posición creada según como vayamos a querer que estén localizados nuestros ejes del *WorkObject*. Con la herramienta de giro podemos modificar esto.

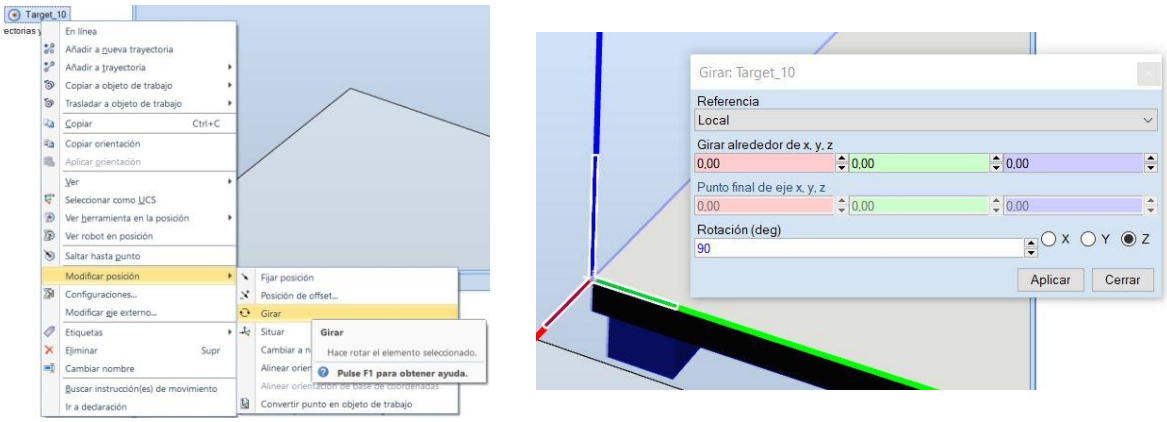


Figura 85: Girar *Target_10 RobotStudio*

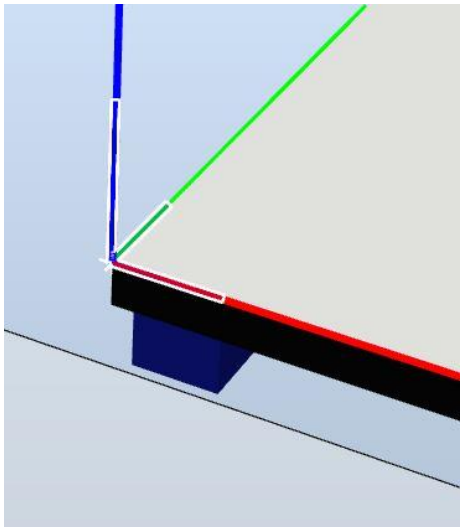


Figura 86: Posición correcta de *Target_10 RobotStudio*

Con esto ya está listo para ser convertido a objeto de trabajo, obteniendo un nuevo *WorkObject*.

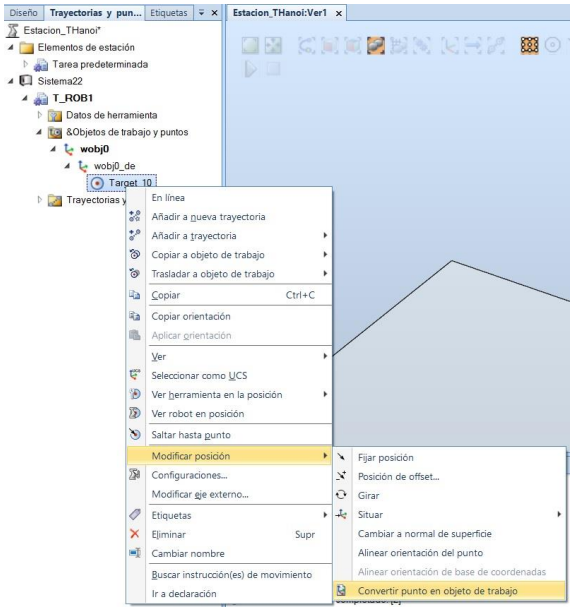


Figura 87: Selección convertir punto en objeto de trabajo *RobotStudio*

Una vez tenemos nuestro *WorkObject*, para poder posicionar objetos respecto a él, necesitamos seleccionarlo como *UCS*.

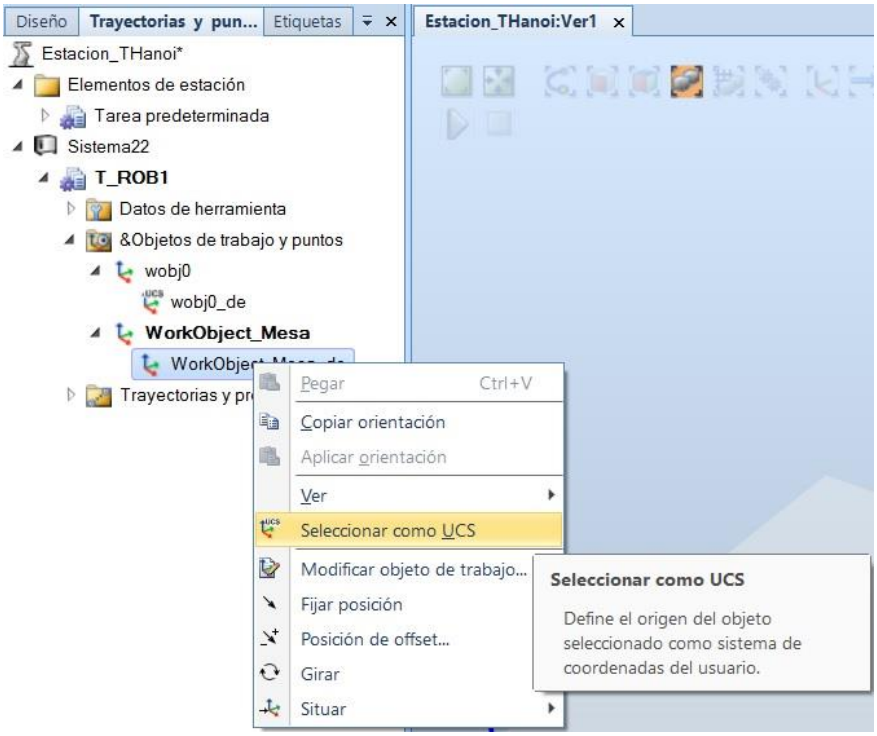


Figura 88 Seleccionar como UCS *RobotStudio*

Ya con el *WorkObject* en el vértice de la mesa y habiendo sido seleccionado este como *UCS*, podemos ahora posicionar el robot y la base respecto a él de una forma más cómoda.

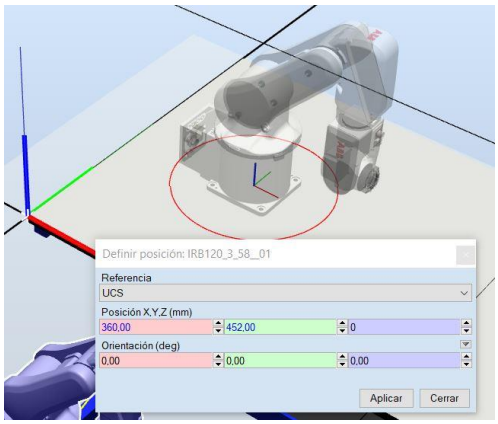


Figura 89: Posicionamiento respecto a UCS *RobotStudio*

Podríamos igualmente haberlos situaos usando el sistema de referencia *"Mundo"*, pero parece más adecuado usar el proceso anterior para situar objetos sobre un plano.

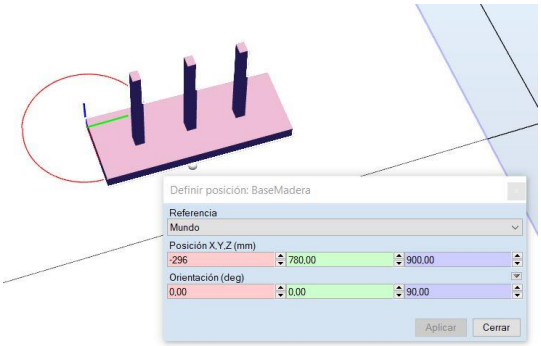


Figura 90: Posicionamiento respecto a mundo *RobotStudio*

3.3.2 Crear y posicionar *WorkObject*

Los pasos utilizados aquí son exactamente los mismos que en el apartado anterior cuando creamos el *"WorkObject_Mesa"*. Aquí, sin embargo, se elige uno de los vertices de la base del juego como posición sobre la que crear el *WorkObject*. Éste será el que usemos cuando programemos. Podemos observar que posicionar una objeto de trabajo de esta forma nos facilita mucho la comprensión de los movimientos relativos que se harán en el juego y la definición de objetivos. De igual forma, definida la resolución respecto uno de estos sistemas de coordenadas, podemos, en el caso que se quisiera, mover la base a otra posición y diciendo donde se localiza, el origen de la base, la programación debe ser la misma.

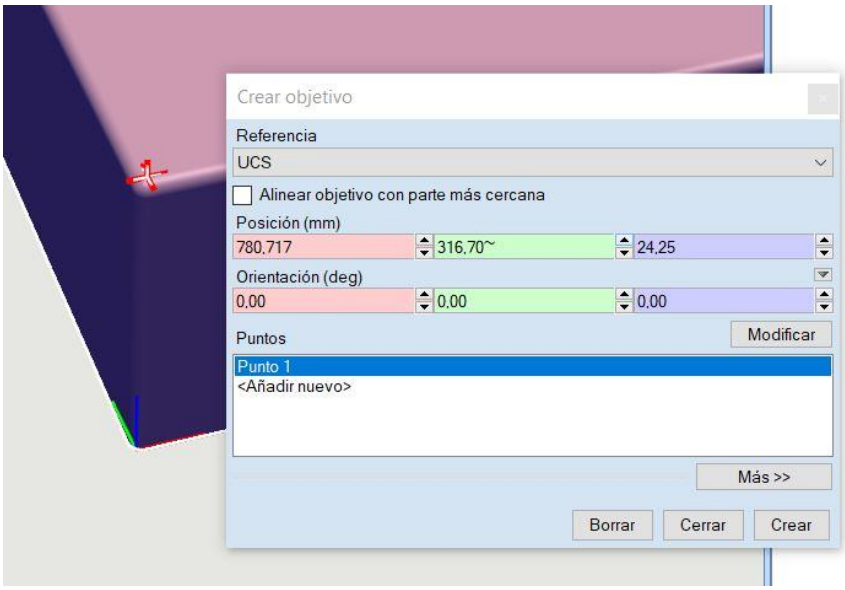


Figura 91: Creación de WorkObject del juego *RobotStudio*

Como ya tenemos definido el objeto de trabajo de la base del juego, es facil posicionar las fichas del juego estableciendo su origen local en el centro de sus bases y tras ello llevarlas a la posición del eje de torre.

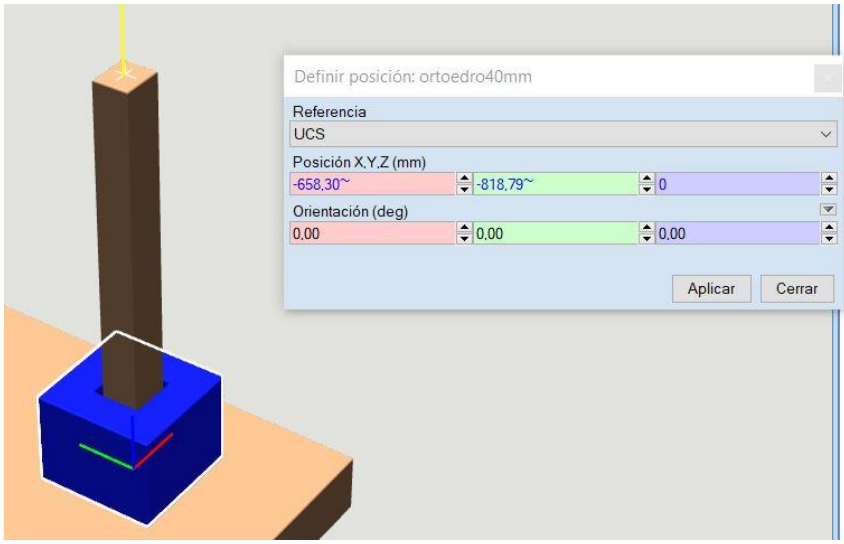


Figura 92: Posicionar uno de los bloques *RobotStudio*

Posicionamos de forma similar el resto de bloques.

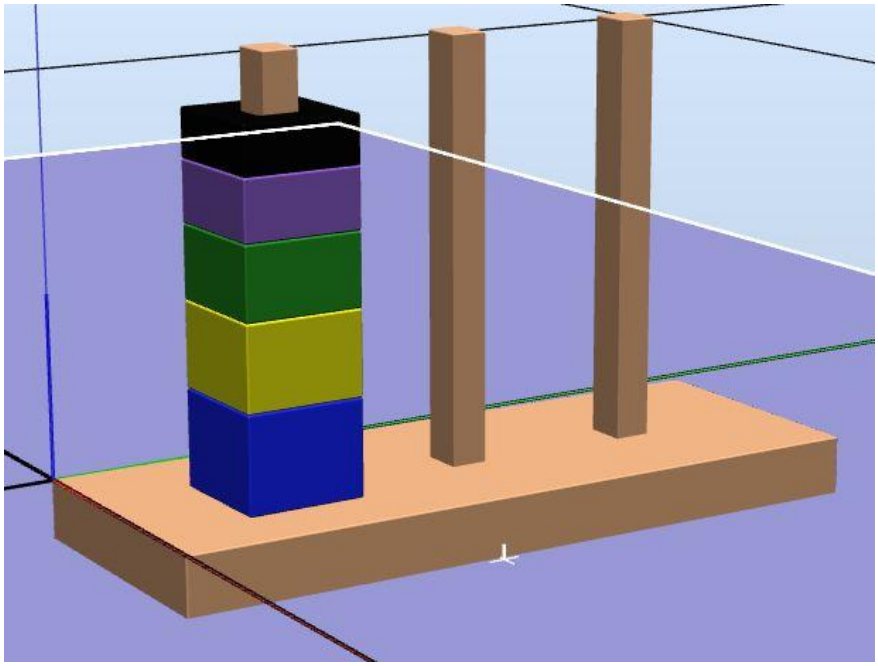


Figura 93: Juego completo posicionado *RobotStudio*

3.3.3 LLevar el *SmartComponent* "pinza" a robot

Debemos arrastras el *SmartComponent* al *IRB120* para que se coloque en la terminación, recordemos que el origen del *SmartComponent* coincide con el de la herramienta que fue calculado de forma que la herramienta no queda forma arbitraria. Por lo que decimos sí a actualizar la posición.

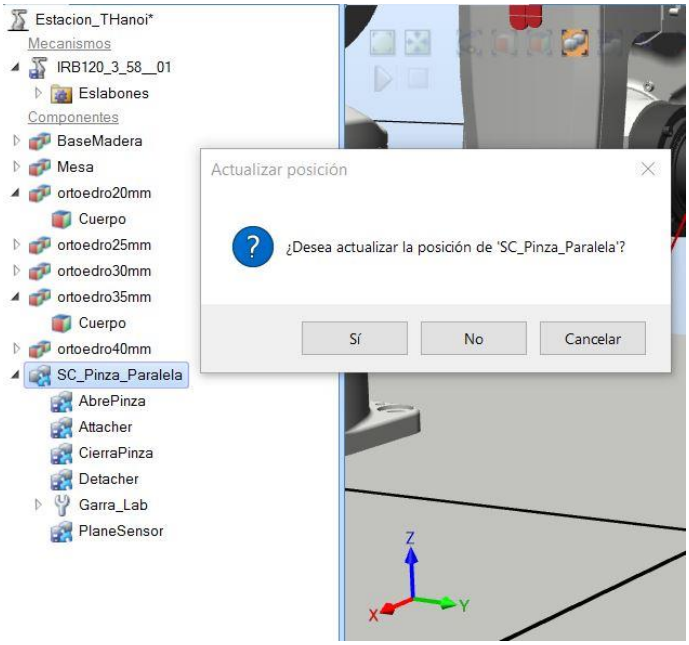


Figura 94: Actualizar posición del *SmartComponent* a *IRB120 RobotStudio*

3.4 Pasos finales y sincronización con *RAPID*

Llegados aquí ya tenemos nuestros objetos de la estación perfectamente colocados.

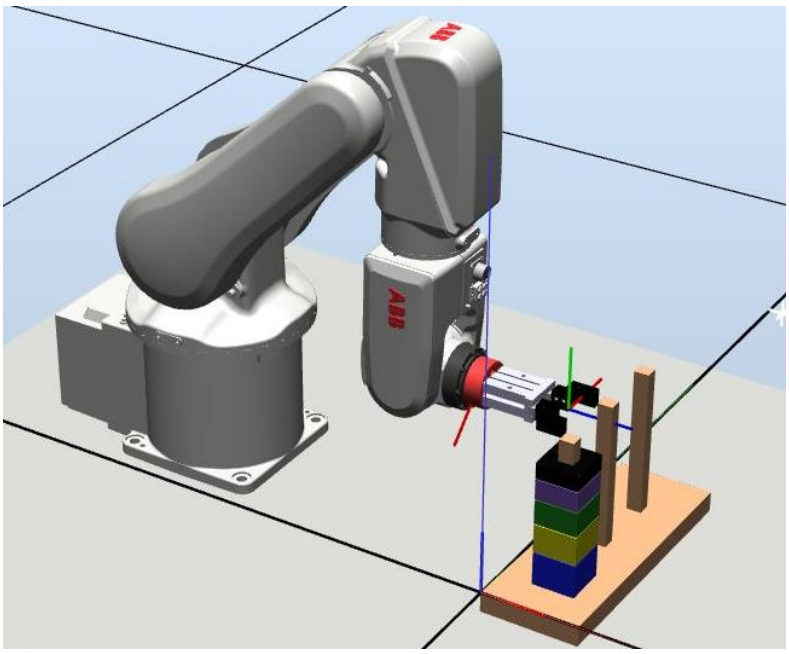


Figura 95: Estación final colocada *RobotStudio*

Además se crearon posiciones de apoyo a la programación, con el propósito de no tener que medir coordenadas e introducir manualmente las posiciones. Para que estas pasen a *RAPID* es necesario crear una trayectoria y llevar los puntos a ella

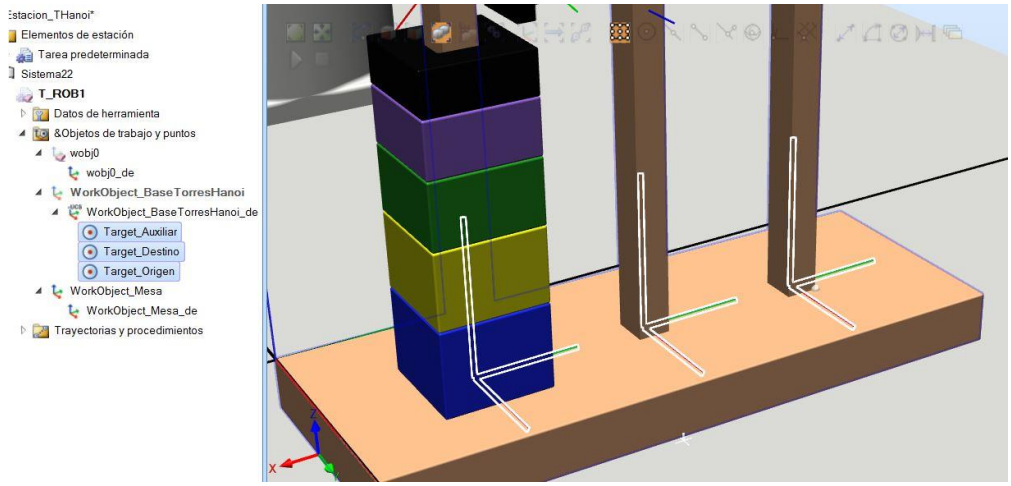


Figura 96: Creación de posiciones de apoyo *RobotStudio*

Figura 97: Trayectoria con prohibidos *RobotStudio*

Observamos que en primera instancia tenemos un icono de dirección prohibida sobre los movimientos de la trayectoria. En un principio puede parecer que todo está correcto, pero la herramienta no puede coger los bloques ya que el robot no puede llegar a colocar la herramienta de forma que su *TCP* coincida con el punto. Es por ello que hay que cambiar la orientación de los puntos haciendo que sea la adecuada para entrar a coger los bloques.

Para que la herramienta coja correctamente las piezas, sin tener que hacer ninguna transformación, tenemos que definir estos objetivos con una orientación que coincida con el *TCP* de la herramienta ya que de esta forma va a coger los bloques. Giramos uno de nuestros puntos hasta hacerlo coincidir con los ejes del *TCP* y tras esto copiamos su orientación y se las damos al los otros dos objetivos.

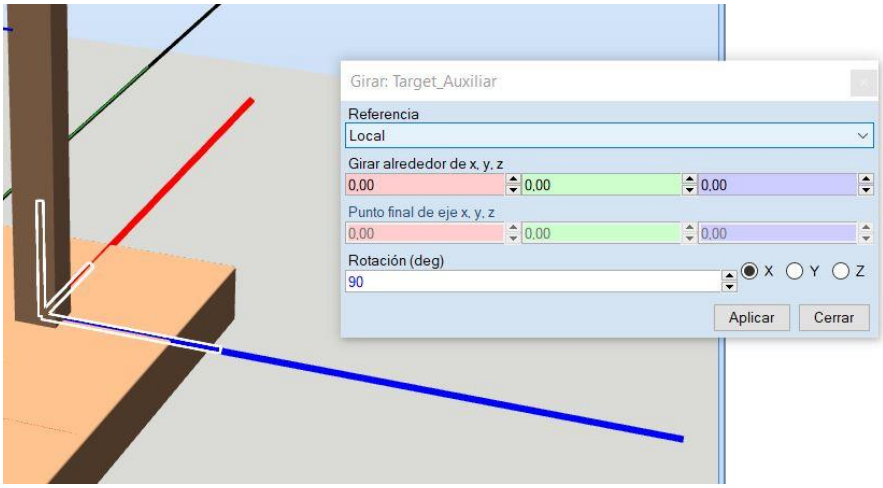


Figura 98: Girar posición para tener correcta orientación para cogida *RobotStudio*

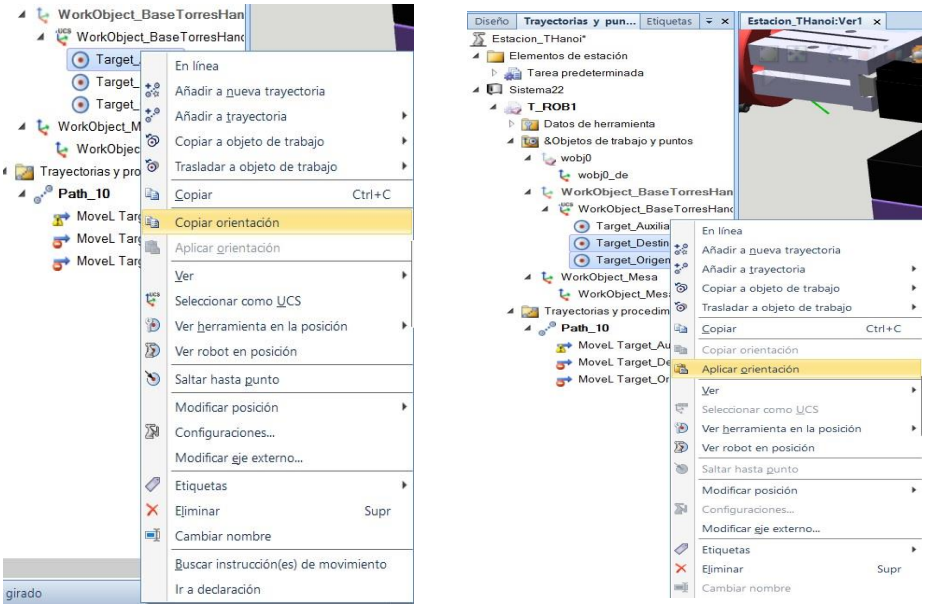


Figura 99: Copiar y aplicar orientación *RobotStudio*

Por último sincronizamos nuestra estación con el controlador. Inmediatamente se crea el código en *RAPID* que define nuestra herramienta, los distintos *WorkObjects* y los puntos objetivos que estarán en un proceso, en este caso el proceso "*Path_10*".

Figura 100: Sincronizar *RobotStudio*

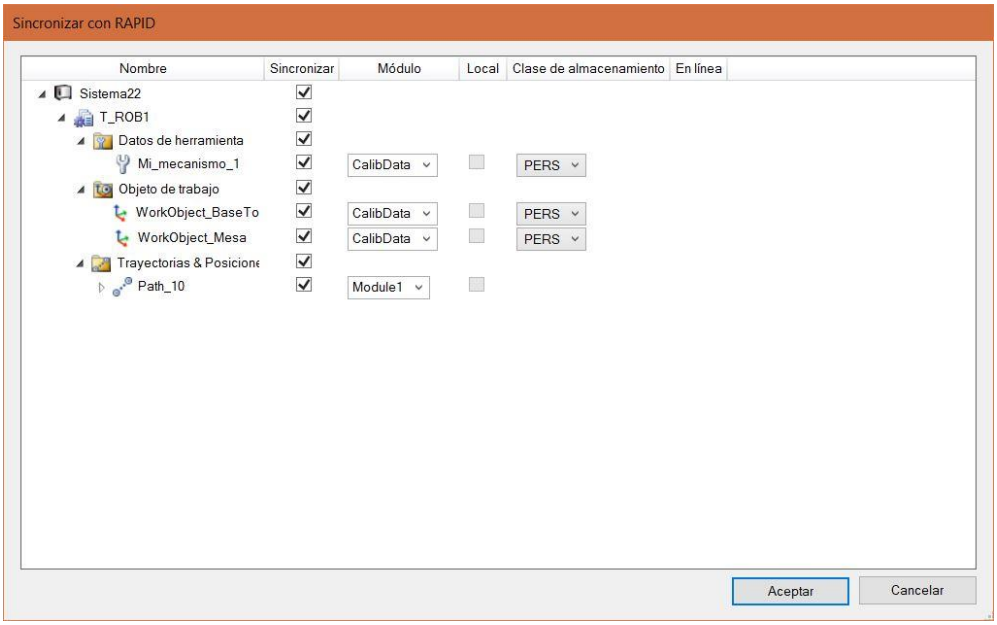


Figura 101: Ventana de sincronización con RAPID *RobotStudio*

4 FLEXPENDANT

El *IRB120* posee una controladora portátil denominada *Flexpendant*, con ella podremos gestionar los movimientos del robot, para programar y para crear o modificar parámetros.

Figura 102: *FlexPendant*

Además, *RobotStudio* posee una controladora virtual con la que podremos hacer las mismas operaciones que haríamos con la real, por lo que es muy útil para practicar. Para acceder a ella hay que entrar en la pestaña controlador y seleccionar el icono de la *FlexPendant*. Que hará expandiserse una ventana auxiliar donde está simulada la *FlexPendant*.

Figura 103: Icono FlexPendant RobotStudio

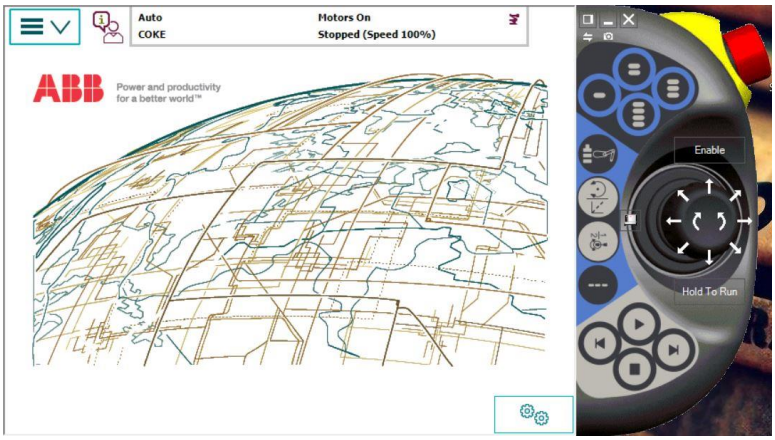


Figura 104: Interfaz *FlexPendant RobotStudio*

Haciendo click en el icono de arriba a la izquierda de la imagen se abrirá un menú desde el cual podemos acceder a las distintas funciones de la controladora.

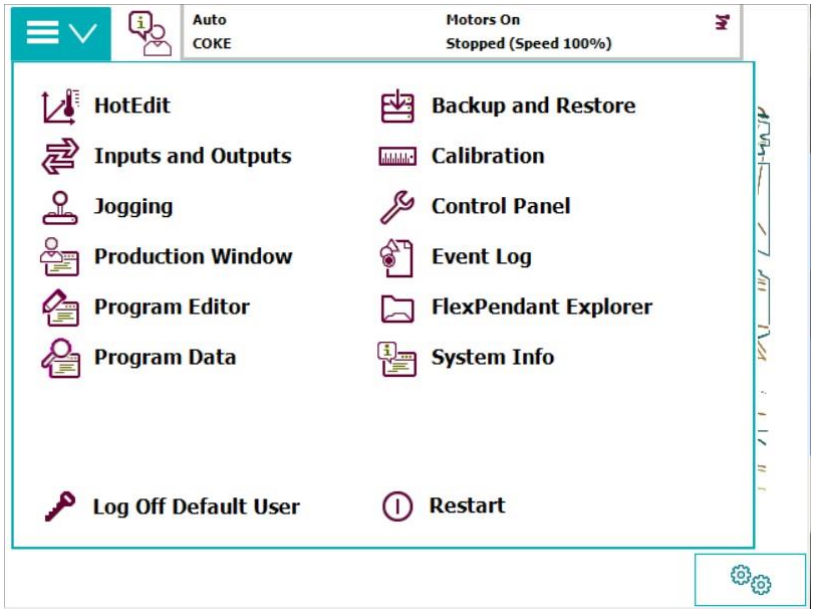


Figura 105: Menú *FLexPendant*

4.1 Manejo básico

Para usar la controladora hemos de saber previamente que dispone de diferentes opciones de tipo de ejecución. Estas son *Auto*, *Manual* y *veloc máxima manual*.

Figura 106: Botonería para control

En la controladora virtual tambien disponemos de estos modos. Para tener acceso a ellos debemos ir a la pestaña de *Controlador* y seleccionar el icono de *Panel de control*. Tras ello podremos seleccionar el modo que queramos.

Figura 107: Panel de control RobotStudio

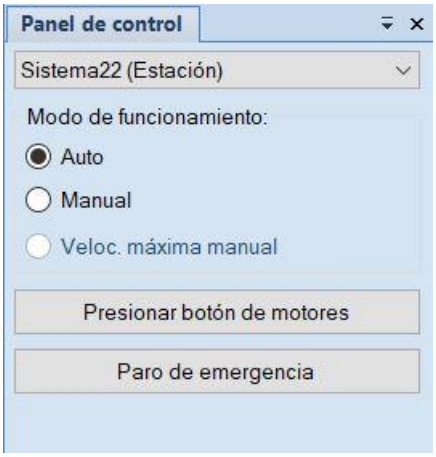


Figura 108: Pesaña Panel de control *RobotStudio*

De vuelta al menu de la *Flexpendant* podemos ejecutar movimientos, sin nada programado previamente, hacienda uso de la opción *Jogging*. Al abrirse aparece una ventana donde tenemos acceso a las opciones de movimiento.

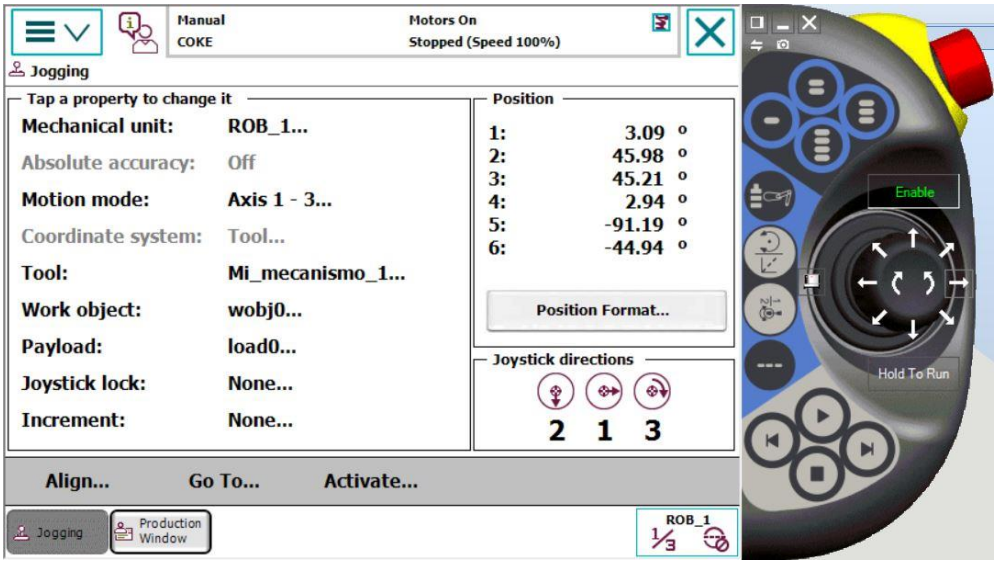


Figura 109: Ventana Jogging *FlexPendant RobotStudio*

A la izquierda podemos ver difrentes opciones. Las más relevantes en este proyecto han sido la de *Tool* y *WorkObject*. Con la primera podemos decir que herramienta tenemos puesta en el robot y con la segunda respecto a que sistemas de coordenadas, o *WorkObject* como se denomina en *RobotStudio*, nos estamos movimiendo. Si clicásemos sobre ellos nos aparecerían las diferestes herramientas y *WorkObjets*, deben haber sido creados previamente, se verá en apartados siguientes.

Al seleccionar *Motion Mode,* podemos elegir el tipo de movimiento que vamos a hacer. En el ejemplo de la ventana estaríamos controlando los ejes 1, 2 y 3 y es por ello que en el apartado position nos aparecen los grados a los que están los ejes del robot. Otra opción sería mover los ejes 4, 5 o 6 o incluso movernos de forma rectilínea, lo cual es bastante interesante. En este caso *Position* nos da las coordenadas en la que se encuentra el *TCP* de la herramienta en coordenadas cartesianas y respecto al *WorkObject* elegido ademas de sus correspondiente cuaterniones.

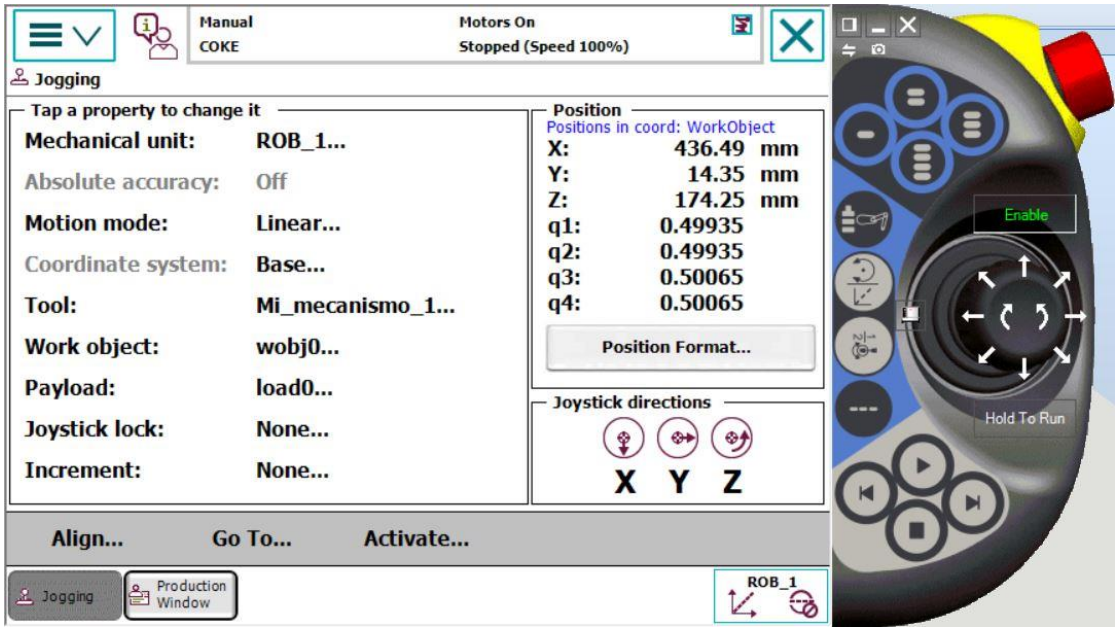


Figura 110: Ventana *Jogging Motion mode linear RobotStudio*

Motion Mode también nos da la opción de hacer movimientos de reorientación sobre el *TCP* que hayamos definido.

Un paso que debemos dar antes de modificar la posición manualmente es habilitar la opción *Enable*. La controladora real tiene un botón en la parte trasera que nos permite habilitar el movimiento si la pulsamos (además recordar que debemos haber seleccionado *Manual* en el panel de control) y si liberamos el botón deja al robot muerto, sin poder hacer movimientos.

Hacinedo uso del *joystick* podemos mover el robot a nuestro antojo según la opción seleccionada y teniendo en cuenta sus limitaciones de movimiento. Podemos de igual manera elegir la velocidad a la que se realiza este movimiento pulsando en el icono de abajo a la derecha de la pantalla, en el se muestran diferentes opciones pero en concreto la segunda nos permite modificar la velocidad.

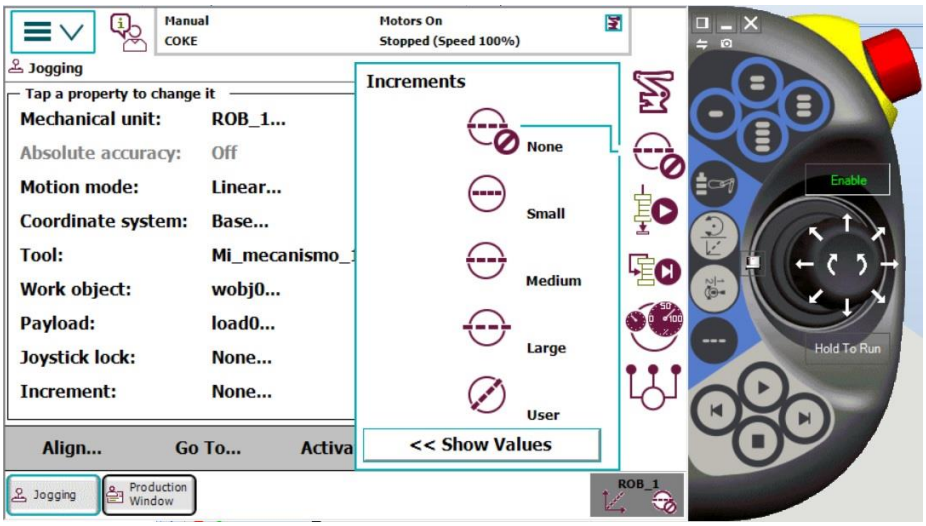


Figura 111: Menú cambio de velocidades *FlexPendant RobotStudio*

Si en el menú despegable pulsasemos el icono *Production Window*, nos aparecería nuestro programa cargado en la controladora. Para ejectar un programa lo normal es indicar un puntero a *main* mediante el botón *PP to main*. Indicado la dirección del puntero y seleccionado *Enable* podemos ejecutar nuestros programa haciendo click en el botón *Play*. Recordemos que podemos dejar el robot muerto si liberamos *Enable*, muy importante para salvar errores.

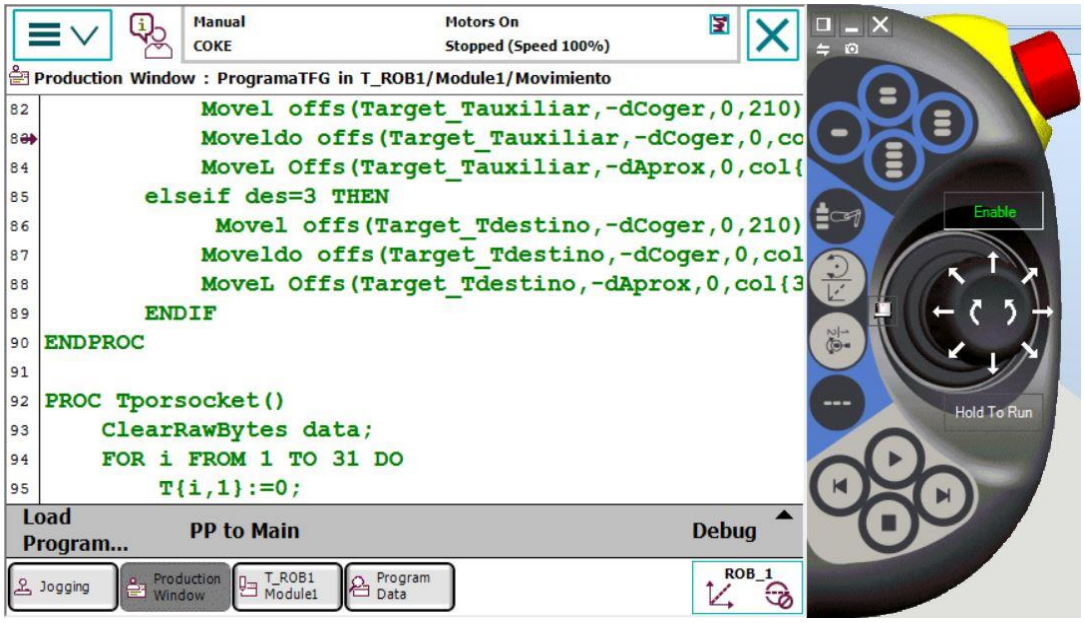


Figura 112: Ventana de producción *RobotStudio*

En el menú podemos encontrar más opciones pero además de las explicadas y *Program Data* que se explicará en los siguientes puntos, el resto no hicieron falta en este proyecto, para más información consultar el manual de la *Flexpendant* , ver "*Bibliografía"*.

4.2 Definir herramienta

Para definir una herramienta es necesario seleccionar *Program Data* en el menú desplegable y pinchar en *tooldata>showdata*. Ahora aparecen las herramientas que tengamos creadas.

Si queremos crear una nueva le debemos dar a *New*. En nuestro caso creamos una con nombre *"Pinza"*, que sea vista globalmente, tipo de dato persistente, para el sistema *"T_ROB1"* y creada en el modulo de programación *CalibData*.

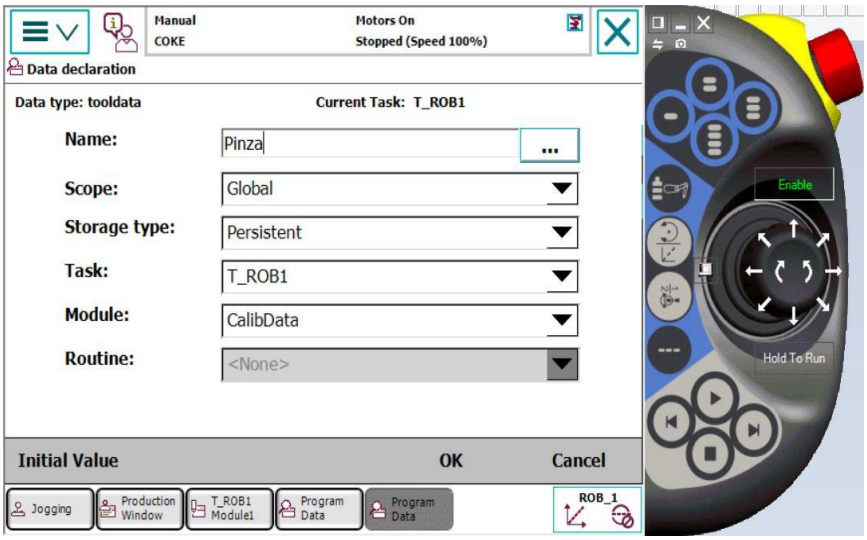


Figura 113: Creación de tooldata *FlexPendant Robotstudio*

Una vez creada podemos editar la definición de la herramienta pulsando en *edit>define*. Seleccionando el método *TCP* por cuatro puntos se nos da la posibilidad de definir la herramienta de una forma práctica. Para ello el *software* tiene programado un método según el cual nosotros le damos cuatro puntos y él inmediatamente calcula la herramienta. Los puntos no pueden ser cuales se quiera, se supone un *TCP* en nuestro caso donde queremos coger la pieza, y se lleva su origen a un punto en el espacio. Y además debemos acceder a ese punto en posiciones y orientaciones diferentes. El manual recomienda también alejarnos del punto y volver por otro sitio para que la definición sea lo más correcta posible.

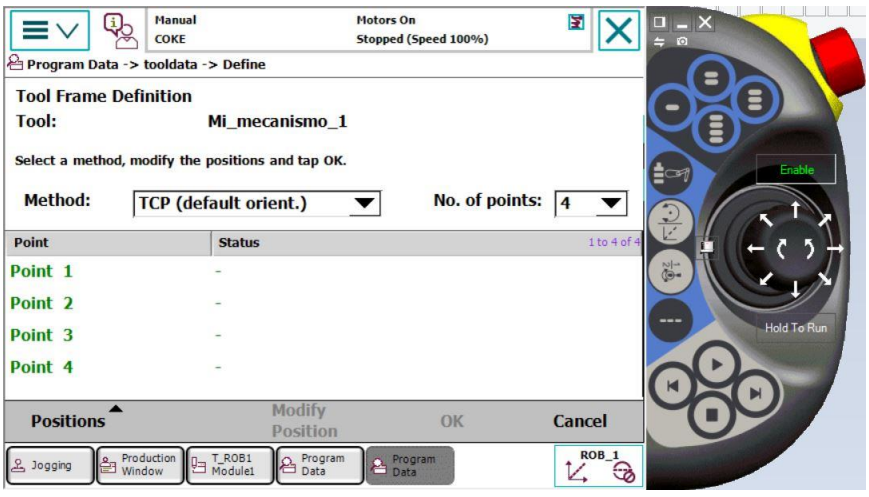


Figura 114: Definición de tooldata *Flexpendant RobotStudio*

Las siguientes imagenes muestran las diferentes posiciones del robot hacia un mismo punto para definir la herramienta por el método de los cuatro puntos.

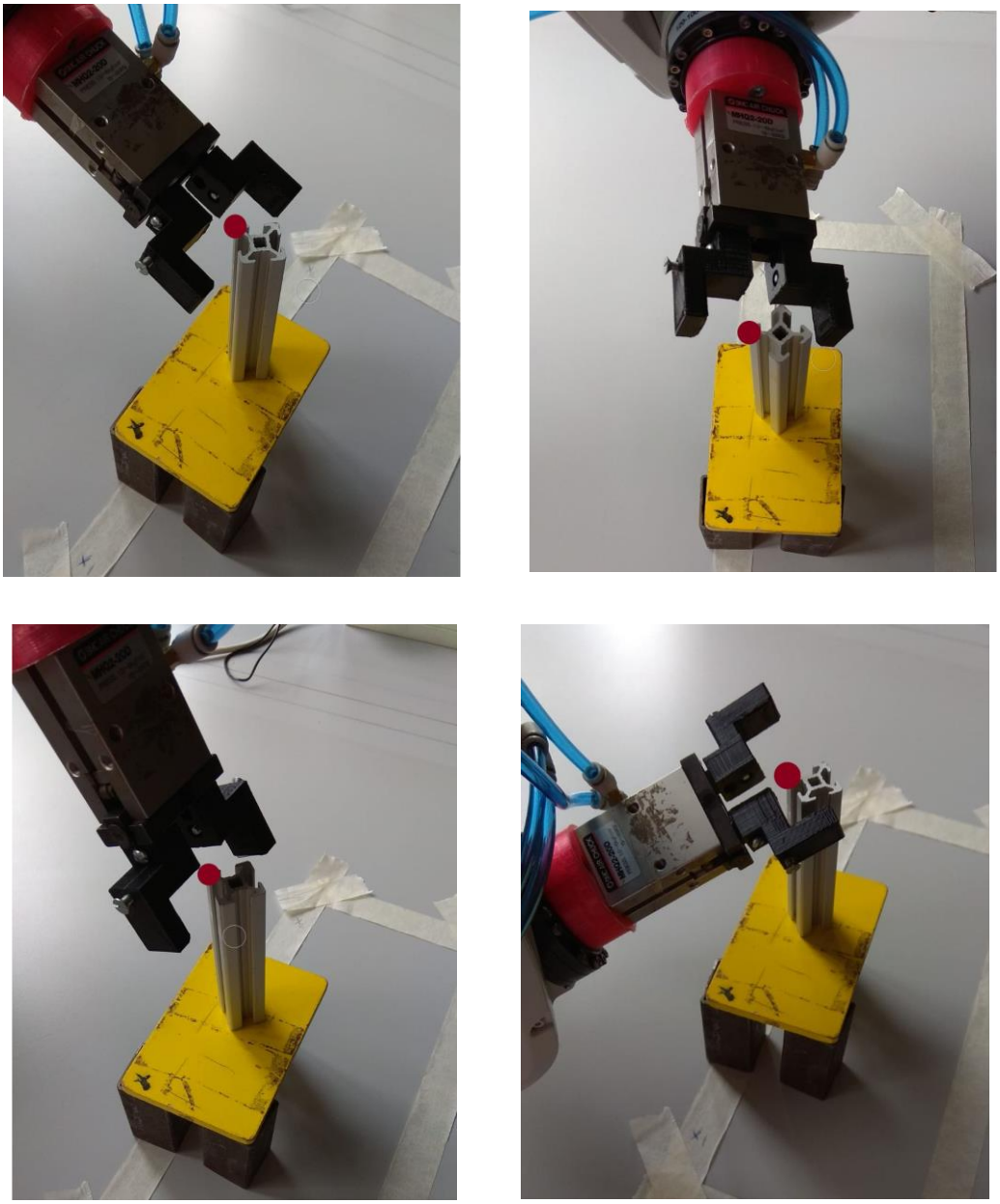


Figura 115: Posiciones para definir tooldata

Calculados los puntos le damos a *OK* y ya tendriamos nuestra herramienta completamente definida.

4.3 Definir *WorkObject* con la *FlexPendant*

Para definir un *WorkObject* se entra de igual manera en *menú>program data>wobjdata*. La pantalla que ahora aparece es de características similares a la de la creación de herramientas. Sin embargo, ahora se nos muestra los *WorkObjects* creados.

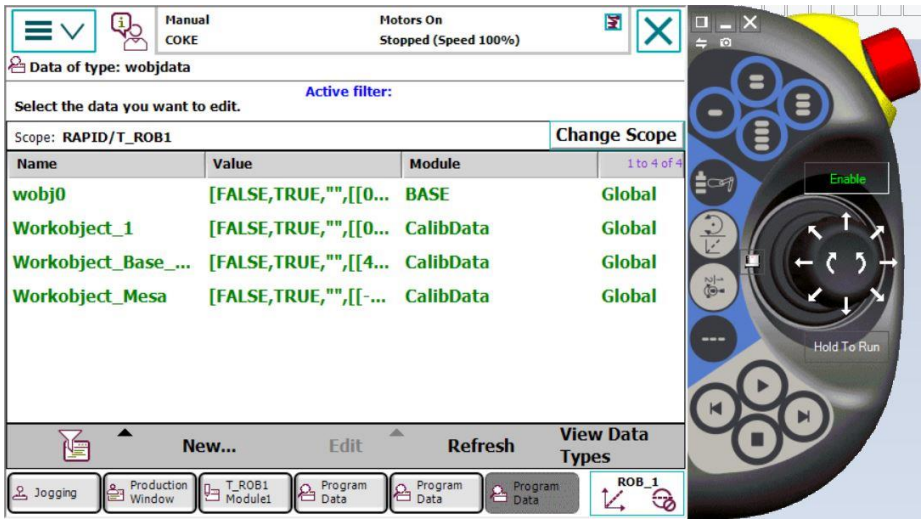


Figura 116: Ventana wobdata *FlexPendant RobotStudio*

Elegimos new y nombramos nuestro *WorkObject* y le daremos a *OK*. El resto de opciones son las mismas que para la creación de la herramienta.

Teniendo declarado el *WorkObject* ahora podemos definirlo. Para ello pulsamos *edit>define* donde nos aparecían todos los *WorkObjects* creados. A continuación se eligió usar el *Método por 3 puntos*.

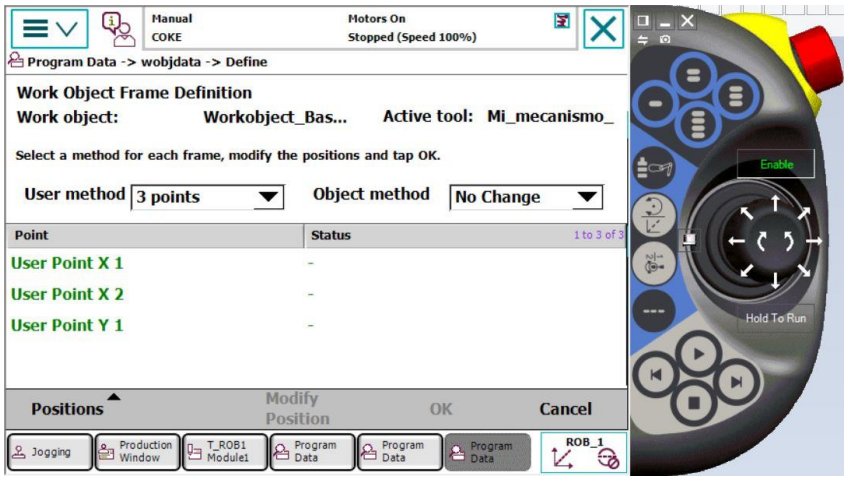


Figura 117: Ventana Define de Wobdata *FlexPendant RobotStudio*

El método crea un sistema de coordenadas cartesiano tomando como referencia dos puntos del eje *X* y otro del eje *Y*. Con estos ya puede definir el *WorkObject* (tener en cuenta la regla del tornillo o la mano derecha para saber donde aparecerá el eje Z). En el caso que nos ocupa al formar un plano el elemento base de la base y sabiendo que los ejes deben estar igual que los de la estación virtual, se lleva la herramienta a los susodichos puntos. En este apartado es especialemtne útil utilizar los movimientos rectilíneos del robot accediendo a *Jogging*.

A continuación imágenes del proceso. Incluye la comprobación de que la base es la correcta, para ello se llevó el *TCP* a la posición que debería ser nuestro origen y se comprobó en el menú *Jogging* que position referida a

la base de coordenadas creada está en (0,0,0) , basta desplazarse realizar movimientos para comprobar que los ejes también están definidos correctamente.

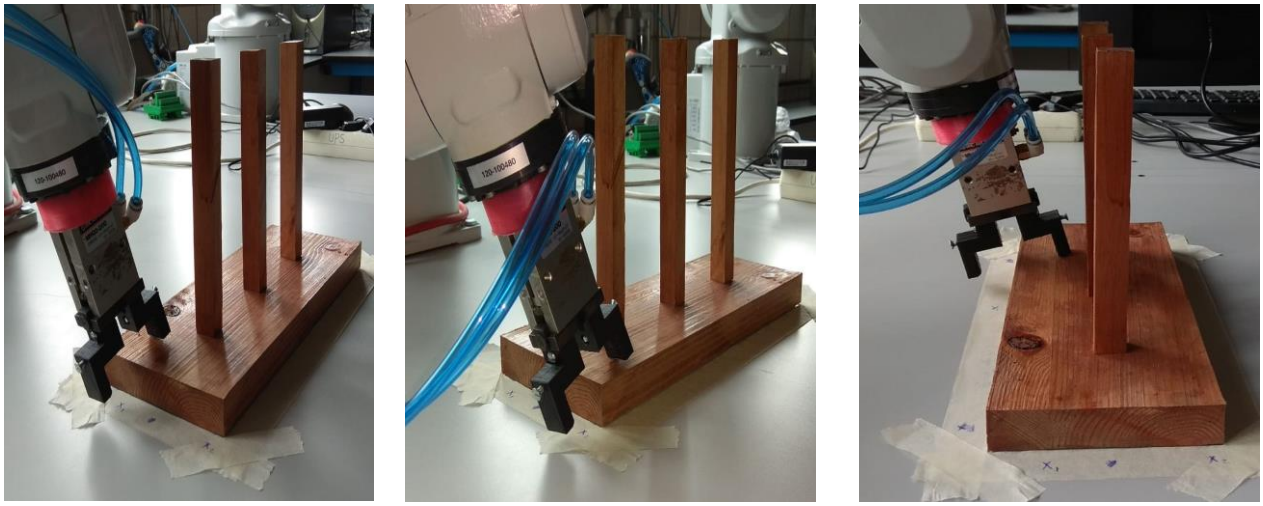


Figura 118: Posiciones para definir *WorkObject_Base*

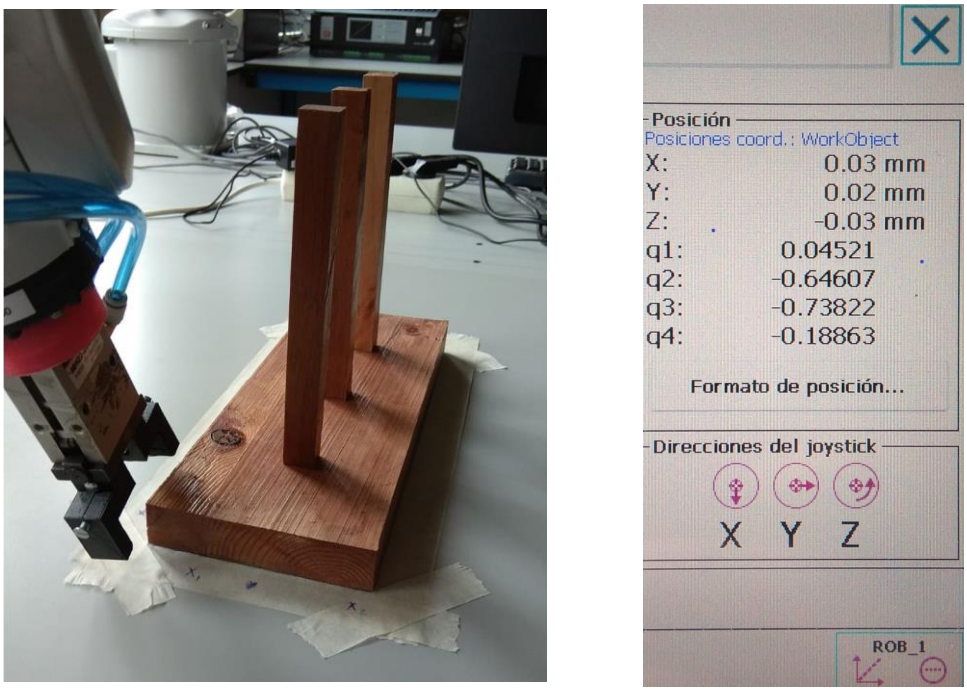


Figura 119: Comprobación origen bien situado

5 COMUNICACIÓN POR *SOCKET* ENTRE *ROBOTSTUDIO* Y *MATLAB*

Un reto que surgió a la hora de realizar el proyecto que nos ocupa fue el de comunicar a *RobotStudio* con algún otro *software* de mayor potencia matemática para la resolución del problema recursivo de las *Torres de Hanoi*. Es por ello que se pensó en la comunicación por *socket* para la cual *RobotStudio* está preparado. Como programa para resolver el algoritmo se pensó inmediatamente en *Matlab*, *software* también disponible si se posee un usuario en la *Universidad de Sevilla*, ya que es una herramienta muy potente y no tiene problemas a la hora de resolver este tipo de problemas. Además para que al usuario le sea más fácil la comunicación se creó una *GUI* (*Graphical User Interface*) donde poder seleccionar el puerto y la *IP* a la que nos queramos conectar y el número de fichas que tiene nuestro juego en ese instante.

5.1 Creación de la *GUI*

La creación de gráficos de *GUI* es bastante intuitive aunque no tanto su programación. Por lo que en este apartado se separarán ambos. Para la creación de una *GUI* se puede escribir en *Matlab* guide en la ventana *Command Window* o bien mediante *New>Graphical User Interface*.

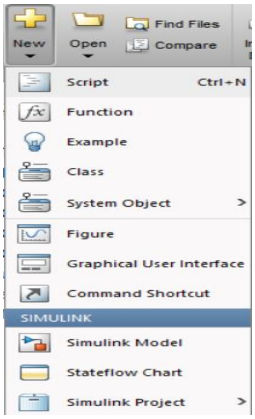


Figura 120: Selección nueva GUI *Matlab*

Se nos abre una ventana emergente donde podemos crear una nueva interfaz gráfica.

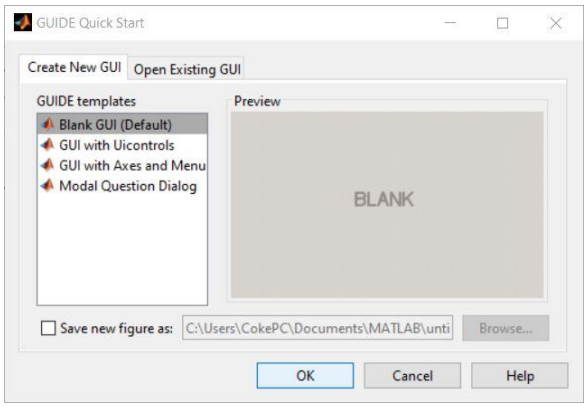


Figura 121: Ventana *GUIDE* Quick Start *Matlab*

5.1.1 *GUI*: modelado de la interfaz

Al crear una nueva *GUI* nos aparece inmediatamente la imagen siguiente, en ella podremos hacer uso de las herramientas localizadas a la izquierda para crear una interfaz que se nos antoje.

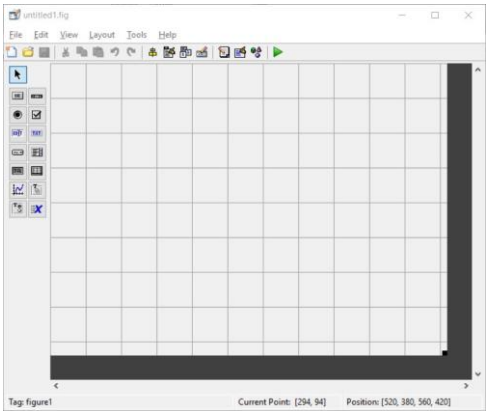


Figura 122: ventana para edición de gráficos *GUI*

Concretamente aquí se utilizaron:

- *Push bottton*: para la creación de botones conectar, desconectar y enviar.

Figura 123: Icono Push botton *Matlab*

- *Pop-up-menu*: para elegir el número de fichas de nuestro juego.

Figura 124: Icono Pop-up-menu *Matlab*

- *Edit text*: Al introducir texto en estos casilleros podremos utilizarlos. Se utiliza para obtener la dirección y puerto al que conectarse.

Figura 125: Icono Edit text *Matlab*

- *Static text*: crea cuadros de texto. Se usan para identificar los edit text creados.

Figura 126: Icono Static text *Matlab*

- *Table*: Usado para crear tablas. En nuestra interfaz se usa para representar la matriz de movimientos a realizar por el robot.

Figura 127: Icono Table *Matlab*

- *Aligne*: se usa para alinear y situar los componentes de la interfaz.

Figura 128: Icono Aligne *Matlab*

Haciendo doble click en los elementos podemos configurar sus propiedades como el nombre, el tipo de color, etc. En especial, *Tag* nos sirve para hacer referencia a nuestro objeto en la programación por lo que es importante darle un nombre identificativo.

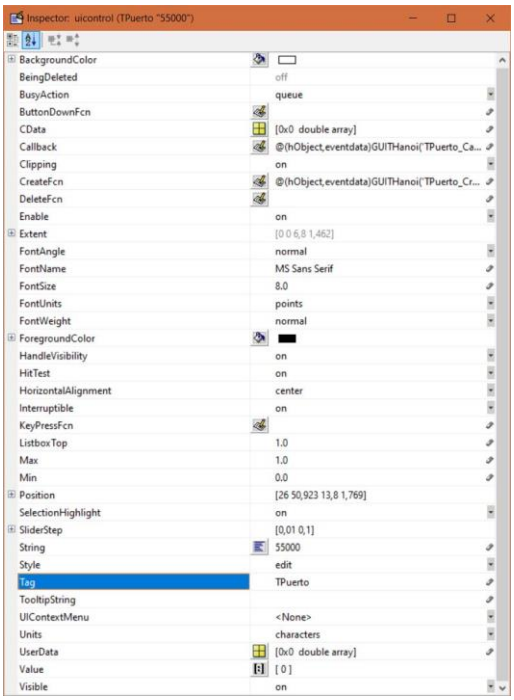


Figura 129: Ventana propiedades elementos *GUI Matlab*

Tabla 1: Objetos y sus correspondientes Tags

Objeto	Tag
Texto editable de puerto	TPuerto
Texto editable de IP	TIP
Pop-up numero de ortoedros	TnOrtoedros
Botón enviar	TEnviar
Botón Desconectar	TDesconectar
Botón conectar	TConectar
Tabla	TTable

La tabla es un elemento algo particular, para editar sus propiedades hay que hacer click derecho sobre ella y seleccionar *Table Property Editor*. En ella editamos los apartados *Columns* y *Rows* para poder mostrar la matriz de movimientos adecuadamente.

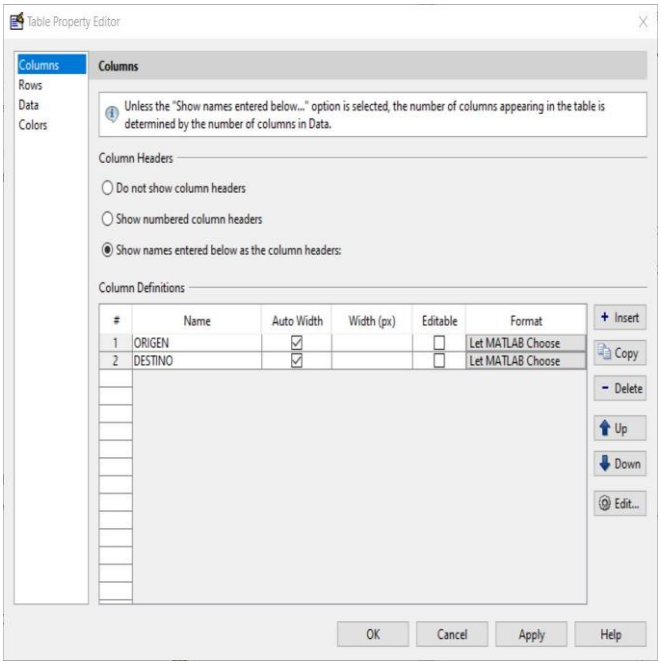


Figura 130: Ventana edición de propiedades de tablas
GUI Matlab

Una vez finalicemos los gráficos de nuestra interfaz le clicamos en la casilla run y tras darle un nombre a nuestra interfaz *Matlab* nos creará un archivo *.fig* donde almacena los gráficos y otro *.m* donde está la programación de dichos gráficos y podremos incluir acciones en ellos como veremos en el siguiente punto.

Figura 131: Icono run *GUI Matlab*

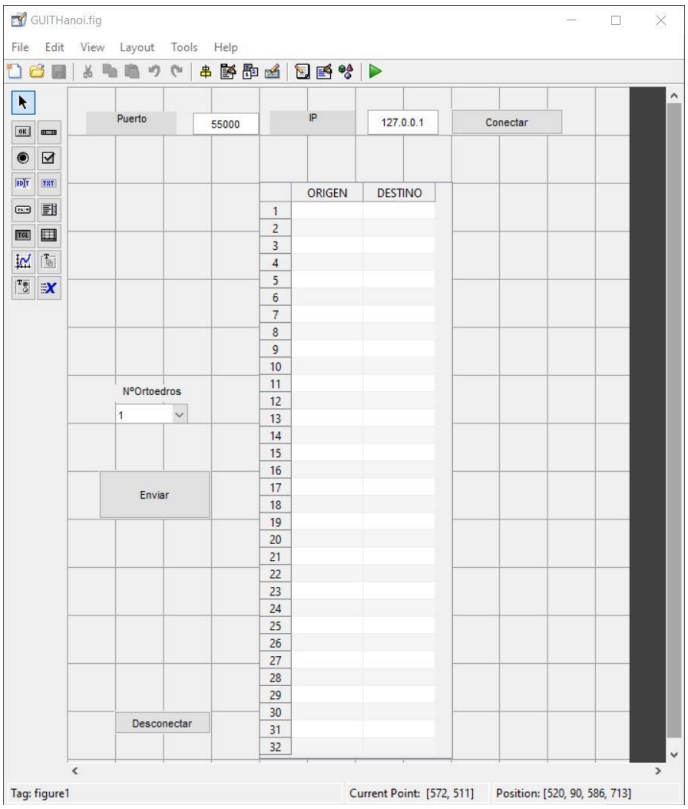


Figura 132: Interfaz *Gui Matlab* final

5.1.2 *GUI*: Programación

Creados los gráficos procedemos a programar sus acciones. Para esto necesitamos abrir el fichero *.m* que fue creado al terminar el punto anterior. Este fichero contiene la programación necesaria para que *Matlab* ejecute la ventana gráficos para el usuario. Además, está preparado para que podamos programar las acciones que realizarán los *pop-up*, los textos editables y los botones cuando los usemos.

5.1.2.1 Programación para la obtención de la *IP* y el puerto

La obtención de la *IP* y del número de puerto se hace de forma similar. Para ello, una vez localizada las funciones que llevan el nombre de sus tags, definiremos las variables globales "*IP"* y "*PORT"* donde almacenaremos la *IP* y el puerto introducidos por el usuario. Para que *IP* y puerto puedan guardar el dato introducido en la interfaz es necesario el uso de la función *get* la cual programada como *get(hObject,'String')* obtendría el valor de la propiedad *String* de nuestro objeto gráfico. El objeto gráfico es una variable tipo *Handle*

Como particularidad, el puerto al ser introducido para crear la comunicación por *socket*, necesita estar en formato *doble* para ello utilizamos la función *str2double* que nos permite transformar un *string* a *double*.

5.1.2.2 Programación de los botones conectar

Obtenida la *IP* y el puerto podemos crear un objeto *TCPIP* con este podremos realizar la conexión por *socket* utilizando la función *fopen*. Es necesario que el otro de la conexión esté ya conectado ya que si no la conexión fallaría. En nuestro caso es necesario que hayamos inicialido el Socket en nuestro programa de *RAPID* antes de conectarlo a *Matlab*. Si la conexión ha sido realizada correctamente aparecerá una ventana emergente que nos escribe un mensaje de verificación para ello se utiliza la función msgbox.

Un dato importante es que "*IP*" y "*PORT*" deben haber sido definidas como variables globales para que podamos trabajar con ellas en esta función. De igual manera, la variable *"t"* se define como global ya que será utilizada en otra función, en concreto para el envío de datos por ese *socket*.

5.1.2.3 Obtención del número de fichas del juego y escritura en la tabla

La obtención del número de piezas se hace mediante el *pop-up* que contiene una lista desde uno hasta cinco. Para obtener el valor seleccionado no podemos hacerlo como en el caso de la obtención de la *IP* o el puerto, esto es debido a que al usar *get(hObject,'String')* estaremos guardando un *cell array* de valores que contiene los *strings* 1,2,3,4 y 5. Por lo que para accede a lo que el usuario ha seleccionado en primer lugar hemos de guardar en una variable, en nuestro caso contenido, el *cell array*. Una vez hecho esto, podemos usar *get(hObject,'value')* que nos dará la ubicación del número seleccionado. Con *contenido(a)* obtendremos el *string* seleccionado. De nuevo este valor se necesita en formato *double* por lo que se hace uso de la función *str2double* y se guarda en *"n"*, también definida como *global* para su uso en otras funciones.

Ya con el valor de fichas del juego podemos obtener la matriz de movimientos del juego. Se utiliza para ello la función "*Tower"* la cual necesita el número de piezas *"n"* y en que orden están las torres, ya que podría ser que quisieramos que en vez de en la torre de la izquierda el juego comenzase en la torre de la derecha, por ejemplo.

La función "*Tower"* resuelve el problema de las *"Torres de Hanoi"* de forma recursiva .

```
function m=TOwer(n,sp,ep,ip)
m=[];
if n <=0,return;end
m=[m;Tower(n-1,sp,ip,ep)];
m=[m;sp ep];
m=[m;Tower(n-1,ip,ep,sp)];
```

Figura 133: Función revursiva para resolver las Torres de Hanoi *Matlab*

En esencia esta solución está basada en tres pasos: poner los n-1 discos de menor tamaño en la torre auxiliar, una vez montada montada la torre auxiliar poner el disco más grande en la tercera torre y por ultimo pasar la torre montada en la auxiliar al destino. Con esto hemos conseguido un problema recursivo, ya que para mover los discos de una torre a otra uno de los pasos es mover uno de los discos de una torre a otra. Como caso base que no depende de la recursividad es el que se tiene cuando sólo hay un disco en la torre origen, este caso no depende de operaciones intermedias, basta con mover el disco de la torre inicial a la final.

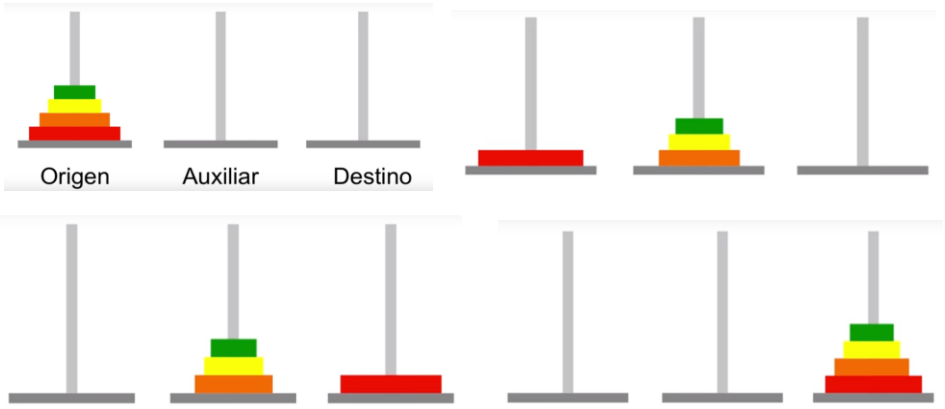


Figura 134: Explicación gráfica algoritmo

El resultado de "*Tower"* es una matriz de movimientos que será guardada en *"mMovimientos"*. Que inmediatamente será posible visualizar en la tabla creada en la interfaz. Esto viene en la última parte de la programación de este apartado, *set(handles.tTable,'data',mMovimientos*) modifica el dato de la *Table* poniendo nuestra matriz de movimientos.

5.1.2.4 Programación de botón *"enviar"*

Este último paso para el envío por Socket, necesita la matriz de movimientos, el objeto *"t"* donde viene nuestro puerto e *IP* y *"n"* el número de fichas.

En primer lugar se calculó aquí el número de movimientos que el juego necesitaría hacer según el algoritmo recursivo y se guardó en "*nmov"*.

Con *fwrite* podemos enviar por *socket* la matriz de movimiento y además el número de movimientos necesarios. Para ello antes hemos de convertirlo en formato *byte* con la función *uint8*, como particularidad, no podemos enviar la matriz numérica como tal si no en forma de vector, por lo que se decidió que fuera un vector cuya primera componente fuera en primer lugar el número de movimientos necesarios, tras ella la primera columna de la matriz de movimimientos (la que corresponde a la columna del juego donde tengo que coger la ficha a mover) y por último la segunda columna de la matriz (correspondiente a la columna del juego donde tengo que dejar la ficha). La función *fwrite* además necesita el objeto *TCPIP* en nuestro caso *"t"*.

5.1.2.5 Programación del botón desconexión

Por último, el botón de desconexión llama a la función cierre completo la cual mediante la serie de comandos de la imagen nos permite eliminar el objeto *TCPIP* y por tanto la conexión. Pese a esto aun habría que cerrar la conexión por el lado del software de *RobotStudio*.

5.1.2.6 Resultado final *GUI*

El resultado final de la *GUI* es el mostrado en las dos imagenes siguientes. En ella podemos ver como se rellena la matriz de movimimientos según el número de fichas que hayamos introducido. Esta puede ser enviada haciendo conexión y pulsando el botón "*enviar*".

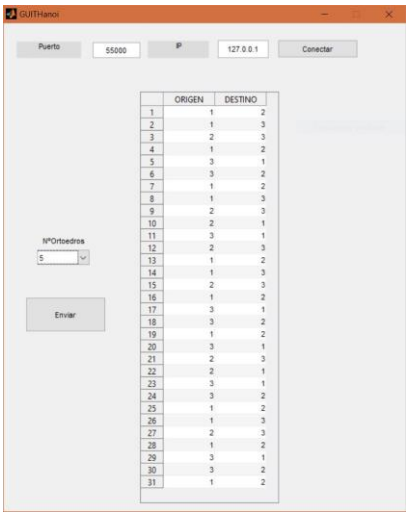


Figura 136: *GUI* con matriz para juego con 5 ortoedros

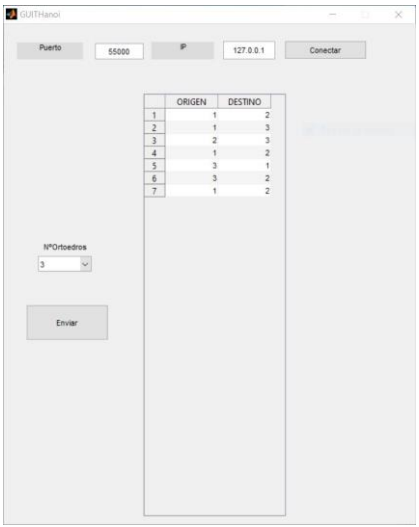


Figura 136: *GUI* con matriz para juego con 3 ortoedros

5.2 Programación en *RAPID* para la comunicación por *socket*

De igual forma que para la comunicación por *socket* por *Matlab* hemos tenido que programar teniendo encuenta las peculiaridades de *Matlab*, *RAPID* tiene sus peculiaridades que hacen que la programación no sea automática.

5.2.1 Crear el *socket* programado en escucha y aceptar conexión

En primer lugar es necesario crear la variable tipo *socket*, en nuestro caso la llamamos *"server"*, para ello se utiliza la función *CreateSocket* de *Rapid*. Además previamente en la definición de variables debimos definir la variable server como *var socketdev server*.

Tras esto es necesario conectar el *socket* a un puerto e *IP* que debe coincidir con el que el usuario introduzca en *Matlab* para realizar la conexión.

En nuestro caso "*server*" recibirá la matriz de movimientos y el número de movimientos por lo que se programa de tipo escucha.

Con *SocketAccept* server acepta la conexión con cliente, que en nuestro caso será la conexión con *Matlab*.

5.2.2 Recibir datos y desempaquetar

Con *SocketReceive* recibimos de cliente los datos y los guardamos en data que es una variable de tipo RawData propia del lenguaje *RAPID*. *RawData* es un tipo específico de *RAPID* que es capaz de almacenar cualquier tipo. Además para que el usuario pueda mandar desde Matlab los datos con tranquilidad, se usa *\Time:=WAIT_MAX* con lo que se esperarán los datos por tiempo indefinido.

Por último para tener los datos es necesario desempaquetarlos con *UnpackRawbytes* es necesario programarlo con la orden *\ASCII:=1* que nos indica que los datos desempaquetados son tipo *byte*. Debido a que rebimos el número de fichas y la matriz de movimientos, que recordemos que venía escrita en forma de vector por lo que habrá que volver a ponerla en forma de matriz, es necesario guardar los datos en su respectiva variable y localización.

5.2.3 Fin de conexión

Para finalizar la conexión con el cliente es necesario usar *SocketClose client*. De igual forma utilizando la misma llamada pero con client cerramos nuestro cliente. Cerrando por completo el *socket*.

6 PROGRAMA *RAPID* EN ESTACIÓN VIRTUAL

En los archivos adjuntos encontramos el programa escrito en *RAPID* para la estación virtual, ver *"11.6 Anexo E: Archivos programación RAPID"*. Éste es exactamente igual que el programa para el caso real salvo en el proceso movimiento debido a las necesidades del robot real.

Al inicio del módulo principal podemos encontrar la definición de todas las variables usadas, muchas de ellas ya inicializadas. Seguidamente, encontramos el proceso "*main"*, en cuyo inicio encontramos la inicialización de algunas variables. Inmediatamente después llama al proceso "*Tporsocket*" que es el encargado de realizar las operaciones oportunas para obtener la matriz de movimientos de *Matlab*. Una vez obtenida dicha matriz, procedemos a resolver la torre haciendo uso de un bucle y de los procesos "*Estado*" y "*Movimiento*".

6.1 Recibir matriz de movimientos desde *Matlab*

Este primer apartado fue descrito en el apartado "5.2.2 Recibir datos y desempaquetar". Como resumen obtuvimos la matriz de movimientos, *"T"*, que resuelve el juego a través de la conexión por *socket* entre *RobotStudio* y *Matlab*.

6.2 Proceso *"Estado"*

Es el primer proceso que realizamos al entrar en el bucle para resolver el juego. Este recibe las variables de la matriz *"T"* que indican los movimientos o, dicho de otra manera, de que torre vamos a quitar y a que torre le vamos a poner.

Los dos *If* del proceso resuelven la necesidad de saber en cada instante el estado en el que van a quedar las torres, con esto podemos saber el tamaño que tiene la pieza que vamos a mover. Éste lo almacenamos en la variable *"tamano"*.

6.3 Proceso "*Movimiento"*

Al igual que *"Estado"*, *"Movimiento"* recibe los dos postes, de uno cogemos la pieza y en el otro la soltamos. Además recibe el tamaño calculado en el porceso estado. Por lo que ya podemos realizar nuestra secuencia de movimientos.

El proceso movimiento necesita tres variables, origen , *ori*, que nos servirá para indicar a la columna que tenemos que recoger la pieza ,destino, *des*, que indica a la columna que queremos dejar la pieza y por último *tamano* que indica el tamaño de la pieza.

```
PROC Movimiento(var num ori,var num des,var num tamano)
```

Figura 137: Proceso "*Movimiento"* y parámetros a recibir

Tras su invocación, lo primero que hace el proceso son los movimientos necesarios para coger una pieza y salir por el eje en el que estaba dicha pieza. Para decidir de que columna debe coger usará la variable *ori.* Supongamos que *ori* vale 1. Entonces entrará en el *if* correspondiente y ejecutará una primera orden de aproximación paralela a la pieza, esto se hace así para evitar colisiones con el alrededor, primero se lleva a una posición segura. Al ser un movimiento con *offsets* nos estaremos moviendo en posiciones relativas al punto "*Target_Torigen*" en este caso. *dAprox* nos da la distancia de aproximación respecto a *X* del movimiento. Además la coordenada *Z* que nos da la altura a la que queremos coger será la que tenga *col{1}* que recordemos era donde se guardaban la alturas actuales de las torres. Esta altura *Z* aun no está ajustada por lo que es natural pensar que cogeremos los bloques a una altura media, por lo que restamos *tamano/2* teniendo acceso a una pieza justo por su mitad. Pese a ello, sigue haciendo falta un último ajuste ya que los dedos de la pinza son más gruesos que algunos laterales de pieza, es por ello que se añade una pequeña altura para que no pueda chocar con otros bloques o con la base cuando está cogiendo una pieza.

La segunda secuencia se basa en la primera. Una vez realizada la aproximación procedemos a hacer *Moveldo* este comando permite movernos a la posición a la que queremos coger y tras ello activar una señal, para este caso *"CierraPinza"*.

Hizo falta una espera tras esto, realizada con *WaitTime*, ya que si no inmediatamente se iba a otra posición sin tiempo a que la pinza cerrase, por lo que no llegaba del todo a la posición y había problemas de alturas. Ya con la pieza cogida procedemos a sacar la pieza del eje, se decidió a una altura de 210 mm ya que esta permitía la salida de las piezas y no era demasiada altura.

```
!Coge pieza en...
IF ori=1 THEN
    MoveJ Offs(Target_Torigen,-dAprox,0,col{1}-tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
    Moveldo offs(Target_Torigen,-dCoger,0,col{1}-tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres,CierraPinza,1;
    waitTime 1;
    movel offs(Target_Torigen,-dCoger,0,210),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
ELSEIF ori=2 THEN
    MoveL Offs(Target_Tauxiliar,-dAprox,0,col{2}-tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
    Moveldo offs(Target_Tauxiliar,-dCoger,0,col{2}-tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres,CierraPinza,1;
    waitTime 1;
    movel offs(Target_Tauxiliar,-dCoger,0,210),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
elseif ori=3 THEN
    MoveL Offs(Target_Tdestino,-dAprox,0,col{3}-tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
    Moveldo offs(Target_Tdestino,-dCoger,0,col{3}-tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres,CierraPinza,1;
    waitTime 1;
    movel offs(Target_Tdestino,-dCoger,0,210),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
ENDIF
```

Figura 138: Programación de movimiento para coger pieza

Proceso similar ocurre con la dejaba de la pieza, ahora tendremos que llevar la pieza a la posición indicada por des. Y con ella irnos desplazando por el eje para llegar a la altura que tenía *col{des}* más una altura que en principio coincidiría con la de *tamano/2* pero que al igual que para coger necesita ajuste, ya no sólo para evitar el choque con el resto de elementos si no también porque al ser la cogida más alta de la mitad provocaría que estuviéramos dejando la pieza a una altura más baja de la que deberíamos. *Moveldo* realizaría el movimiento de colocación de la base en la posición de dejada y tras ello abriría la pinza. De igual forma que en el apartado anterior volvemos a necesitar un WaitTime. Por último, salimos perpendicularme respeto al eje de la torre a una distancia de seguridad dAprox para continuar con el siguiente movimiento.

```
!Deja pieza en....
IF des=1 THEN
    Movel offs(Target_Torigen,-dCoger,0,210),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
    Moveldo offs(Target_Torigen,-dCoger,0,col{1}+tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres,CierraPinza,0;
    waitTime 1;
    MoveL Offs(Target_Torigen,-dAprox,0,col{1}+tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
ELSEIF des=2 THEN
    Movel offs(Target_Tauxiliar,-dCoger,0,210),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
    Moveldo offs(Target_Tauxiliar,-dCoger,0,col{2}+tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres,CierraPinza,0;
    waitTime 1;
    MoveL Offs(Target_Tauxiliar,-dAprox,0,col{2}+tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
elseif des=3 THEN
    Movel offs(Target_Tdestino,-dCoger,0,210),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
    Moveldo offs(Target_Tdestino,-dCoger,0,col{3}+tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres,CierraPinza,0;
    waitTime 1;
    MoveL Offs(Target_Tdestino,-dAprox,0,col{3}+tamano/2+ajuste),v100,fine,Mi_mecanismo_1\WObj:=Workobject_Base_Torres;
ENDIF
```

Figura 139: Programación de movimiento para dejar pieza

7 SIMULACIÓN

Entrando en la pestaña de simulación encontramos la siguiente botonera:

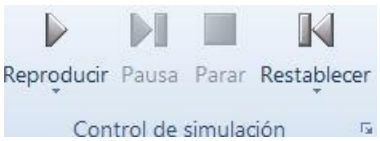


Figura 140: Botonera simulaciones *RobotStudio*

Con ella podemos ejecutar nuestro código de *RAPID* en la estación actual y en el caso de que queramos, grabar la simulación seleccionando la opción de reproducir, grabar en visor.

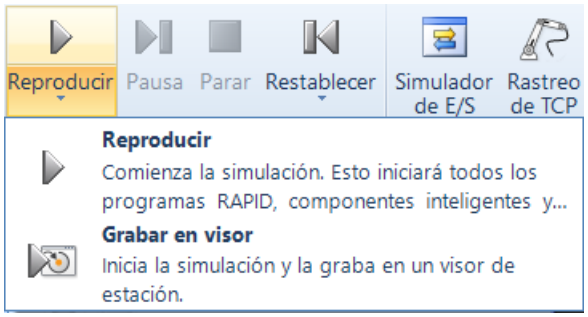


Figura 141: Opciones reproducción *RobotStudio*

Con la opción *Pausa* estando en reproducción, podemos ir pausando nuestra simulación para ver que es lo que está pasando en nuestro sistema paso a paso. Parar detiene por completo la simulación y no permite volver a reproducir, habría que restablecer y darla reproducir de nuevo.

Cuando estamos realizando ensayos sobre una situación, lo normal es que queramos volver a una situación de partida cada vez que queramos y no sea necesario volver a colocar todos nuestros componentes de nuevo. *Restablecer* permite al programador guardar una situación en la estación de forma que pueda volver cuantas veces quiera a ella sin problema. E incluso también podemos tener guardadas diferentes posiciones lo cual también es de gran utilidad. Pulsando en las opciones de *Restablecer* podemos guardar el estado actual para futuros usos.

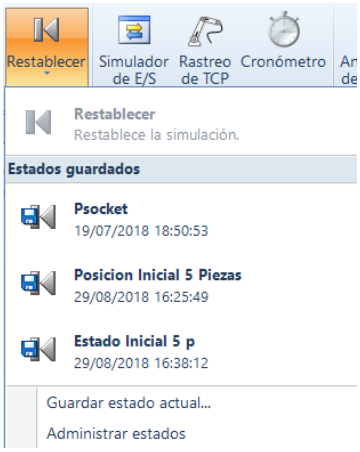


Figura 142: Opciones restablecer *RobotStudio*

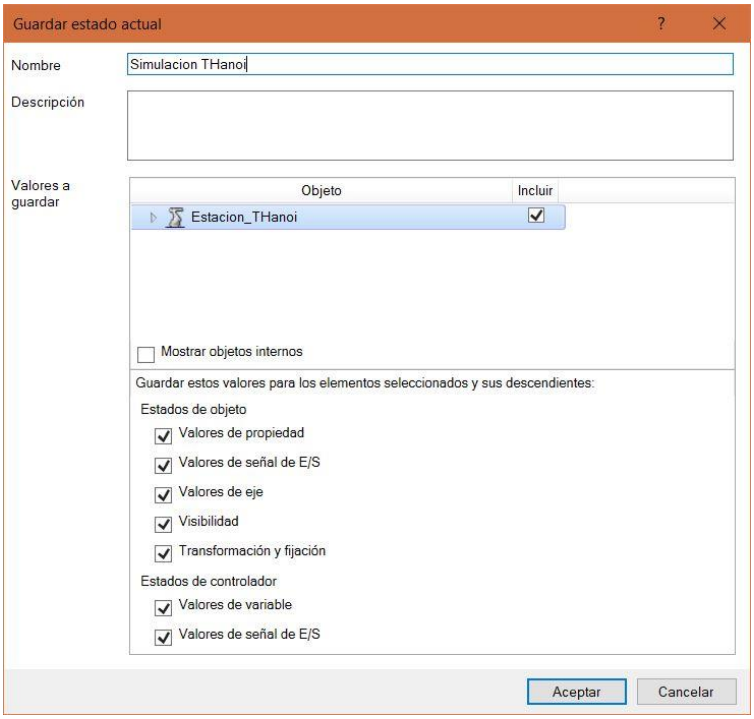


Figura 143: Ventana guardar estado actual *RobotStudio*

Por último en la simulación se hizo uso de los conjuntos de colisiones, creando dichos conjuntos podemos ver si los diferentes objetos de nuestra estación chocan entre ellos.

Figura 144: Crear conjunto
de colisiones Robotstudio

Una vez clickado, nos aparecerá unas carpetas donde debemos llevar los objetos que queremos ver si colisionan entre si, por ejemplo la *"Garra_Lab"* y el ortoedro de 20 mm.

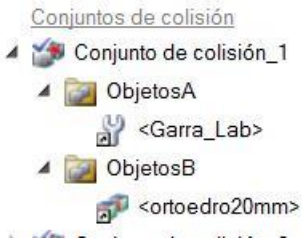


Figura 145: Conjuntos de
colisiones *RobotStudio*

Una vez activada la simulación nos mostrará las colisiones existentes.

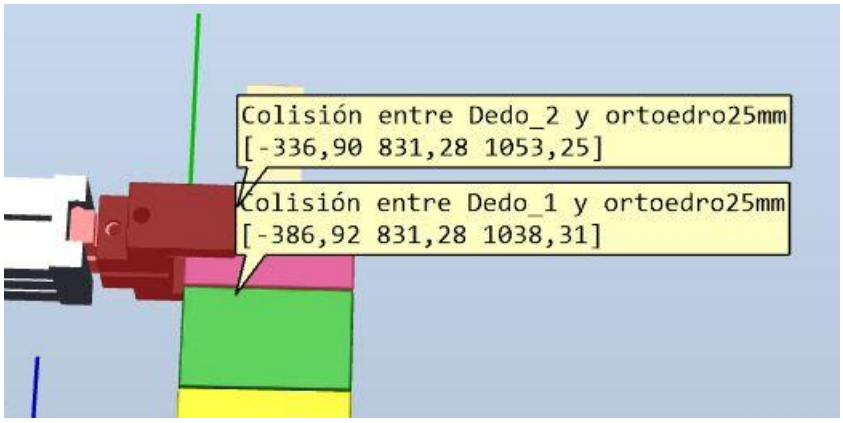


Figura 146: Simulación con conjunto de colisiones *RobotStudio*

8 PROGRAMA PARA ROBOT REAL

La programación del robot real debería ser simplemente pasar el código *RAPID* de simulación a la controladora directamente. Pero hay una serie de diferencias respecto al diseño 3D de gran importancia. En primer existe una diferencia entre el *WorkObject* real e imaginario, por lo que debemos ajustar el movimiento atendiendo estas difrencias. Además la pendiente de los ejes de las torres presentan cierta inclinación por lo que el movimiento de los ortoedros por ellas no es una vertical si no que presenta cierta pendiente. Por último, la pinza real coge las piezas con mucha fuerza haciendo impredecible su forma de dejada y por tanto la de cogida en movimientos posteriores. Es por ello que fue necesario el diseño de una estrategia de recolocación con la que poder coger las fichas sin problema.

8.1 Cálculo de pendientes de los ejes de las torres

Para medir las desviaciones que presenta los ejes de las torres se hizo uso de la pinza en modo jogging con el ortoedro de 20 mm de altura cogido. Haciendo pasar el bloque por los postes ajustando el movimiento de forma que no choque se pudo calcular las pendientes. Por ejemplo, en el primero de los postes se situó un ortoedro sujeto por la garra y vimos que una posición centrada del ortoedro respecto al poste era de *X=35.3 Y=70* a una altura de 15 mm. Estas medidas son las del *TCP* de la herramienta y nos la da la *Flexpendant*. Comenzamos a subir por el poste y nos paramos en una altura de 45 mm. Aquí podemos observer que el ortoedro ha dejado de estar centrado por lo que volvemos a situarlo y una posición que parece favorable es la *X=34.3 Y=70*. Una vez más continuamos y al llegar a una altura de 120 mm vuelve a estar descentrada y una posición acceptable es la de *X=31.3* e *Y=70*. Haciendo una interpolación podríamos generar una recta que recorriese el eje inclinado, pero en este caso es incluso más facil ya que los puntos pertenecen a una recta situada en el plano *Y=70*.

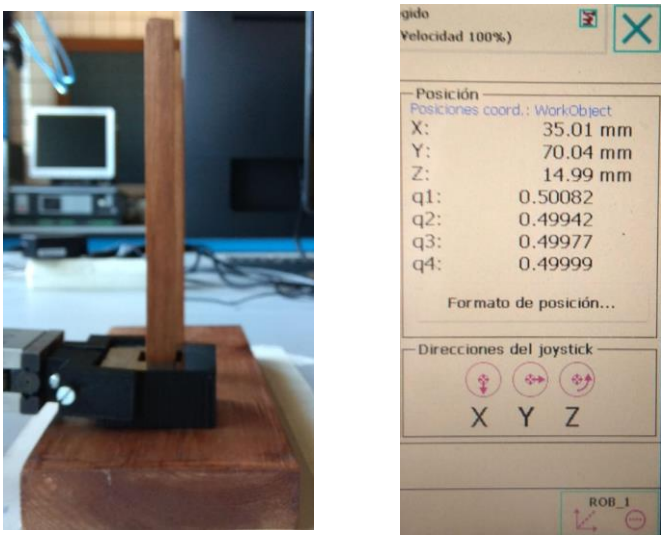


Figura 147: Robot inicia movimientos verticales sobre eje origen

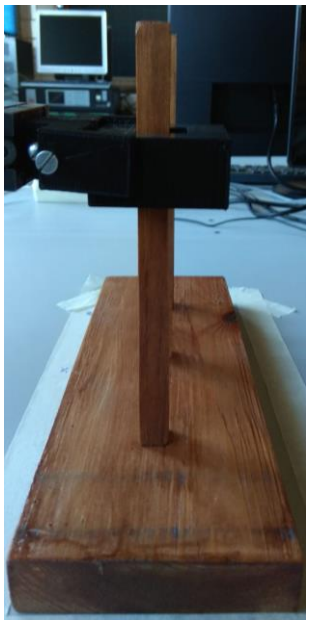
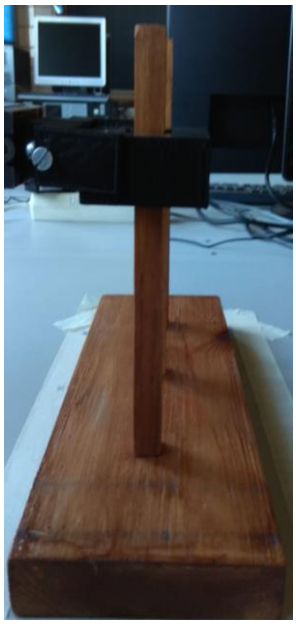


Figura 149: Colisión con eje al desplazarse verticalmente



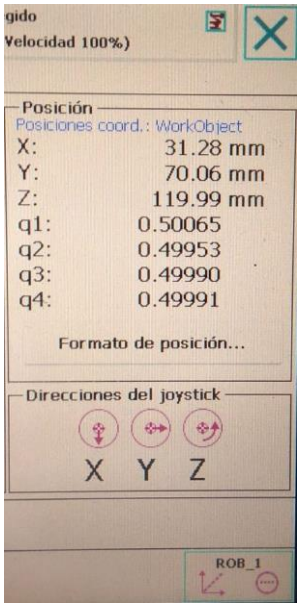


Figura 149: Reajuste tras colisión

De esta forma hemos obtenido la inclinación de los ejes de torre, pese a ello aun existe una pequeña desviación entre las pendientes de las rectas. Por lo que en el laboratorio se hicieron pequeños ajustes para el correcto funcionamiento.

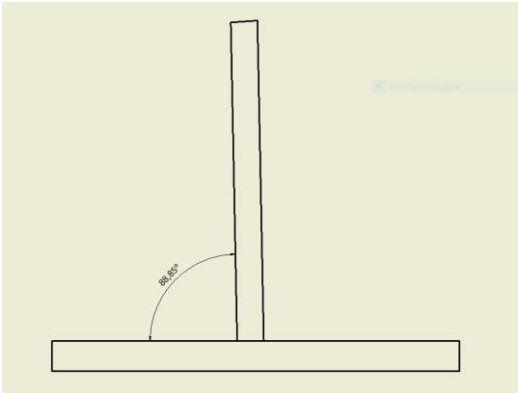


Figura 150: Inclinación eje

Tras realizar los cálculos necesarios se obtuvieron las respectivas rectas de cada eje.

Eje Origen	$X = -11.25 - 0.025 * (Z - 15)$
Eje Destino	$X = -20.85 - 0.022 * (Z - 15)$
Eje Auxiliar	$X = -12.5 - 0.01 * (Z - 15)$

Tabla 2: Rectas iniciales de cada eje

Pese a ello en la programación se tuvieron que ajustar un poco quedando finalmente como:

Eje Origen	$X = -8.75 - 0.025 * (Z - 15)$
Eje Destino	$X = -16.5 - 0.22 * (Z - 15)$
Eje Auxiliar	$X = -12.5 - 0.02 * (Z - 15)$

Tabla 3: Rectas de salida tras corrección para cada eje

Sabiendo ahora la inclinación, en la programamación el movimiento de los bloques por los ejes de torre no es una recta vertical, si no una recta con una leve desviación respecto a la vertical. Es por ello que según la altura a la que vayamos a poner la pieza vamos a llevarla a una posición *X* diferente.

La cogida de la pieza al ser inexta y generar un ángulo y el plano paralelo a la mesa, obliga a que la bajada de las piezas también tenga que ser corregida para que no haya ningún contacto.

Eje Origen	$X = -10.75 - 0.025 * (Z - 15)$
Eje Destino	$X = -17.5 - 0.22 * (Z - 15)$
Eje Auxiliar	$X = 12.5 - 0.02 * (Z - 15)$

Tabla 4: Rectas de entrada tras corrección para cada eje

8.2 Estrategia de recolocación para cogida de piezas

Debido a la fuerza con la que la pinza agarra y suelta las piezas fue necesario realizar una estrategia de recolocación para poder coger la pieza sin problema. Para ello se pensó que lo mejor era ayudarnos de los postes de la base. En primer lugar llevariamos la pieza descolocada al poste recuperando así su paralelismo respecto a la pinza.Tras ello se moverá la pieza hacia un lado hasta que choque con el poste de la base de nuevo y una vez estamos en esa posición obligatoriamente fija, procedemos a su centralización. Una vez centrada basta mover la pinza a una posición de cogida también centrada de esta forma cogemos siempre la pieza de la misma forma.

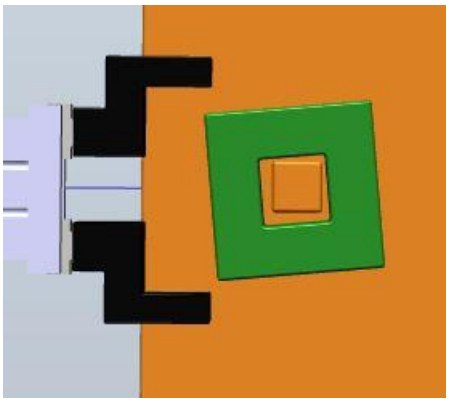


Figura 151: Bloque mal situado para cogida

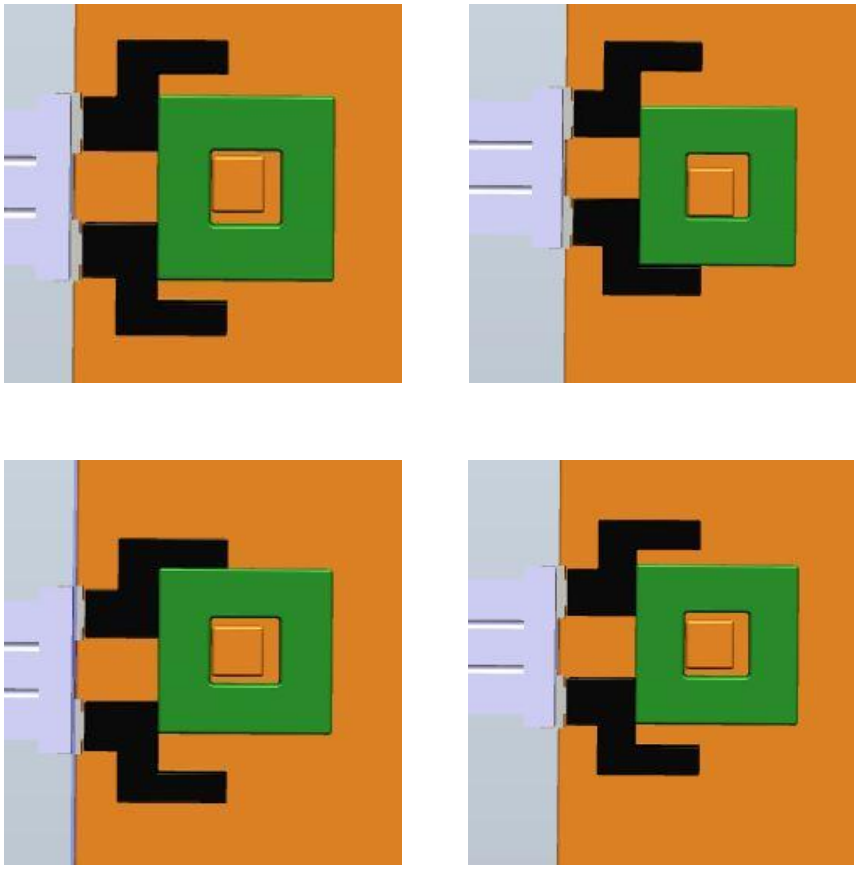


Figura 152: Movimientos de recolocación para cogida

Si la pienza cogiese y dejase las fichas con suficiente suavidad podriamos prescindir de este método. Incluso de la base en si misma, ya que podriamos mover las torres de un sitio a otro con muchísima precisión.

En "*11.4 Anexo E: Archivos programación RAPID*" si abrimos el programa *"Programa_Base_Real_Con_Reposicionamiento"* encontramos el proceso recolocar que es el encargado de realizar esta operación.

Un problema añadido, debido a que la cogida y suelte de la pieza es muy brusco de nuevo, es que pueden acabar las piezas con distinta inclinación respecto a la horizontal de la mesa. Por lo que de nuevo en el código no podemos coger las piezas a una altura media como en nuestra programación en el ideal si no que para las distintas alturas de pieza tendremos que cogerlas a una altura ensayada.

8.3 Programa para robot real

El programa es exactamente el mismo que el de la estación virtual, salvo el proceso de movimiento que tuvo que ser cambiado debido a los errores anteriores. Además la altura de cogida de las piezas no podía ser la misma que en la estación virtual, ya que de nuevo la fuerza de las pinzas no permite coger con toda la exactitud que se debiera los ortoedros.

La programación de los movimientos en el robot real según la matriz que llega desde *Matlab* se basa en dos tipos de procesos, uno es coger la pieza de cierta torre y el otro mover a cierta torre. Haciendo uso de las alturas que le llegan desde la matriz de estados comentadas en el apartado *"6.2 Proceso "Estado""* el programa es capaz de mover las fichas de la forma correspondiente haciendo uso de las procesos para cogida de piezas. Además precisan los cambios necesarios mencionados anteriormente así como algunos leves ajustes a la hora de entrar a dejar una pieza debido a la falta de exactitude a la hora de coger las piezas. Esta última modificación se hizo por ajuste visual.

9 CAMBIOS EN LA ESTACIÓN PARA SIMULAR RECOLOCACIÓN

El mecanismo utilizado para accionar la herramienta hace que ésta coja y suelte las piezas coja las piezas con demasiadas imprecisiones y por ello tuvimos que crear un proceso en la programación que se encargase de la recolocación de las piezas para llevarla a una posición de cogida óptima. Es por ello que se decidió hacer un último ajuste en la estación que emula la base real y hacer que en ésta se simule también este mecanismo de recolocación. Sin embargo, esta tarea nos obliga a modificar el *SmartComponent*, ya que no está preparado para que sus dedos sirvan para ajustar una pieza a cierta posición en posición de abierto.

Atendiendo a este problema creamos un nuevo *SmartComponent* a partir del anterior al que agregamos tres plane sensor con los cuales podemos emular como al hacer contacto con los dedos de la garra se van moviendo los bloques.

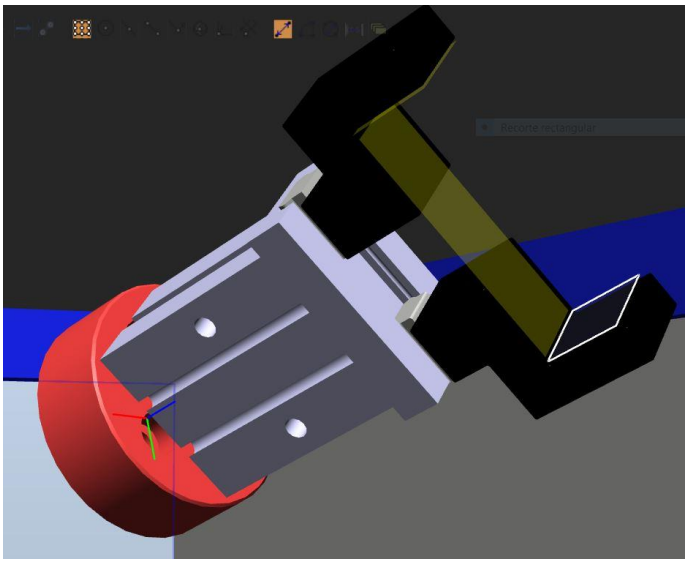


Figura 153: *SmartComponent* para mover bloques sin cerrar *RobotStudio*

De forma similar al apartado *"3.2 Creación de SmartComponent y lógica de la estación"* debemos diseñar el *SmartComponent* de forma que al estar activo uno de los *PlaneSensor* seamos capaces de arrastrar la pieza, y una vez que desactivemos liberemos la pieza.

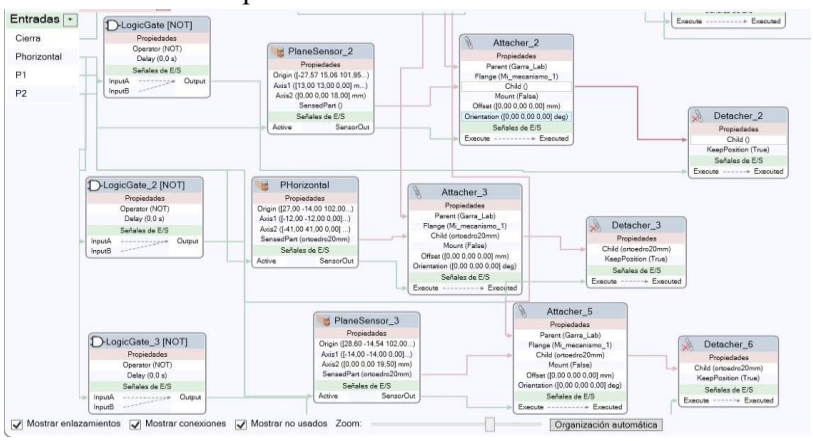


Figura 154: Diseño de *PlaneSensors* para mover piezas *RobotStudio*

Recordemos también que es necesaria la creación de las señales que activen nuestros plane sensor así como reajustar la lógica de la estación.

P1	Digital Output	d652	P1	2		All
P2	Digital Output	d652	P2	3		All
PANEL24OVLD	Digital Input	PANEL	Overload Panelboard 24V	30	safety	ReadOnly
PANFAN	Digital Input	PANEL	Supervision of Main Computer FAN	29	safety	ReadOnly
Phorizontal	Digital Output	d652	Phorizontal	1		All

Figura 155: Señales para *PlaneSensor Robotstudio*

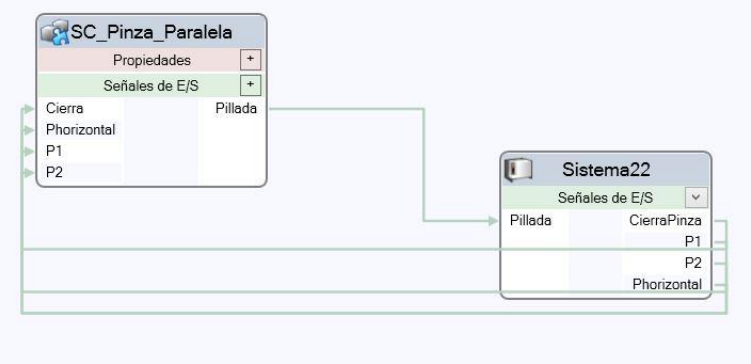


Figura 156: Lógica de la estación *RobotStudio*

De esta forma ya podemos programar los movimientos de recolocación de nuestra pieza haciendo uso de la activación de señales.

10 CONCLUSIONES

Tras la conclusión de este trabajo de fin de grado, se pueden sacar distintas conclusiones.

- El carácter de este proyecto hace que el creador tenga que aprender a moverse en un entorno con bastantes similitudes a las asignaturas dadas en la escuela, pero a su vez es necesario una gran adquisición de conocimientos para poder desarrollar todo el contenido.
- El diseño de los juegos y piezas tuvo gran importancia en el desarrollo del proyecto. Resulta muy interesante tener a disposición una impresora 3D ya que abarata mucho la creación de piezas pudiendo de esta forma ensayar con nuestros modelos de forma rápida y económica.
- Todo el trabajo desarrollado con *ABB* ha sido de gran utilidad, ya que en cierta manera se ha aprendido un oficio. De hecho ahora se tiene gran habilidad con el entorno y el desarrollo de otros proyectos requeriría muchísimo menos tiempo de aprendizaje. Prueba de ello es el último ajuste del SmartComponent para el cual se tardó un tiempo irrelevante respecto a sus primeras componsiciones.
- Poder llevar la simulación a un sistema real con todos los problemas que esto puede llegar a dar, es una experiencia muy valiosa. En los sistemas virtuales todo funciona a la perfección una vez programado, pero una vez queramos implantar nuestros sistemas siempre hay que tener en cuenta los pequeños errores que hacen a un sistema real único e irrepetible. Es por ello que en la actualidad muchos los trabajadores que se dedican a la producción en masa en la actualidad no tratan de resolver los problemas de forma inteligentes, directamente van punto a punto programando trayectorias.

Siendo estos los puntos concluyentes más relevantes, se puede decir que la experiencia en su elaboración ha sido bastante satisfactoria.

11 BIBLIOGRAFÍA

- José Luis Pozo Acosta, «Adaptación física y electrónica de un actuador eléctrico a un robot manipulador», Trabajo Fin de Máster, 2016.
- Agustín Ramos Hurtado, « Diseño, programación y simulación de estaciones robotizadas industriales con Robotstudio», Trabajo Fin de Grado, 2016.
- Mª Luisa Fernández Iglesias, «Diseño y fabricación de Garras para un IRB120 », Trabajo Fin de Grado, 2017.
- Azahara Gutiérrez Corbacho, «Desarrollo de una interfaz para el control del robot IRB120 desde Matlab», Trabajo Fin de Grado, 2014.
- Fabricante ABB, «Procedimientos iniciales-IRC5 y RobotStudio », manual del operador.
- Fabricante ABB, «IRC5 con FlexPendant», manual del operador.
- Fabricante SMC, «Parallel Type Air Gripper Series MHZ», catálogo.
- Fabricante SMC, «Pinza eléctrica de 2 dedos / con cubierta antipolvo. Serie LEHZJ», datasheet.
- Apuntes de la asignatura *Control y Programación de Robots* de *4º GIERM*, *Universidad de Sevilla*.
- Peter So, «*course materials for 2.003J / 1.053J Dynamics and Control I*», Fall 2007. MIT OpenCourseWare (http://ocw.mit.edu), Massachusetts Institute of Technology. Downloaded on [DD Month YYYY].

11 ANEXO

11.1 Anexo A: Planos

En la carpeta *"Diseños_Planos_y_Formatos"* adjunta en el disco del proyecto se encuentran los ficheros *.dwg* y *.pdf* de todos los elementos de la estación.

- *"Modelos_De_Juegos"* contiene a su vez la carpeta con las distintas soluciones planteadas desde el inicio.
    - *"Solucion_Con_Cilindros">"Juegos_Cilindros_Planos".*
    - *"Solucion_Con_Gorritos_Para_Pinza_Angular">"Juegos_Gorritos_Planos".*
    - *"Solucion_Con_Ortoedros">"Juego_Ortoedros_Planos".*
- *"Pinza_Paralela_50mm">"Dedo_Pinza_50mm"* tiene los planos de las piezas creadas dentro de las dos piezas creadas.
    - *"Adaptador"* encontramos los planos del adaptador.
    - *Dedo_Pinza_Paralela_50mm"* encontramos los planos del dedo creado.
- *"Mesa"* contiene los planos de mesa.
- *"Base_Madera_Real"* contiene los planos de la base de madera real.

11.2 Anexo B: Archivos para *Autodesk Inventor*

En la carpeta *"Diseños_Planos_y_Formatos"* se dispone de los ficheros de los distintos componentes del proyecto en format *.ipt.*

- *"Modelos_De_Juegos"* contiene a su vez la carpeta con las distintas soluciones planteadas desde el inicio.
    - *"Solucion_Con_Cilindros">"Juegos_Cilindros_ipt".*
    - *"Solucion_Con_Gorritos_Para_Pinza_Angular">"Juegos_Gorritos_ ipt".*
    - *"Solucion_Con_Ortoedros">"Juego_Ortoedros_ ipt".*
- *"Pinza_Paralela_50mm">"Dedo_Pinza_50mm"* tiene los diseño *.ipt* creadas dentro de las dos carpetas separadas.
    - *"Adaptador"* encontramos el *.ipt* del adaptador.
    - *Dedo_Pinza_Paralela_50mm"* encontramos el *.ipt* del dedo creado.
- *"Mesa"* contiene el *.ipt* de mesa.
- *"Base_Madera_Real"* contiene el *.ipt* de la base de madera real.

11.3 Anexo C: Archivos para impresión 3D

En la carpeta *"Diseños_Planos_y_Formatos"* se dispone de los archivos *.stl* mandados a imprimir.

- *"Modelos_De_Juegos">"Solucion_Con_Ortoedros">"Juego_Ortoedros_Formato_stl"* se encuentran los ortoedros para la impression, en format *.stl*.
- *"Pinza_Paralela_50_mm">"Dedo_Pinza_Paralela_50mm" .stl* del dedo de la pinza.

11.4 Anexo D: Archivos de componentes para *RobotStudio*

En la carpeta *"Diseños_Planos_y_Formatos"* se dispone de los archivos *.sat* mandados a imprimir.

- *"Modelos_De_Juegos">"Solucion_Con_Ortoedros">"Juego_Ortoedros_Formato_sat"* se encuentran los ortoedros en formato *.sat*.
- *"Pinza_Paralela_50_mm">"Dedo_Pinza_Paralela_50mm" .sat* del dedo de la pinza.
- *"Pinza_Paralela_50_mm">"Adaptador" .sat* del dedo de la pinza.
- *"Mesa"* contiene el *.sat* de mesa.
- *"Base_Madera_Real"* contiene el *.sat* de la base de madera real.

11.5 Anexo E: Estaciones *RobotStudio* y *SmartComponent*.

En la carpeta *"Estaciones_RobotStudio"* se pueden encontrar las estaciones creadas así como los dos *SmartComponent* finales.

- *"Estacion_Base_Ideal"* contiene la estación para el caso ideal.
- *"Estacion_Base_Real"* contiene la estación para el caso real sin reposicionamiento.
- *"Estacion_Base_Real_Reposicionamiento"* contiene la estación para el caso real con reposicionamiento.
- *"SmartComponent_Caso_Ideal"* contiene el *SmartComponent* creado para el caso ideal sin la opción de reposicionamiento.
- *"SmartComponent_Caso_Real_Recolocacion"* contiene el *SmartComponent* creado para el caso ideal sin la opción de reposicionamiento.

11.6 Anexo F: Archivos programación *RAPID*

En la carpeta *"Programas_RAPID_RobotStudio"* encontramos los distintos programas creados para cada estación, recordemos que hay que cargarlos antes de iniciar la simulación según cada caso.

- *"Programa_Base_Ideal"* contiene la programación para el caso de la base ideal.
- *"Programa_Base_Real"* contiene la programación para el caso de la base ideal.
- *"Programa_Base_Real_Con_Reposicionamiento"* contiene la programación para el caso de la base ideal.

11.7 Anexo G: Archivos GUI

En la carpeta *"GUI_Matlab"* se encuentran los documentos necesarios para correr la GUI. *"GUITHanoi.m"* es el que se debe correr para inicilizar la interfaz gráfica de usurio.

11.8 Anexo H: Memoria en format *word*

En la carpeta *"Memoria"* se aloja la memoria en formato word.

11.9 Anexo I: Especificaciones *IRB120*

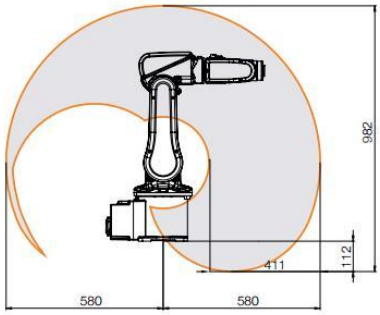
Specification			
Variants	**Reach**	**Payload**	**Armload**
IRB 120-3/0.6	580 mm	3 kg (4kg)*	0.3 kg

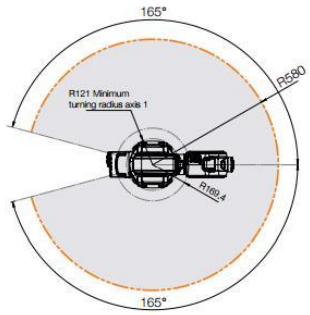
Features	
Integrated signal supply	10 signals on wrist
Integrated air supply	4 air on wrist (5 bar)
Position repeatability	0.01 mm
Robot mounting	Any angle
Degree of protection	IP30
Controllers	IRC5 Compact / IRC5 Single cabinet

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

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

Working range at wrist center & load diagram



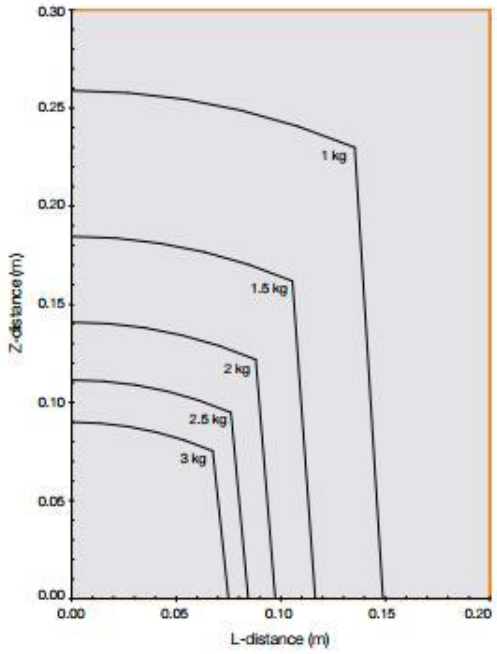


Electrical connections	
Supply voltage	200–600 V, 50/60 Hz
Rated power Transformer rating	3.0 kVA
Power consumption	0.25 kW

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

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

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



11.10 Anexo J: Vínculos *Autodesk Inventor*

- Curso de Autodesk Inventor en *Youtube*

  https://www.youtube.com/watch?v=fg02yUwyVkY
- Foro Autodesk

  https://forums.autodesk.com/t5/inventor-mechanical-electrical/bd-p/323

11.11 Anexo K: Vínculos *RobotStudio*

- Curso de *RobotStudio* en *Youtube*

  https://www.youtube.com/watch?v=4l2FT-MdqZU&list=PL7fQXFqChkTe0snNKx67OHyaZxYfdlRBz
- Curso de *RobotStudio* en *Youtube*

  https://www.youtube.com/watch?v=oDnHaVMZjp4&list=PLVdvHpsfqw1bX194j7Slup6ri5se2668p
- Página principal *ABB*

  https://new.abb.com/es
- Foro *ABB*

  https://forums.robotstudio.com/
- Página desarrolladores *ABB*

  http://developercenter.robotstudio.com/

11.12 Anexo L: Vínculos *GUI*

- Curso de *GUI Matlab* en *Youtube*

  https://www.youtube.com/watch?v=S4xR2DjedG8&list=PLjApqg2Zsw31HHJONnV9IQkxTQfUrW_9F
- Página principal Matlab

  https://es.mathworks.com/?s_tid=gn_logo